ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN PARA IDENTIFICAR INFORMACIÓN RELEVANTE EN VIGILANCIA ESTRATÉGICA

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO MENCION EN INGENIERA EN SISTEMAS INFORMATICOS Y DE COMPUTACION

FREDDY ALEXANDER CORONEL FLORES

Freddy.coronel@epn.edu.ec

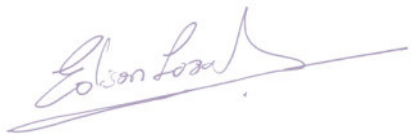
DIRECTOR: EDISON FERNANDO LOZA AGUIRRE, PhD

Edison.loza@epn.edu.ec

Quito, mayo 2021

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Freddy Alexander Coronel Flores, bajo mi supervisión.



Edison Fernando Loza Aguirre

DIRECTOR DE PROYECTO

DECLARACIÓN

Yo Freddy Alexander Coronel Flores, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Freddy Alexander Coronel Flores

Contenido

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN

Con el objetivo de estar atentos ante eventos externos que puedan afectar decisiones a nivel estratégico, las organizaciones utilizan métodos basados en Vigilancia Estratégica para analizar y tomar decisiones en base a información externa disponible en artículos noticiarios. Sin embargo, la existencia de grandes cantidades de artículos hace que el proceso de Vigilancia Estratégica sea ineficiente e incluso tedioso para el personal encargado del proceso. En este sentido, en el presente proyecto se propone un sistema de recomendación para, en base a similitud de documentos, poder proveer artículos de interés con el objetivo de ayudar en el proceso de Vigilancia Estratégica y podrá ser integrado con otras soluciones ya propuestas para afrontar la problemática de sobrecarga de información. Este sistema ha sido desarrollado considerando nuevas tecnologías y procesos de desarrollo de software recomendados para este tipo de sistemas informáticos.

Este sistema ha sido evaluado tanto en un ambiente offline, para prevenir inconsistencias en el código y los componentes que lo conforman, como online con un grupo de personas tomando en cuenta modelos de evaluación de aceptabilidad de tecnología: utilidad percibida y facilidad de uso. Sobre esta evaluación cabe destacar que un 75% de las personas consideraron que el sistema ofrece facilidad de uso y un 100% afirmaron que el sistema cumple con su propósito. Sin embargo, también se obtuvo retroalimentación acerca de las mejoras que se pueden aplicar al sistema con el objetivo de que el usuario pueda interactuar de manera más efectiva y así poder ayudar con el proceso de Vigilancia Estratégica.

Palabras clave: Vigilancia Estratégica, Sistemas de Recomendación basados en contenidos, perfil de usuario, artículos, sobrecarga de información.

ABSTRACT

To be attentive to external events that may affect decisions at a strategic level, organizations use methods based on Strategic Vigilance to analyse and make decisions based on these external events that may be available in news articles. However, the existence of large quantities of articles makes the Strategic Vigilance process inefficient and even tedious for the personnel in charge of the process. In this sense, in this project a Recommendation System is proposed to, based on the similarity of documents, be able to provide articles of interest to help in the Strategic Vigilance process and may be integrated with other solutions already proposed to face the information overload problem. This system has been developed considering new technologies and recommended software development processes for this type of computer systems.

This system has been evaluated both in an offline environment, to prevent inconsistencies in the code and its components, and online with a group of people considering technology acceptability evaluation models: perceived utility and ease of use. Regarding this evaluation, it should be noted that 75% of the people considered that the system offers ease of use and 100% affirmed that the system fulfils its purpose. However, feedback was also obtained about the improvements that can be applied to the system in order that the user can interact more effectively and thus be able to help with the Strategic Vigilance process.

keywords: Strategic Vigilance, Recommendation Systems, content-based, user profile, articles, information overload.

1. MARCO TEORICO

1.1 Introducción

Hoy en día, la robustez y la competitividad de las organizaciones se encuentran consolidadas por su capacidad para crear valor mediante la gestión de la información y el conocimiento [1]. Mediante esta gestión, las organizaciones pueden tomar decisiones de nivel estratégico. Así, mediante actividades de Vigilancia Estratégica (*Strategic Scanning*, SScan) las organizaciones pueden analizar información relevante que se encuentra en su entorno; la cual, por lo general, tiene un alto valor estratégico. Por ejemplo, una organización puede aprovechar la información de artículos que se encuentran en la Web que le pueden permitir a estar alerta sobre innovaciones susceptibles de crear oportunidades o amenazas que afecten su posicionamiento estratégico [2]. Así, un sistema de SScan es la solución a la necesidad de conocer y entender el estado del entorno externo a la organización, gracias a la recolección y análisis de información que pueda ser útil para esta [3].

No obstante, un problema relacionado con la SScan radica en que no toda la información, recolectada de la Web, o que una organización tiene almacenada en sus bases, es relevante para la organización en un momento dado. Más aún, mucha de la información recolectada suele ser ignorada al momento de realizar un análisis estratégico debido, en gran medida, a la ausencia de mecanismos que permitan recomendar información adicional relacionada con aquella que está siendo analizada en un momento dado. Es por esto como, en el presente trabajo, se propone el desarrollo de un sistema de recomendación que permita sugerir información que esté almacenada en la base de conocimientos de la organización y que esté conectada con los tópicos de interés de la organización en un momento dado.

Siendo los CEO quiénes deben dar sentido a la información obtenida por la SScan, el interés de nuestro trabajo radica en asegurar que el tiempo de búsqueda y recomendación de artículos adecuados sea el óptimo. Pudiendo así, brindar una solución a la falta de tiempo de los CEO para analizar la información, lo cual ha sido reportado como una de las razones por las cuales los jefes ejecutivos no realizaban una correcta implementación de la SScan [3].

Así, con el sistema de recomendación implementado se podrá:

- Reducir el tiempo de búsqueda de artículos relevantes para una organización.
- Reducir el tiempo de los CEO para analizar documentos relevantes para SScan.
- Recomendar documentos que se encuentran dentro de los tópicos de interés de una empresa.
- Estar al tanto de patrones o tendencias que se encuentran en los documentos.

1.2 Objetivos

1.2.1 Objetivo General

Desarrollar un sistema de recomendación para identificar información relevante en vigilancia estratégica utilizando algoritmos de recomendación basados en contenidos.

1.2.2 Objetivos Específicos

Para cumplir con el objetivo general, se ha propuesto un conjunto de objetivos específicos los cuales se listan a continuación:

- Buscar características que representen a un documento, de tal manera de que pueda ser manipulado por el algoritmo de recomendación.
- Implementar una representación vectorial de un documento en base a las frecuencias de las palabras.
- Identificar el mejor enfoque para la recomendación de la información.
- Implementar un algoritmo de recomendación en base al conjunto de documentos para realizar la recomendación en un sistema.
- Evaluar el desempeño del algoritmo de recomendación en base a un conjunto de artículos para la prueba del sistema.

1.3 Alcance y limitaciones

La creciente demanda de aplicaciones Web y la alta gama de frameworks de desarrollo Web que juegan un papel importante en la rápida implementación de aplicaciones [4], nos ha motivado para que nuestro sistema de recomendación se integre con un cliente Web en donde el usuario podrá interactuar con artículos de noticias, recibir las recomendaciones en base a

un histórico de artículos los cuales han sido de su interés y búsqueda mediante palabras clave. Este cliente se limitará a la Web y no a aplicaciones móviles o de escritorio.

En cuanto al sistema de recomendación, se decidió que el tipo de recomendador a implementar sea el basado en contenidos. En nuestro caso se generarán recomendaciones a partir de la similitud de documentos [5] [6] usando un modelo de Procesamiento de Lenguaje Natural disponible mediante la librería SpaCy [7] . En el proceso de recomendar artículos se tendrá en cuenta la representación vectorial generada por el modelo de SpaCy, para calcular, mediante el coseno de similitud, que tan similares son dos documentos en base al ángulo que existe entre dos vectores que los represente. Este recomendador se implementará en el Backend mediante un esquema de GraphQL para la API y Django como framework principal. Además, se tomará en cuenta un conjunto existente de documentos, especialmente los relacionadas con el COVID-19, que serán parte del sistema.

El sistema de recomendación no integrará manejo de ratings de los artículos realizados por los usuarios, por lo tanto, no se usará algoritmos que se incluyen en los Sistemas de Recomendación basados en filtrado colaborativo (collaborative filtering) [8]. Además, no se implementarán distintos paradigmas de Procesamiento de Natural de Lenguaje Natural como LDA o LSA, para la generación de recomendaciones [9]. Además, se implementará un buscador sencillo que ayudará en la búsqueda de artículos en base de palabras clave.

Una vez determinado el alcance de nuestro proyecto, debemos tener en cuenta la teoría del problema que se va a atacar (SScan), las etapas de recolección y preanálisis de información, así como también lo referente a la teoría de sistema de recomendación y las herramientas que se utilizarán.

1.4 Marco Teórico

1.4.1 Vigilancia Estratégica

Las organizaciones conviven en un entorno socioeconómico en el cual también conviven otras organizaciones, sus clientes y/o proveedores. En este contexto, toda organización necesita poder extraer y analizar la información externa que se encuentra en su entorno. Es aquí donde se implementa un sistema de Vigilancia Estratégica (SScan) [3], el cual recolecta la

información externa del entorno, la almacena y la procesa para asistir a la toma de decisiones a nivel estratégico y generar acciones que mejoren su nivel competitivo.

El proceso de SScan se puede resumir como una serie de actividades que empieza desde la definición de las necesidades informacionales de la organización, la identificación y recolección de información relevante hasta la toma de decisiones en base al análisis realizado [10] (Figura 1). De todos los pasos del proceso de SScan, nuestro sistema se integrará en las etapas de selección y preanálisis de información, las cuales se describen en las siguientes secciones.

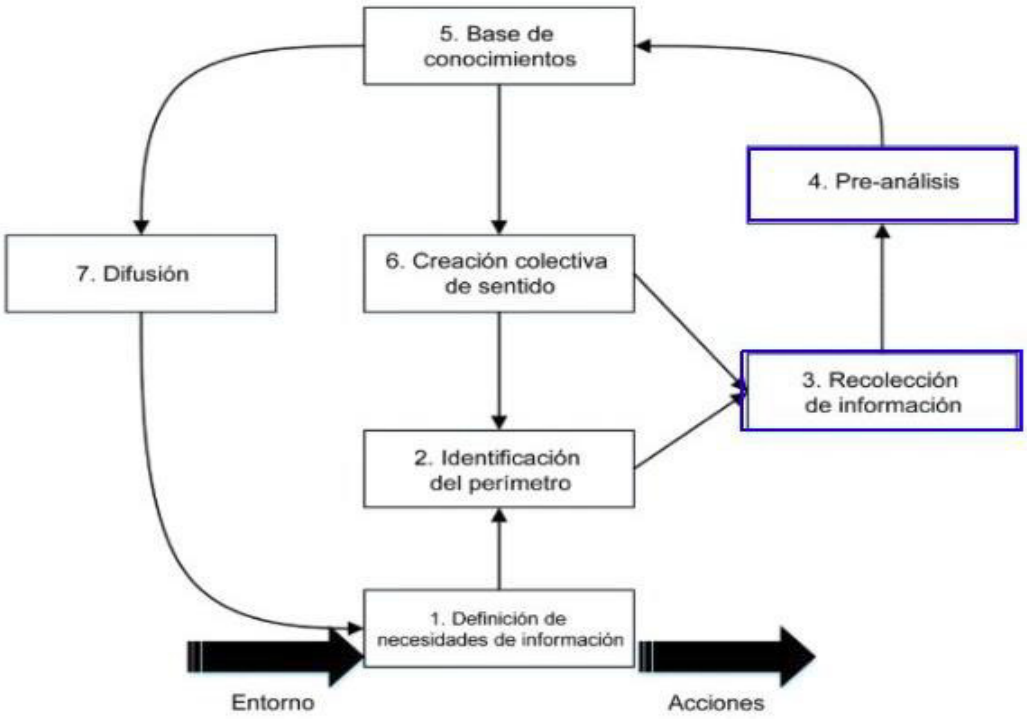


Figura 1. Proceso de Vigilancia Estratégica, se resaltan las etapas de Recolección de Información y Preanálisis.

Recolección de Información

En esta etapa, miembros de la organización recolectan información que puede ser extraída directamente de fuentes como Internet, bases de datos o de clientes, proveedores o la competencia. Dado que la cantidad de información en formato digital se ha incrementado rápidamente en los últimos años, la carga de trabajo necesaria para su análisis por parte de los gerentes también se ha incrementado [11]. En el pasado, se han propuesto diversos

métodos y herramientas para extracción de información de páginas Web mediante Spiders que navegan y la extraen de la Web. Otra herramienta útil para la búsqueda y recolección de información ha sido mediante el uso de motores de búsqueda como Google o Yahoo! [11].

Así, en esta fase existen dos tipos de recolectores que ayudan a obtención de información de diversas fuentes [12]:

- Los recolectores sedentarios, que trabajan generalmente en un escritorio y que acceden a fuentes de información formales (bases de datos, internet, etc.).
- Los recolectores de terreno (móviles, itinerantes, nómadas) que se encuentran frecuentemente en desplazamiento y en contacto con actores diversos: clientes, proveedores, competidores, etc. Sus reportes corresponden a informaciones especialmente sensoriales (auditivas, visuales, olfativas, táctiles, gustativas) captadas en la fuente misma.

Preanálisis de Información

Corresponde a todas las operaciones que permiten retener únicamente la información de SScan susceptible de interesar a los utilizadores potenciales de la misma en nuestra organización. Esta etapa de preanálisis puede además aportar con nuevos elementos que permitan mejorar las tareas de adquisición de información y de identificación del perímetro de búsqueda, como por ejemplo la definición de nuevos actores, temas o palabras claves.

Uno de los principales criterios de selección de la información reside en el carácter anticipativo de la misma [10]. Es decir, existe un interés preponderante por la información que permite desvelar eventos o circunstancias futuras, y no aquellos que ya han sucedido. Criterios secundarios se relacionan con la pertinencia de la información al perímetro definido para el dispositivo de SScan.

Sin embargo, el incremento de la cantidad de información recolectada por nuevas tecnologías implica una sobrecarga de información y a su vez, un proceso de SScan inefectivo [13] [14] [15]. Es en este caso que un Sistema de Recomendación podrá ayudar a mitigar el problema de sobrecarga en las etapas de recolección y preanálisis de información del proceso SScan. En la siguiente sección se explica qué es un Sistema de Recomendación y sus diferentes tipos

para poder entender de mejor manera cómo la implementación de este sistema puede ayudar a resolver el problema descrito previamente.

1.4.2 Sistemas de Recomendación

Considere que un usuario visita un restaurante, el cual tiene a disposición un gran conjunto de platos y bebidas. El usuario interactúa con un mesero, al cual le pide sugerencias sobre los platos que este ya ha degustado en el pasado. El mesero, al ya conocer los gustos anteriores de este usuario, sugiere una lista que puede ser de agrado. Sin embargo, el usuario pregunta qué platos han sido los más populares entre otros clientes del mismo restaurante, de nuevo, el mesero recomienda otra lista basándose en sus conocimientos de otros usuarios. Un Sistema de Recomendación actúa como el mesero que conoce los gustos de todos los clientes del restaurante y sugiere una lista de platos a cada uno de estos para que los disfruten. Estos tipos de sistemas no comprenden únicamente la implementación de un algoritmo, sino implican el entender los datos y los usuarios con los cuales interactúa para así poder proveer sugerencias [5]. Estas recomendaciones se basan frecuentemente en las interacciones previas entre usuarios e ítems; esto debido a que los ítems que han sido de interés para el usuario en el pasado, son buenos indicadores para establecer qué ítems pueden ser de interés para el usuario en el futuro [8].

1.4.3 Objetivos de un Sistema de Recomendación

De Acuerdo con Aggarwal [8] los objetivos de un sistema de recomendación son:

a. Predicción de Ratings: Para *m* usuarios y *n* ítems se tiene una matriz *m* x *n*, en donde los valores especificados (u observados) son usados para el entrenamiento. Los valores no presentes (no observados) serán calculados por un algoritmo. Teniendo en cuenta lo anterior, un Sistema de Recomendación, calculará ratings basándose en interacciones de usuarios e ítems.
b. Clasificación del tipo de problema: En la práctica, no se requiere, necesariamente, predecir los ratings de usuarios a ítems con el fin de realizar recomendaciones. Así. un sistema recomendador, puede obtener los *k* ítems más relevantes para un usuario, o determinar los más similares a otro ítem.
c. Relevancia: Un sistema recomendador debe ser capaz de sugerir ítems que son relevantes a los usuarios.

d. Novedad (Novelty): Un Sistema de Recomendación debe ser capaz de sugerir ítems que sean relevantes al usuario, pero que son desconocidos para el usuario.
e. Serendipia (Serendipity): Se refiere a la capacidad de recomendar ítems al usuario de manera sorpresiva de acuerdo con la relevancia del ítem.
f. Incrementar la diversidad de las recomendaciones: Cuando se sugiere una gran cantidad de recomendaciones de diferentes tipos de ítems, se tiene una gran probabilidad de que el usuario encuentra ítems relevantes.

1.4.4 Tipos de Sistemas de Recomendación

Un Sistema de Recomendación está supeditado a la interacción entre usuarios e ítems, que pueden ser desde películas hasta artículos en texto como noticias. Sin embargo, para cada tipo de ítem a recomendar, se han propuesto diferentes tipos de Sistemas de Recomendación, los mismos que se ajustan a un problema en específico; como sugerir películas basándose en ratings o artículos de noticias basándose en similitud entre documentos [8]. A continuación, se lista los diferentes tipos:

- Sistemas de Recomendación basados en Filtrado Colaborativo
    - Métodos Basados en Memoria.
        - Filtrado Colaborativo basado en Usuarios.
        - Filtrado Colaborativo basado en ítems.
    - Métodos basados en Modelos.
        - Modelos Bayesianos.
        - Modelos basados en Reglas.
- Sistemas de Recomendación basados en Contenidos.
- Sistemas de Recomendación basados en Conocimientos.

1.4.5 Sistemas de Recomendación basados en Filtrado colaborativo

Utiliza usuarios y ratings de ítems para poder generar recomendaciones. El principal reto es diseñar los métodos de filtración colaborativa para que, mediante matrices se pueda calcular ratings de ítems de los cuales no se tiene un rating o cuyos valores no están disponibles en la matriz. Esta matriz se obtiene del conjunto de usuarios e ítems que han sido calificados por cada uno de los usuarios. En filtrado colaborativo se utiliza la similitud entre usuarios para

poder calcular ratings y así generar recomendaciones. Considere los usuarios e ítems que se observan en la Figura 2, en este caso películas; el Sistema de Recomendación basado en Filtrado Colaborativo tratará de predecir, en este caso, los valores vacíos (aquellos en gris) en base de la similitud entre usuarios para así poder encontrar el rating y, a su vez, poder recomendar los ítems más relevantes o con un rating mayor [5].

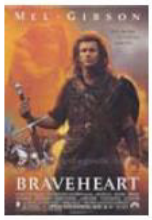
						
	Comedy	**Action**	**Comedy**	**Action**	**Drama**	**Drama**
Sara	5	3		2	2	2
Jesper	4	3	4		3	3
Therese	5	2	5	2	1	1
Helle	3	5	3		1	1
Pietro	3	3	3	2	4	5
Ekaterina	2	3	2	3	5	5

Figura 2. Matriz de recomendaciones en métodos basados en Filtrado Colaborativo.

Para este tipo de sistemas, existen dos métodos que permiten poder calcular rating y generar recomendaciones [8]:

Métodos basados en Memoria: Comúnmente conocidos como algoritmos de filtrado colaborativo basado en vecindarios. En este método los ratings de la combinación usuario-ítem son calculados en base a sus vecinos. Estos vecindarios pueden ser definidos de dos maneras:

- **Filtrado Colaborativo basado en Usuarios**: En este caso, los ratings son proveídos por usuarios similares a un usuario A y se utilizan para generar recomendaciones para A. La idea básica es determinar los usuarios que

comparten similitud con A y predecir ratings para ítems que pueden ser relevantes para A.

- **Filtrado Colaborativo basado en ítems:** Para poder calcular ratings de un usuario A hacia un ítem B, el primer paso es determinar un conjunto S de ítems que son similares al ítem objetivo B. Este conjunto se obtiene de ítems que el usuario A ha encontrado relevante en el pasado, y en base a esto se puede determinar si el ítem B es relevante o no para A.

Métodos Basados en Modelos: Se utiliza modelos predictivos basados en Machine Learning y minería de datos. Ejemplos de estos modelos son: arboles de decisión, modelos basados en reglas, métodos bayesianos, etc [8].

1.4.6 Sistemas de Recomendación Basados en Contenidos

En este tipo de Sistemas de Recomendación, se utiliza los atributos descriptivos de los ítems para generar recomendaciones. El término "contenido" se refiere a un conjunto de atributos propios de un tipo de ítem. En estos tipos de sistemas, los ratings y comportamientos del usuario son combinados con la información descriptiva disponible de los ítems. Los métodos de recomendación basados en contenido tienen la ventaja de generar recomendaciones para nuevos ítems, cuando los datos de ratings no están disponibles para el ítem en cuestión. Por lo tanto, un modelo recomendará estos ítems nuevos a usuarios basándose en su similitud entre ítems [8].

Considere que un usuario ha disfrutado la película *Ex Machina*, la cual tiene temática de robots que cambian a un comportamiento agresivo. El sistema buscará y generará recomendaciones en base a la similitud entre películas con el mismo género o temática a *Ex Machina* y en base a esto se sugerirá al usuario una lista del top *k* de películas más similares. Este proceso se lo puede observar en la Figura 3 [5].

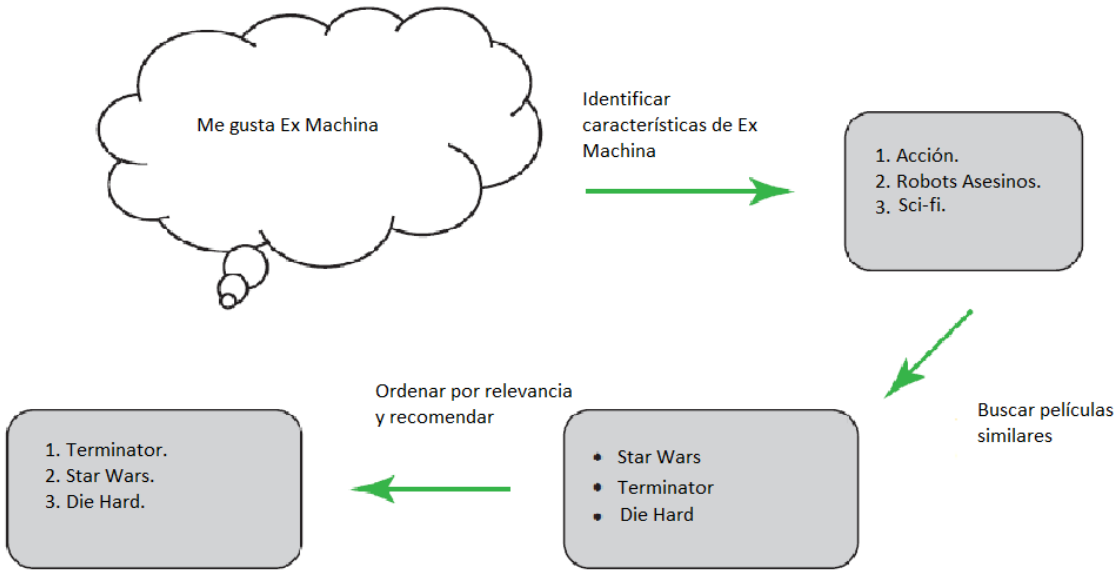


Figura 3. Proceso de generar recomendaciones en base a contenidos *[5]*.

1.4.7 Sistemas de Recomendación basados en Conocimiento

Estos tipos de sistemas se utilizan al recomendar ítems que son personalizables, que pueden cambiar con el tiempo o de los que no se tiene la suficiente cantidad de ratings disponibles. Por ejemplo, tours de turismo, autos, casas etc. En este caso se debe tener en cuenta los requerimientos del usuario y poder generar recomendaciones en base a cómo los ítems se ajustan a estos requerimientos. Por ejemplo, un usuario puede degustar la combinación de características de un automóvil como requerimientos de entrada. El sistema debe ser capaz de usar estos requerimientos para encontrar automóviles similares y sugerir una lista de los más relevantes [8].

En general, los Sistemas de Recomendación han sido adaptados para su uso en aplicaciones Web. Así, de acuerdo con Falk [5] “la Web es el lugar en donde un sistema de recomendación vive”. En la siguiente sección se discutirá las tecnologías de desarrollo Web como el modelo cliente-servidor, las Interfaces de programación de aplicaciones hasta las tecnologías Web usadas para la implementación de nuestro sistema de recomendación.

1.4.8 Desarrollo Web

Una aplicación Web es un programa de computadora que realiza una función específica usando un navegador Web como su cliente [16]. La aplicación puede ser sencilla como una que solo guarda notas hasta una más compleja como un ERP Web. La mayoría de las aplicaciones Web de hoy en día funcionan mediante un modelo cliente-servidor. El cliente es la aplicación en la cual el usuario interactúa y el servidor es en donde la información se almacena y procesa.

1.4.9 Protocolo HTTP

Es el protocolo el cual permite obtener recursos, como documentos HTML (documentos de HyperTexto). Este protocolo cliente-servidor ha sido la piedra angular del intercambio de datos y recursos en la Web [17]. La compartición de recursos lo realiza el servidor al enviar un Response (respuesta) y es iniciado mediante un Request (petición) por parte del cliente en un navegador Web. HTTP es un protocolo de capa de transporte y se envía sobre TCP o sobre una conexión TSL encriptada sobre TCP. Debido a su extensibilidad, este protocolo no es solamente usado para enviar documentos de HyperTexto, sino también para imágenes y videos.

1.4.10 Modelo Cliente Servidor

En este modelo se introducen dos roles que en realidad son procesos: el rol de cliente y el servidor [18]. El servidor provee servicios hacia uno o más clientes que se comunican mediante el uso de protocolos, como es el caso de HTTP en las aplicaciones Web. Por otro lado, el cliente es el encargado de consumir los servicios del servidor y presentarlos hacia el usuario mediante una interfaz gráfica. En Desarrollo Web, el cliente es la aplicación Web con la cual el usuario interactúa (navegador Web).

En HTTP, el navegador Web siempre inicia las peticiones al servidor y nunca, al contrario. Para presentar una página Web o aplicación Web, el navegador envía una petición al servidor para obtener el documento HTML que representa a la página Web mediante etiquetas, luego realiza un análisis sintáctico al documento y peticiones adicionales que corresponden a scripts de ejecución, estilos CSS y otros recursos usados para armar la página Web en el navegador para que el usuario pueda visualizar. Esta interacción entre el cliente y servidor puede ser observada en la Figura 4.

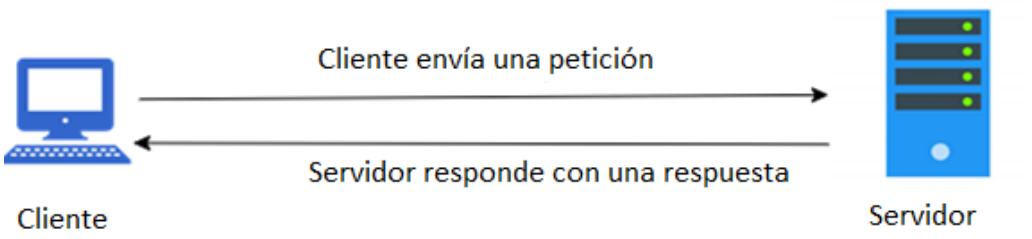


Figura 4. Modelo cliente-servidor

1.4.11 Aplicación Web de una sola Página (Single Page Application)

En el diseño tradicional de una página Web, por cada enlace de navegación con el que el usuario interactuaba, se realizaba una petición para obtener, desde el servidor, documentos HTML y así poder armar lo requerido en el navegador. Las aplicaciones Web de una sola página (SPA) obtienen todo un documento HTML y cada vez que el usuario navega mediante click, ya no realiza una petición al servidor para obtener el documento el HTML asociado, sino que, obtiene datos en formato JSON para armar la página requerida por el usuario en el navegador. Este enfoque de desarrollo Web permite que la aplicación Web sea más rápida y posibilita entonces que el usuario pueda visualizar lo requerido más rápidamente [17] [19]. Una de las principales librerías utilizadas para el desarrollo de estas aplicaciones, es la librería ReactJs usada con el lenguaje de programación JavaScript. En la Figura 5 se puede ver el proceso de interacción entre un cliente y el servidor de una SPA.

Figura 5. Funcionamiento de una SPA.

Ventajas de las SPA

- La carga de la aplicación es más rápida desde el punto de vista del usuario.
- Permite el desarrollo de aplicaciones móviles basadas en SPAs.

- Son fáciles de transformar a aplicaciones Web progresivas, las cuales provee una experiencia offline en el caso de que no se tenga acceso a internet.
  Son excelentes cuando se trabaja en equipo, los desarrolladores de Backend solo se enfocan en el desarrollo de la API y los desarrolladores de Frontend se dedican más a mejorar la experiencia de usuario.

1.4.12 ReactJs

ReactJs es una librería que permite el fácil desarrollo de interfaces de usuario interactivas. ReactJs se basa en el desarrollo de vistas declarativas que permiten al código ser más predecible y fácil de encontrar inconsistencias. Esta librería se basa en componentes encapsulados que administran su propio estado (data), independientemente de otros componentes, el conjunto de estos componentes forma la vista que el usuario visualiza. Estos componentes se escriben mediante el uso de JavaScript y se puede intercambiar data a través de la aplicación sin tener que interactuar con el DOM (Document Object Model, HTML) [20].

1.4.13 Diferencias en los frameworks y librerías para el desarrollo de SPAs

Existen diferentes frameworks para el desarrollo de SPAs, actualmente la comunidad Open Source recomienda los tres más populares, estos son Angular, VueJs, ReactJs *[4]*. En la Tabla 1 se puede observar las principales diferencias y similitudes entre estos frameworks. Sin embargo, debido a ventajas, como facilidad de aprendizaje, en el presente proyecto se eligió ReactJs para el desarrollo del cliente Web.

Tabla 1. Diferencia entre los diferentes frameworks.

Feature	**Angular**	**VueJs**	**ReactJs**
Rendering	Procesa al DOM en dos partes, renderiza	Crea una copia del DOM, la procesa y compara con la original.	Crea una copia del DOM, la procesa y compara con la original.
Popularidad [22]	Popular: 21% de los votos.	No muy popular: 0.8% de los votos	Muy popular: 78% de los votos.
Arquitectura	Basada en módulos que encapsula componentes	Basada en componentes	Basada en componentes.
Respaldado por:	Google	Comunidad OpenSource	Facebook
Modularidad	Si	No	No

Facilidad de aprendizaje	No	Si	Si

1.4.14 Application Programing Interface (API)

Cuando una aplicación se conecta a internet necesita de alguna manera de poder comunicarse con el servidor el cual provee data y servicios. La API funciona como un intermediario entre la aplicación y el servidor que provee comunicación entre estos dos. Gracias a las APIs podemos obtener data que se encuentra remotamente, servicios que se procesan en la nube y funcionalidades que no se realizan directamente en la aplicación [21] .

1.4.15 GraphQL

GraphQL es un estándar de desarrollo de APIs que provee una alternativa más eficiente y flexible al estándar REST [22]. Ah sido desarrollado por Facebook y ahora es mantenida por una larga comunidad de organizaciones y desarrolladores del todo el mundo. En su núcleo, GraphQL permite la obtención de datos de manera declarativa, en donde el cliente puede especificar qué “data” necesita de la API. En vez de definir varios Endpoints (URL de un recurso) que retornan estructuras de datos fijas, un servidor de GraphQL expone solamente un Endpoint y responde lo que exactamente el cliente solicitó. GraphQL es frecuentemente confundido por una tecnología basada en bases de datos, esto es una confusión, GraphQL es un lenguaje de consultas para APIs, no basado en base de datos. En este sentido, la base de datos es transparente para el servidor de GraphQL [22].

1.4.16 Schema de GraphQL

Un esquema de GrapQL es definición de la estructura de la API, en donde se especifican los tipos, consultas y mutaciones con todas sus partes, valores de entrada y de retorno. GraphQL utiliza su sistema de tipado para definir el esquema de la API. En pocas palabras, un esquema es el esqueleto de la API. Una definición de esquema es la forma más concisa de especificar un esquema GraphQL. La sintaxis está bien definida y forma parte de la especificación oficial de GraphQL. Las definiciones de esquema a veces se denominan IDL (lenguaje de definición de interfaz) o SDL (lenguaje de definición de esquema) [23].

1.4.17 Types de GraphQL

El principal componente de un esquema de GraphQL son los tipos (Types) y sus campos (Fields). Cada tipo puede ser entendido como un objeto con propiedades que lo representan, como una Tabla y sus columnas en un esquema de base de datos relacionales [23]. La forma de especificar un tipo es mediante la siguiente sintaxis:

```
type Character {
        name: String!
        appearsIn: [Episode!]!
}
```

En donde:

- Character: Es el nombre del tipo.
- Name y appearsIn: son campos del tipo Character. El símbolo de exclamación indica que son campos requeridos y no nulos. En cambio, el campo appearsIn entre corchetes indica que es una lista con objetos de tipo Episode.

1.4.18 Consultas en GraphQL

Es la forma en que un cliente solicita datos de un servidor de GraphQL. Una de las características más importantes es la que, en GraphQL es que el cliente puede especificar qué campos puede retornar el servidor [23]. La sintaxis básica de una consulta es la siguiente:

```
{
  allPersons {
            name
          }
}
```

Y la respuesta que puede obtener del servidor corresponde a lo que se pidió en la consulta:

```
{
```

```
        "allPersons": [
                { "name": "Johnny" },
                { "name": "Sarah" },
                { "name": "Alice" }
              ]
}
```

1.4.19 Mutaciones en GraphQL

La mayoría de las aplicaciones también necesitan una manera de realizar cambios en los datos que se almacenan en el Backend. Con GraphQL [23], estos cambios son realizados mediante mutaciones y estas se dividen en tres tipos:

- Creación
- Actualización
- Eliminación

Las mutaciones tienen la misma estructura sintáctica que las consultas, pero siempre empiezan con la palabra clave “mutation” para indicar que la operación solicitada es una mutación.

```
mutation {
  createPerson(name: "Bob", age: 36) {
                name
                age  }}
```

Del Ejemplo anterior, el servidor responde con los datos solicitados del registro creado, name y age:

```
"createPerson": {
   "name": "Bob",
   "age": 36,}
```

1.4.20 Subscripciones

La obtención de datos en tiempo real de un servidor es otro requerimiento de muchas aplicaciones de hoy en día. Con GraphQL esto se soluciona mediante la definición de

subscripciones (subscriptions). Cuando un cliente se subscribe a un evento, iniciará una conexión en estado de escucha de eventos que el servidor puede responder en cualquier momento. A diferencia de las mutaciones y consultas que retornan un conjunto de datos fijo, las mutaciones devuelven un flujo de datos en stream [23]. La sintaxis para definir una subscripción:

```
subscription {
        newPerson {
                name
                age
              }
        }
```

1.4.21 Ventajas de GraphQL

- En GraphQL se tiene un solo Endpoint que representa varios recursos. Esto se puede observar la Figura 6.
- Con REST, el cliente puede obtener data que no solicitó en la petición. En cambio, con GraphQL el cliente obtiene exactamente lo que solicitó.
- En otros esquemas como REST, el cliente necesita realizar varias peticiones para obtener la data solicitada. En cambio, en GraphQL con pocas peticiones se obtiene lo requerido.
- GraphQL es ideal para aplicaciones que necesitan realizar pocas peticiones al servidor.
- GraphQL es autodocumentado. En cambio, REST necesita de recursos externos para su documentación.

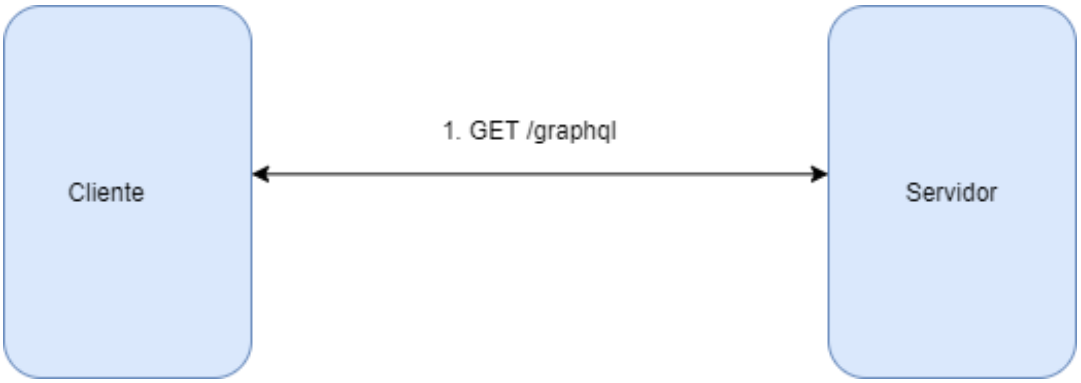


Figura 6. En una API de GraphQL un solo Endpoint es utilizado para la obtención de datos desde el servidor.

1.4.22 Django

Django es un marco de trabajo Web en Python de alto nivel que incentiva el desarrollo rápido, diseño limpio y pragmático. Este marco de trabajo ha sido creado por desarrolladores experimentados enfocados en el diseño de aplicaciones que ajustan a las necesidades del usuario si tener que reinventar la rueda [24].

1.4.23 Ventajas de Django

- Django ha sido desarrollado para el rápido desarrollo de aplicaciones.
- Django ayuda a los desarrolladores a resolver problemas comunes relacionados con la seguridad en aplicaciones.
- Django es flexiblemente escalable lo que permite integrar con librerías de terceros y plugins.
- Provee un simple sistema de administración de la base de datos.
- Soporte para el ORM.

1.4.24 Graphene Python

Es una librería para el desarrollo de APIs en Python de manera fácil, su principal objetivo es proveer APIs de manera simple y extensible para los desarrolladores [25].

1.4.25 SpaCy

Es una librería de Python que ofrece funcionalidad para el Procesamiento de Lenguaje Natural a nivel Industrial. Si un proyecto requiere procesar documentos de la Web con contenido extenso, esta librería es la indicada para el trabajo. Otra ventaja de SpaCy es la capacidad de poder ayudar a preparar texto para Aprendizaje Profundo y puede ser integrado con otras librerías del ecosistema de Inteligencia Artificial de Python como TensorFlow, PyTorch, Gensim, etc [7].

Una vez definidos los recursos y herramientas que nos ayudarán al desarrollo, necesitamos de una metodología que nos guíe en el desarrollo a través de etapas de nuestro sistema. Por lo tanto, en la siguiente sección se discute acerca de la metodología de desarrollo utilizada.

2. METODOLOGÍA

Para el desarrollo de este proyecto se ha tomado en cuenta el enfoque propuesto por Falk [5] el cual se presenta en la Figura 7. Este proceso se caracteriza por ser basado en datos (data-driven). Es decir, los algoritmos y modelos de recomendación se construyen de acuerdo con los datos disponibles del problema que se quiere atacar.

Las fases del enfoque propuesto por Falk, se listan a continuación:

1. Idea: basada en el problema que se quiere resolver.
2. Datos: Los datos que componen el problema.
3. Algoritmos: Algoritmos que pueden ser utilizados de acuerdo con los datos para resolver el problema.
4. Modelo: Diseñar el modelo de recomendación.
5. Offline Testing: pruebas de código y del sistema en un ambiente de desarrollo.
6. Online Testing: pruebas con usuarios reales.

Esos pasos son representados por Pressman [26] mediante un enfoque basado en prototipos y desarrollo iterativo, el cual se aplicó en el presente proyecto.

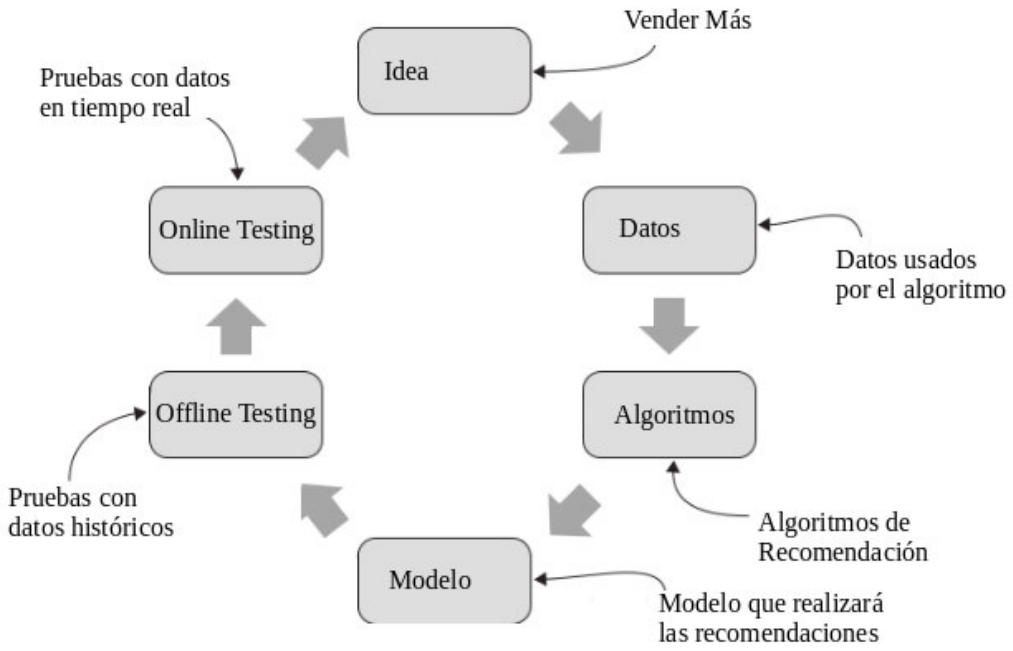


Figura 7. Enfoque basado en Datos para el desarrollo de Sistemas de Recomendación *[5]*.

Este proceso de desarrollo entra en la categoría de Procesos de Desarrollo de Software evolutivo o iterativo. De acuerdo con Pressman [26], en estos procesos de desarrollo se tiene un conjunto de requerimientos básicos de los cuales se genera una primera versión del sistema y este evoluciona en el tiempo de acuerdo con las necesidades del negocio o problemas con la primera versión. Por ejemplo, de la Figura anterior una vez completado un ciclo, se tiene un Sistema de Recomendación básico y este evolucionará de acuerdo con los resultados de las pruebas online y/o offline. Una de las modificaciones puede ser definir una idea diferente o usar un algoritmo distinto para la recomendación.

2.1 Prototipos

Cada iteración del ciclo propuesto por Pressman produce un prototipo de software. Se conoce como prototipo a una primera versión del software desarrollado en base a un conjunto de requerimientos generales y específicos [26]. Este prototipo pasa por una serie de pruebas y se presenta al usuario final para que en base a un feedback y al resultado de las pruebas, el sistema evolucione en la siguiente iteración. La idea general de generar prototipos tiene como fin la identificación de los requerimientos del software y el primer prototipo puede ser desechado. Sin embargo, existen casos en los que la primera versión evoluciona hasta convertirse en un producto final [26].

De manera general el paradigma de desarrollo de prototipos comienza con la comunicación, en donde los desarrolladores se reúnen con los usuarios o dueños de sistema para definir los requerimientos y objetivos generales, en esta fase también se planifica la iteración para realizar un prototipo y se diseña un modelo inicial del primer prototipo. Este modelo se centra en los aspectos generales que los usuarios finales desean que el sistema realice. De acuerdo con este modelado rápido, se desarrolla la primera versión del software. Este se entrega y es evaluado por los participantes del proyecto y al mismo tiempo permite a los desarrolladores entender lo que necesita ser mejorado [26].

El Paradigma de realizar prototipos se muestra en la Figura 8. Las fases del enfoque propuesto por Falk [5] se ajustan al ciclo iterativo propuesto por Pressman [26]. EL proceso comienza desde la etapa de ‘Comunicación’ en donde se establece la idea lo que se desea desarrollar hasta la etapa de ‘Entrega y despliegue’ en donde se entrega el prototipo y se realizan pruebas online con usuarios potenciales del sistema.

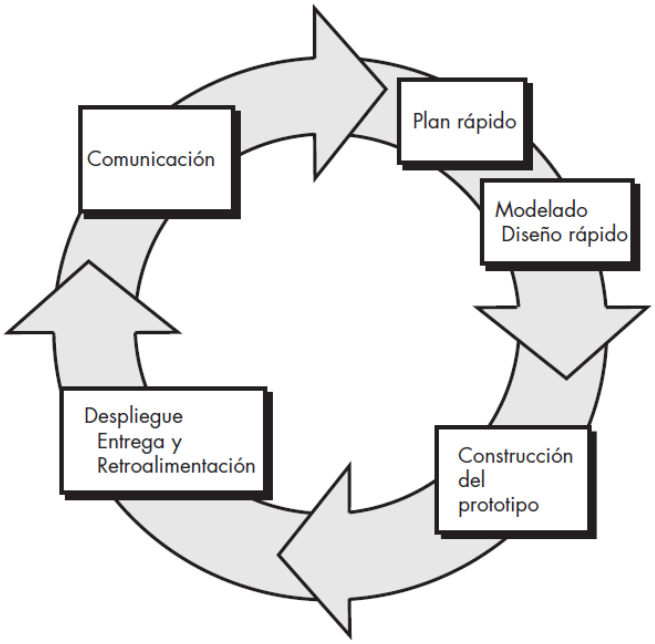


Figura 8. Paradigma de desarrollo de prototipos propuesto por Pressman *[26]*.

En la primera iteración se desarrolla y entrega un prototipo funcional, a este se le realizan pruebas por parte de usuarios potenciales. Después de pasar las pruebas, el sistema evoluciona a partir de las necesidades descritas en la etapa de 'Comunicación', en donde el cliente realiza una retroalimentación del sistema de las mejoras que se deben implementar. A continuación, se lista las fases del proceso propuesto por Pressman [26].

1. Comunicación.
2. Plan Rápido.
3. Modelado y Diseñado rápido.
4. Construcción del prototipo.
5. Despliegue, entrega y retroalimentación.

Sin embargo, para poder desarrollar un software se debe tener en cuenta actividades esenciales en cada una de las fases del proceso descrito anteriormente. Sommerville [27] indica que todo proceso de desarrollo de Software debe constar de las siguientes actividades:

1. Especificaciones de Software.
2. Diseño e Implementación de Software.
3. Validación de Software.

4. Evolución del Software.

2.2 Especificaciones de Software.

El proceso de ingeniería de requerimientos tiene como objetivo la definición correcta de las especificaciones del sistema; es decir, descifrar en un lenguaje técnico lo que el usuario desea que el sistema represente. Este proceso empieza con las especificaciones de stakeholders hasta la definición correcta los requerimientos de usuarios finales y desarrolladores del sistema [27].

El proceso de ingeniería de requerimientos consta de tres actividades principales:

1. Análisis y obtención de requerimientos: Esta actividad se realiza mediante análisis de sistemas existentes, reuniones y discusiones con los dueños del sistema y usuarios finales. El desarrollo de un prototipo ayudará a comprender mejor los requerimientos del sistema.
2. Especificación de Requerimientos: Es la actividad de transformar la información obtenida en requerimientos más detallados en un documento.
3. Validación de Requerimientos: Se evalúa los requerimientos para determinar si son realistas, consistentes y completos.

2.3 Diseño de Software e Implementación.

El Diseño de Software es la descripción de la estructura del sistema a ser implementado, los modelos de datos y estructuras usadas, las interfaces entre los componentes y los algoritmos utilizados. El diseño del sistema no es algo que se realiza de una sola vez, sino que puede evolucionar en diferentes etapas [27].

Las actividades de diseño pueden variar, dependiendo del tipo de sistema a implementar. Por ejemplo, sistemas en tiempo real requieren una fase adicional para el diseño de procesos concurrentes, aunque no incluya la base de datos, por lo que no involucra el diseño de la base de datos. La Figura 9 muestra las 4 principales actividades [27] que pueden ser parte en el proceso de diseño de sistemas de información:

1. **Diseño de la arquitectura:** Identificar la estructura del sistema, principales componentes, módulos, sus relaciones y como están distribuidos.

2. **Diseño de la Base de datos:** Diseño de la estructura de la base de datos o rediseño de un esquema de base de datos existente.
3. **Diseño de Interfaces:** Con una correcta especificación de interfaces, los componentes son transparentes entre sí, cada componente desconoce cómo está implementado el componente con el cual interactúa.
4. **Diseño y selección de Componentes:** Seleccionar componentes para su reutilización en el sistema, si no existen componentes que se pueden reutilizar lo ideal sería diseñar nuevos componentes.

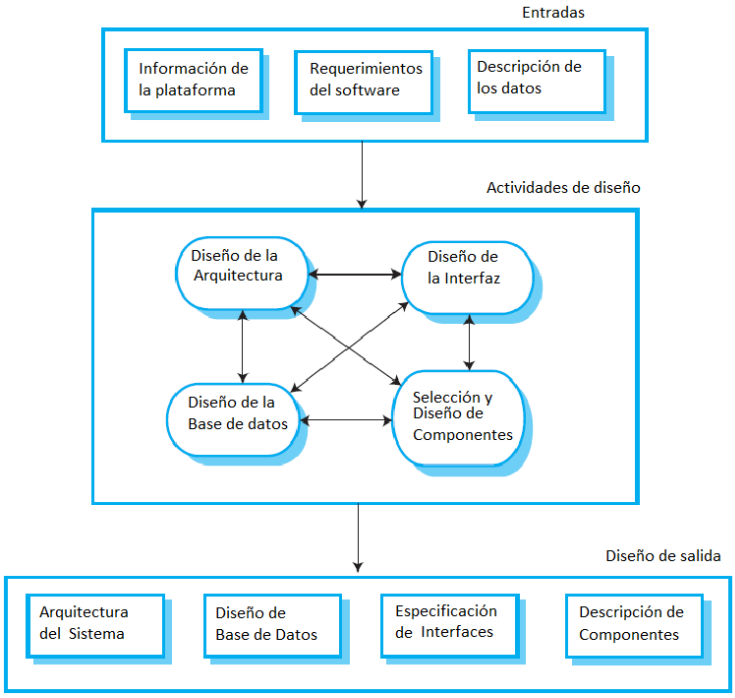


Figura 9. Actividades del proceso de diseño *[27]*.

Todas estas actividades constan de entradas que incluyen información acerca de la plataforma a desarrollarse, los requerimientos y la descripción de los datos. Estos a su vez se convierten en salidas como la arquitectura, el diseño de la base de datos, especificación de interfaz y la descripción de los componentes [27].

2.4 Validación del Software

Esta actividad es más conocida como Validación y Verificación de Sotftware (V & V). Tiene como objetivo revelar que el sistema cumple con las especificaciones y las expectativas de los

clientes. Pruebas del sistema, en donde se prueba la funcionalidad del sistema con datos simulados, es la principal técnica de validación. También puede incluir chequeo de procesos en cada etapa de desarrollo. Sin embargo, se dedica más tiempo a pruebas de sistema en vez del chequeo de procesos [27]. De acuerdo con Sommerville [27] las etapas de pruebas son:

1. **Pruebas de Componentes**: Los componentes que forman parte del sistema, son sometidos a pruebas por parte del equipo de desarrollo. Cada componente pasa por una suite de pruebas individual.
2. **Pruebas del Sistema**: Tienen como fin encontrar errores que pueden resultar de interacciones no anticipadas entre componentes. También se enfoca en revelar que el sistema cumple con las especificaciones funcionales y no funcionales.
3. **Pruebas de Usuarios:** Esta es la etapa final del proceso de pruebas antes de que el sistema sea aceptado para su uso operacional. Pruebas realizadas por los usuarios se realizan al sistema, esto es importante porque errores en el sistema pueden ser revelados.

Si se utiliza un enfoque incremental para el desarrollo, cada incremento debe probarse a medida que se desarrolla, con estas pruebas basadas en los requisitos para ese incremento. En el desarrollo basado en pruebas, que es una parte normal de los procesos ágiles, las pruebas se desarrollan junto con los requisitos antes de que comience el desarrollo. Esto ayuda a los evaluadores y desarrolladores a comprender los requisitos y asegura que no haya demoras a medida que se crean casos de prueba [27].

2.5 Evolución del Software

Una de las principales características del Software es su flexibilidad, debido a que más y más software es incorporado dentro de sistemas grandes y complejos. Esto no ocurre con el hardware que una vez ya fabricado no se le puede incrementar más partes. Sin embargo, en el software puede incrementarse funcionalidad aún después de haber sido entregado al usuario. Debido a esto, es realista pensar que la ingeniería de software como un proceso evolutivo, ver Figura 10, en donde un sistema puede cambiar en el tiempo en respuesta a los requerimientos cambiantes y las necesidades de los usuarios [27].

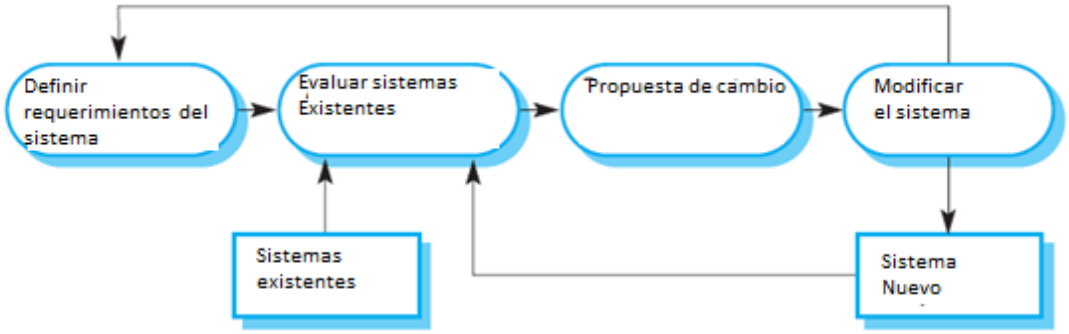


Figura 10. Evolución del Software *[27]*.

Sin embargo, como ayuda al proceso evolutivo considerado para el desarrollo del sistema de recomendación, se han tomado en cuenta paradigmas del agilismo para el estudio como el desarrollo de historias de usuarios, el método Kanban e Integración Continua. En la siguiente sección se discute acerca de estos tres paradigmas.

2.6 Historias de Usuario.

Con el tiempo, los requerimientos del sistema pueden cambiar. Para hacer frente a este cambio los procesos agiles de desarrollo de software integran la obtención de requerimientos como parte de la fase de desarrollo. Para que esto resulte fácil, se ha introducido la idea de "historias de usuarios", en donde una historia de usuario es un escenario de uso del sistema que puede ser experimentado por parte de un usuario final [27].

Para lograr esto, los usuarios deben trabajar en conjunto con los desarrolladores para crear "tarjetas de historias" que describen la historia que encapsula las necesidades de los usuarios. Los desarrolladores implementarán funcionalidad en el sistema de acuerdo con la descripción dada por el usuario en la historia. Una vez definidas las historias, se tiende a dividir éstas en tareas y se estima el esfuerzo y tiempo necesario para la implementación de cada una. De acuerdo con Ambler [28], cuando se escriben las historias de usuario se deben tener en cuenta lo siguiente:

1. Son los stakeholders quienes definen las historias.
2. Usar las herramientas más simples, frecuentemente las historias de usuario se escriben en tarjetas de colores.
3. Tomar en cuenta requerimientos no funcionales.

4. Indicar el tiempo y esfuerzo estimado para la implementación de cada historia.
5. Indicar la prioridad de cada requisito.
6. De manera opcional indicar un identificador único a cada historia.

2.7 Método Kanban

La idea de Kanban surgió de la mano de la empresa automotriz Toyota a finales de los años 40 del siglo 20. En esa época, cuando un empleado de un departamento solicitaba conocer, en tiempo real, el estado de los almacenes de la organización se enviaba una carta o "kanban" entre los departamentos encargados. Esto se aplicaba también en el caso de que un departamento necesitara materiales de otro departamento. Gracias a esto, se decía que en los procesos se aplicaba el principio "justo a tiempo".

El método Kanban ha sido también aplicado en el desarrollo ágil de software. Gracias a esto, los equipos de desarrolladores tienen más flexibilidad en las opciones de planificación, obteniéndose más resultados y transparencia en el ciclo de desarrollo de software. En este método, la ingeniería de software utiliza cartas y un tablero con fines de organización, siendo que estos pueden ser incluso virtuales [29].

2.8 Tableros Kanban

El trabajo del equipo de desarrollo se enfoca alrededor de un tablero Kanban. Esta es una herramienta para visualizar y optimizar el flujo de trabajo dentro del equipo. A pesar de que los tableros físicos son populares entre algunos equipos, los tableros virtuales son una característica esencial en cualquier desarrollo agile de software, en donde se puede colaborar, llevar registros y acceder desde múltiples lugares. La metodología Kanban se basa en la transparencia total del trabajo y la comunicación de capacidad en tiempo real, por lo tanto, el tablero Kanban debe verse como la única fuente de verdad para el trabajo del equipo [29]. En la Figura 11 se puede observar un ejemplo de un tablero Kanban con las tarjetas que lo conforman.

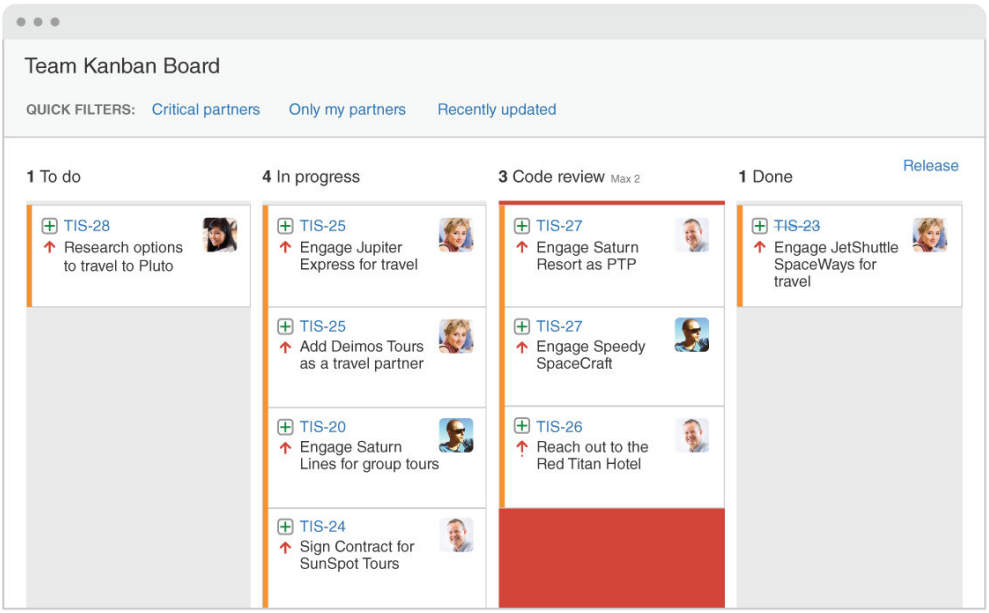


Figura 11. Tablero Kanban y las tarjetas que lo conforman *[29]*.

2.9 Tarjetas Kanban

El principal propósito de Kanban es mostrar el trabajo descrito en tarjetas en el tablero. Gracias a esto, el equipo puede monitorear el progreso de trabajo a través de un flujo. Las tarjetas contienen información esencial acerca la tarea que se realiza; esto da al equipo una visión global acerca de quién está encargado de qué tarea, una descripción del trabajo a realizar, el tiempo estimado, el estado del trabajo realizado, entre otras cosas acerca de la tarea o del trabajo descrito en la tarjeta [29].

2.10 Integración Continua

Es la práctica de integración automática de cambios en el código de múltiples contribuidores en un solo proyecto de software. El proceso de integración continua está compuesto de herramientas automatizadas que aseguran que el nuevo código sea correcto antes de cada integración. Un sistema de control de versiones es el núcleo de este proceso. Este sistema de control de versiones realiza pruebas automatizadas para asegurar la calidad del código, evitar errores de sintaxis, entre otras cosas [30].

La Integración Continua permite que los desarrolladores de software trabajen independientemente, y de forma paralela en la funcionalidad de un sistema de información. Cuando cada funcionalidad está lista para que se integre en el sistema principal, cada

desarrollador realiza la integración de su código al producto final. Este proceso se ha considerado como una buena práctica en sistemas modernos y de alto rendimiento [30].

2.11 Implementación de la Metodología

2.11.1 Comunicación

En esta etapa se mantuvieron reuniones en donde se realizaron las siguientes actividades:

- Inducción acerca del proceso de SScan.
- Estudio acerca del problema de sobrecarga de información digital y herramientas ya propuestas previamente.
- Definición de los requerimientos iniciales del sistema.

Una vez entendido que el problema radica en la sobrecarga de información en el proceso de SScan, se propuso un sistema de recomendación para ayudar a encontrar artículos relevantes entre la información recolectada. Además, se crearon los requerimientos iniciales y se los representó mediante historias de usuarios, los cuales se observan en la Tabla 2.

Tabla 2. Historias de Usuario de los requerimientos del sistema.

Código	**Historia**	**Prioridad**
US01	Yo como usuario deseo poder ver artículos en la aplicación Web mediante una sección	Alta
US02	Yo como usuario deseo poder leer un artículo disponible en la aplicación.	Alta
US03	Yo como usuario deseo poder dar "me gusta" a un artículo mediante un botón.	Alta
US04	Yo como usuario deseo poder observar artículos similares al que estoy leyendo mediante una sección de "artículos similares".	Alta
US05	Yo como usuario deseo poder observar los artículos más recientes mediante una banda.	Media
US06	Yo como usuario deseo poder observar los artículos más votados mediante una banda.	Media
US07	Yo como usuario deseo poder observar recomendaciones en base a mi historial de artículos votados positivamente.	Alta
US08	Yo como usuario deseo poder crear una cuenta en la aplicación Web mediante un formulario.	Media

US9	Yo como usuario deseo poder ingresar a la aplicación mediante un formulario de login.	Media

También se discutió acerca de los datos que se tendrán en cuenta en el sistema de recomendación y qué ítems son los que formarán parte de las recomendaciones. Se llegó a un acuerdo de que los artículos de noticias, especialmente las relacionadas con COVID-19 serían las usadas para el primer prototipo desarrollado. Para esto, se tuvo en cuenta un conjunto de artículos compartidos por otros compañeros que se encuentran trabajando en un proyecto relacionado, este dataset fue proveído en formato CSV, que contiene 662 artículos sobre el coronavirus, con las siguientes columnas:

1. Diario: nombre del diario digital de la noticia original, todos pertenecen al diario El Comercio de Ecuador.
2. País: Ecuador.
3. Fecha: fecha de publicación.
4. Título: título del artículo.
5. Texto: texto completo de la noticia.

Por otro lado, para las noticias relacionadas con temáticas de política y deportes se tomó en cuenta un dataset en formato JSON, el cual fue obtenido en el sitio Webhose [31]. Este dataset contiene noticias en español extraídas en el mes de octubre del 2016. Los siguientes campos se tomaron en cuenta:

1. Título: título de la noticia.
2. Texto: cuerpo de la notica.
3. Categoría: política o deportes

Además, se obtuvo un dataset de noticias en inglés extraídas de la plataforma Kaggle [32], este dataset contiene publicaciones de 15 sitios de noticias de los Estados Unidos. En la siguiente sección se toma en cuenta las herramientas a utilizar y la planificación de proyecto.

2.11.2. Plan Rápido

Una vez detallada la idea, definimos el conjunto de herramientas que vamos a utilizar en el presente proyecto. Como ya se indicó anteriormente el primer prototipo funcional será desarrollado mediante APIs en un esquema de GraphQL y un cliente Web que provee las

recomendaciones y permite la interacción con el usuario. La Tabla 3 lista las diferentes tecnologías usadas en el proyecto, la decisión de usarlas se dió a cabo en la etapa de comunicación cuando se entendió de manera clara el problema.

Tabla 3. Herramientas y Tecnologias usadas en el desarrollo del sistema.

Herramienta/Tecnología	**Descripción**
ReactJs	Librería de Javascript para crear vistas Web basada en componentes.
GraphQL	Tecnología de definición y de consultas de APIs
Django	Framework para desarrollo Web, en donde se configura los servicios Web y APIs.
Graphene	Librería de Python para definición de esquemas de GrapQl
Python	Lenguaje de alto nivel y multipropósito
Javascript	Lenguaje de programación de alto nivel, usado especialmente para desarrollo Web.
SpaCy	Librería de Procesamiento del lenguaje Natural

Una vez definidas las herramientas, comenzamos a estimar el tiempo necesario para completar el proyecto integrador. En nuestro caso, el cronograma tiene un periodo de tiempo de 6 meses, el cual concuerda con la extensión de tiempo de un semestre académico de la Escuela Politécnica Nacional. El mismo se muestra en la sección 7.1 de Anexos. En la siguiente sección, antes del modelado del sistema, se discute y estudia los algoritmos que pueden servir para establecer relaciones y similitudes entre documentos, los cuales se tendrán en cuenta para la generación de recomendaciones.

2.11.3 Algoritmos

En esta fase se estudió los diferentes enfoques y algoritmos para generar recomendaciones, se tomó en cuenta tanto los algoritmos basados en contenidos como aquellos basados en filtrado colaborativo. Sin embargo, debido a la naturaleza del problema y recomendaciones de la literatura revisada, se retuvieron los algoritmos usados para generar recomendaciones basándose en características de los ítems y similitudes entre estos, especialmente algoritmos basados en procesamiento del lenguaje natural [5] [8]. A continuación, se lista el conjunto de algoritmos tomados en cuenta:

1. Tokenización, modelo de Bag-of-Words y el modelo TF-IDF (Term Frecuency – Inverse Document Frecuency).
2. LDA (Laten Dirichlet Allocation).
3. El Modelo NER de SpaCy.

Primer Modelo de vectores con TF-IDF

Para el estudio de este modelo se realizó una investigación acerca de cómo generar vectores de texto basándose en las palabras más representativas y la frecuencia de estas en el texto del artículo, para esto se tomó en cuenta el siguiente workflow:

1. Tokenizar el texto.
2. Eliminar las Stop-Words.
3. Radicalización de palabras.
4. Importancia de las palabras en base a TF-IDF.
5. Similitud entre los vectores de texto generados.

Tokenización y Modelo Bag-of-Words (BoW)

El modelo Bag-of-Words (BoW) crea listas que representan documentos en base a los tokens (palabras o caracteres) que componen el texto. Por ejemplo, dado un texto:

"the man likes big ice creams"

La representación en el modelo BoW sería una lista como la siguiente:

["the", "man", "likes", "big", "ice", "creams"]

Muchas de las veces se tiene un texto extenso, el cual contiene palabras que no añaden ningún conocimiento al modelo BoW, pero sí son útiles para el lenguaje natural. Estas palabras son conocidas como Stop-Words y pueden ser eliminadas. Ejemplos de Stop-Words en el idioma es español pueden ser:

- Allí.
- Ahora.
- Bajo, etc.

Eliminación de Stop-Words

Esta técnica trata de eliminar palabras o caracteres en el texto no aportan ningún conocimiento o valor al documento. Por el ejemplo, la palabra “the”, del ejemplo anterior, no contiene ninguna información descriptiva que nos puede ser de utilidad en un modelo de procesamiento del lenguaje natural. Por lo tanto, estas palabras deben ser eliminadas del modelo BoW. En muchos de los paquetes y librerías de programación ya se tienen definidas listas de Stop-Words para su correcta eliminación de un corpus [5].

Radicalización (Stemming) y Lematización de Palabras

Palabras como *correr* y *corriendo* son consideradas como si fueran diferentes en un modelo de BoW y a su vez pueden ser procesadas de manera diferente. Por lo tanto, se tiene que buscar una manera de normalizar las palabras. La mejor manera es usar un lematizador, el cual genera la base de una palabra. La base para el ejemplo anterior es “correr”.

En cambio, un radicalizador de palabras, es un proceso heurístico que recorta el final de cada palabra con el objetivo de encontrar la base. Un ejemplo de ejecución del algoritmo de stemming puede ser observado en la Figura 12.

```
[19]: # generating the vector with words and without stop words
stop_words = set(stopwords.words('spanish'))
es_stemmer = SnowballStemmer('spanish')
texts = [[
    es_stemmer.stem(word) for word in document.lower().split() if word not in stop_words
]
    for document in documents
]

[20]: pprint.pprint(texts[1])

['chin',
 'incertidumbr',
 'propag',
 'extrañ',
 'cep',
 'neumon',
 'mart',
 'diciembr',
 'infect',
 'person',
 'siet',
 'critic',
 'autor',
 'sanitari',
```

Figura 12. Procesamiento de las noticias de El Comercio, eliminación de stop words y radicalización de cada una de las palabras.

Importancia de las palabras en base a su frecuencia en el documento y el corpus (TF-IDF)

Una manera sencilla de establecer si una palabra tiene importancia en el documento, es mediante la frecuencia de aparición de ésta en el documento:

$$tf(palabra, documento) = numero\ de\ veces\ que\ aparece\ la\ palabra\ en\ el\ documento\ actual.$$

Es común que se use una función logarítmica para representar esta función:

$$tf(palabra, documento) = 1 + log(frecuenciadelapalabra)$$

Mientras una palabra sea más frecuentemente en el texto, la posibilidad de que sea importante es alta, si es que se asume que se han eliminado la Stop-Words. Para dar un mayor significado a las palabras en un conjunto de documentos se puede usar la frecuencia inversa del documento (IDF), la cual es el número de todos los documentos dividido para el total de documentos que contienen la palabra. De esta manera se define TF-IDF como:

$$tf - idf(palabra, documento) = tf(palabra, documento) * idf(palabra, documento)$$

En Python, podemos generar un modelo de TF-IDF en base a un diccionario, palabras entre el corpus de texto, y el texto en sí ya procesado previamente. Esto se muestra en la Figura 13, en donde se puede observar el código del modelo.

```
[25]: dictionary = corpora.Dictionary(processed_corpus) # si permite actualizar el dictionary

Dictionary(5121 unique tokens: ['abarc', 'abiert', 'acced', 'actual', 'acuerd']...)

[30]: bow_corpus = [dictionary.doc2bow(text) for text in processed_corpus] # crear el modelo BoW en base al texto procesado
tfidf_model = TfidfModel(bow_corpus)
```

Figura 13. Creación de un modelo TF-IDF.

Similitud de Documentos con TF-IDF

Una vez establecido el diccionario y el modelo, podemos generar vectores de texto en un espacio dimensional definido por el número total de tokens presentes en el corpus de texto. Para el cálculo de la similitud utilizamos el Coseno de la Similitud que se define como:

$$cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Donde

- n: tamaño de los vectores, en nuestro caso n = 50.
- A: el vector que representa al documento 1.
- B: el vector que representa al documento 2
- i: el i-ésimo elemento del vector A o el B.

En base a lo anterior, se puede afirmar que aquellos documentos cuyo coseno del ángulo es cercano a uno son similares, y diferentes en el caso de ser cercano a cero. En base a esto nosotros podemos generar recomendaciones tomando en cuenta lo siguiente:

1. Para un usuario existe ya un conjunto de artículos que han sido relevantes para él en el pasado.
2. Para cada uno de estos documentos se usa el modelo TF-IDF para buscar otros documentos similares en el corpus.

3. Agregar los documentos similares en la base de datos relacionando con el usuario.
4. Recomendar al usuario estos documentos similares.

Modelo de LDA

Este es un modelo generativo, el cual genera tópicos en base a la frecuencia de las palabras en el corpus. Por ejemplo, considere un texto z el cual tiene el siguiente valor:

Z = Spiderman se encuentra en su casa con su laptop y desayunando con un tenedor.

Si se consideran tres tópicos con las siguientes palabras:

1. Superheroe: Spiderman, vuela, fuerte, superhéroe.
2. CienciasComputación: Computadoras, laptop, cpu.
3. Comida: Comer, desayuno, tenedor.

En base a estos tópicos podemos afirmar que el texto Z es generado en base a una combinación probabilística de los 3 anteriores tópicos o más generalmente:

$$Z = 0.2Superheroe + 0.1CienciasComputación + 0.3Comida$$

De este modelo, no se llegó a una implementación, pero se estudió un ejemplo en el cual se toma en cuenta un corpus que contiene texto de noticias en ingles de diferentes categorías como política, deportes, religión, etc.

Modelo NER de SpaCy

A pesar de que se investigó acerca de TF-IDF para la representación vectorial de documentos que permiten encontrar similitud entre cadenas de texto; sin embargo, en el presente proyecto se utilizó los modelos NER (Name Entity Recognition) de la librería SpaCy. Este modelo es generado utilizando Redes Neuronales Convolucionales, las cuales toman un vector de cada palabra presente en el documento y trata de hallar si cada una de éstas pertenece a una entidad. Durante este proceso también se calcula un vector del texto, el cual es conocido como vector de resumen (summary vector) [33]. A diferencia de TF-IDF, el uso del modelo de SpaCy es de mayor facilidad y permite la identificación de tokens en el texto, lematización, similitud entre documentos y puede ser usado como herramienta de preprocesamiento de texto para modelos de Deep Learning más complejos [34].

Gracias a la representación vectorial obtenida por el Modelo de SpaCy, podemos encontrar similitud entre documentos usando el Coseno de la Similitud. El tamaño del vector depende del modelo utilizado por SpaCy. Para artículos en español usamos el modelo es_core_news_md, el cual ha sido entrenado con un corpus de noticias en español. Estos vectores pertenecen a un espacio vectorial de 50 dimensiones, es decir cada vector consta de 50 elementos.

El enfoque del modelo de SpaCy para el procesamiento de texto que se utilizó para generar el modelo es_core_news_md es el siguiente:

1. **Embed:** En la primera fase se mapea cada una de las palabras de texto en un vector representativo. Para esto se utiliza la técnica doc2Array, el cual extrae 4 atributos de cada palabra:
   1.1. Norm: un id de la forma normalizada de la cadena de texto, el cual es básicamente la cadena en si en minúsculas.
   1.2. Prefix: prefijo de la palabra.
   1.3. Suffix: sufijo de la palabra.
   1.4. Word shape: que reemplazada todos los dígitos con la letra "d".
   Habiendo obtenido los 4 atributos de cada palabra, estos se convierten en entrada de una función que retorna un vector para cada atributo, y luego estos son concatenados para crear el vector representativo de la palabra.
2. **Encode:** Una vez obtenidos los vectores para cada palabra, estos se convierten en la entrada de un modelo de redes neurales convolucionales las cuales, de acuerdo con un filtrador, aprenderá acerca de las representaciones de las palabras en base a sus palabras vecinas.
3. **Attend:** En esta fase se concatenan los vectores de las palabras en un vector de resumen (summary vector).
4. **Predict:** Una vez calculado los vectores para cada palabra y procesados por el modelo neural, se predice si las palabras en el texto pertenecen a una entidad o no. Por ejemplo, la palabra "Nestlé" pertenece a una entidad conocida como organización.

Además, SpaCy ofrece una suite de modelos preentrenados que podemos utilizar. En la Tabla 4 se puede observar los modelos para el idioma inglés y español. Observe que todos los modelos han sido entrenados con corpus de textos de noticias y blogs de la Web.

Tabla 4. Modelos NER disponibles en español de SpaCy.

Nombre	Lenguaje	Tamaño	Vectores precalculados	Tipo	F1-Score
en_core_Web_sm	Ingles	11 MB	n/a	Texto escrito (blogs, noticias, comentarios)	85.36
en_core_Web_md	Ingles	91 MB	20k vectores únicos (300 dimensiones)	Texto escrito (blogs, noticias, comentarios)	85.92
en_core_Web_lg	Ingles	789 MB	685k vectores únicos (300 dimensiones)	Texto escrito (blogs, noticias, comentarios)	86.52
es_core_news_sm	Español	15 MB	n/a	Texto escrito (noticias)	89.41
es_core_news_md	Español	45 MB	20k vectores únicos (300 dimensiones)	Texto escrito (noticias)	89.84
es_core_news_lg	Español	546 MB	500k vectores únicos (300 dimensiones)	Texto escrito (noticias)	90.32

En el presente proyecto se utilizó el modelo es_core_news_md el cual calcula vectores de texto con los cuales podemos calcular similitud entre dos documentos A y B mediante el cálculo de coseno del ángulo entre los vectores. Debido a su extenso uso de procesamiento de texto en español y facilidad de uso se optó por usar el modelo NER de SpaCy para la generación de recomendaciones basadas en contenidos de artículos de noticias. Sin embargo, para poder proveer recomendaciones necesitamos generar modelos de recomendación, la estructura del sistema y la base de datos. Para esto en la siguiente sección se discute del modelado del sistema.

2.11.4. Modelado y diseño rápido

El modelado y diseño del sistema permite establecer los componentes y sus interacciones que, en conjunto, definen el sistema en sí. El primer paso en esta fase es determinar las partes que componen la arquitectura y el modelo del sistema. En la siguiente lista se definen los componentes a modelar en el proyecto:

- Modelo de Base de datos relacional.
- Esquema de GraphQL.

Modelado de Base de datos relacional

Un diagrama de entidad-relación de una base de datos es una representación pictórica, en donde se describen las relaciones entre las distintas Tablas que componen una base de datos relacional. En la Figura 14, se puede observar una primera versión del modelo entidad-relación de la base de datos que se utilizará en este proyecto. Se puede así observar que, en un principio, esta base se componía de pocas Tablas y relaciones. Sin embargo, en las siguientes fases se añadieron más Tablas y relaciones que componen el primer prototipo. El modelo final de la base de datos se describe, en la sección 3.1.2 Backend y APIs.

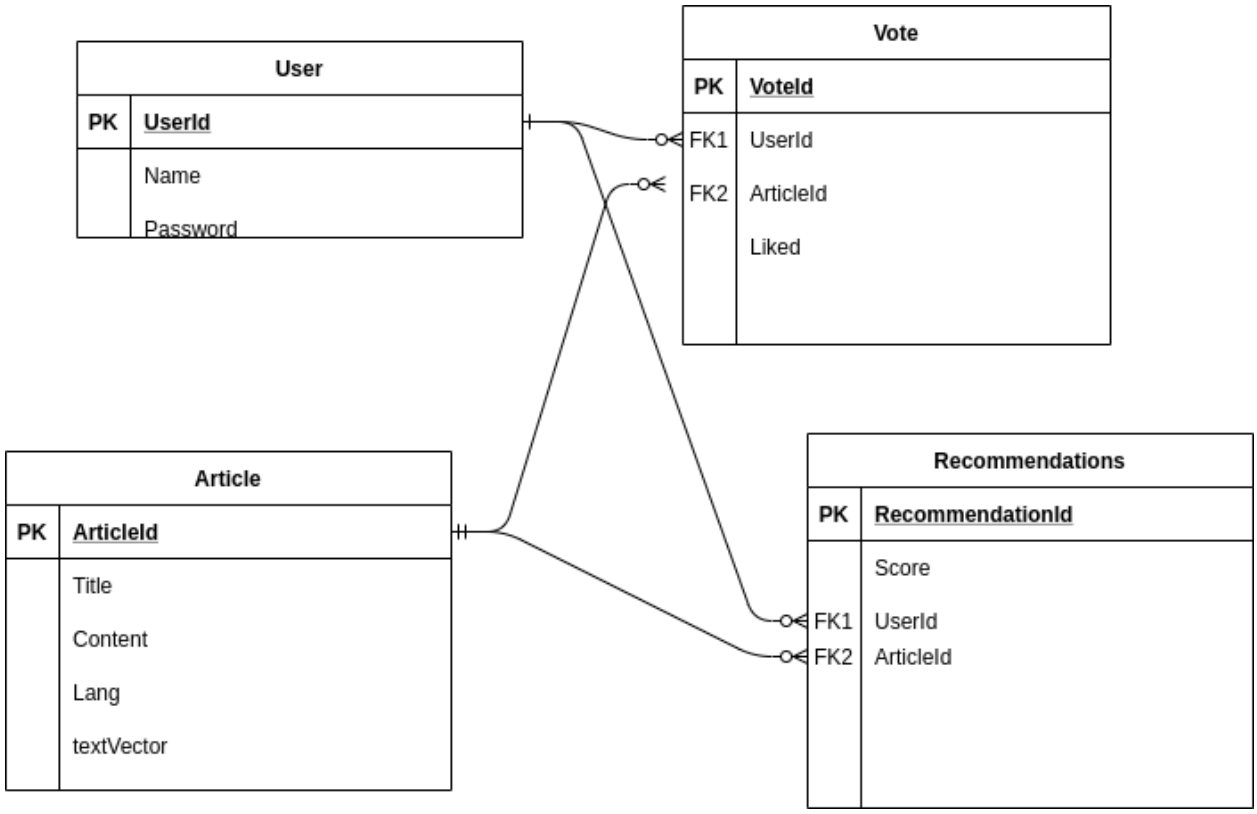


Figura 14. Modelo Inicial de la Base de Datos

En donde:

- User: Es la Tabla que representa al usuario, se conforma de los siguientes atributos:
  - **UserId:** identificador único y clave primaria.
  - **Name:** nombre que identifica al usuario.

- **Password:** contraseña que permite la autenticación del usuario al sistema.

- Vote: Es la Tabla que registra los “me gusta” que realiza un usuario a varios artículos de noticias, se conforma de los siguientes campos:
  - **VoteId**: identificador único de cada registro.
  - **UserId**: clave foránea que representa al usuario que índico que le “gusta” un artículo.
  - **Liked**: indica si al usuario le ha gustado el artículo o no.
- Article: es la Tabla que representa al artículo de noticia.
  - **Title**: título de la noticia.
  - **Content:** contenido o summary del artículo.
  - **Lang:** idioma de la noticia, español por defecto.
  - **textVector:** vector que representa el artículo de noticia, éste es calculado por el modelo de NLP de la librería SpaCy.
- Recommendation: Tabla que representa las recomendaciones que se generan para el usuario.
  - **Score**: similitud con el documento ya degustado por el usuario y que permite establecer si se convierte en recomendación basándose en un valor umbral.
  - **UserId:** id del usuario a quien se le realiza la recomendación.
  - **ArticleId**: id del artículo que será recomendado al usuario.

Esquema de GraphQL

Una vez establecido el modelo de base de datos, definimos el esquema de GraphQL, como se indicó anteriormente. Un esquema de GraphQL define la API con la cual se comunicarán los componentes de la aplicación Web con la funcionalidad presente en el Backend, la cual permite la generación de recomendaciones y operaciones CRUD en la base de datos.

A continuación, se definen los tipos del esquema inicial de GraphQL, de igual manera la versión final de este esquema en esta primera iteración se lo especifica en la sección Descripción del Sistema.

```
type Vote {
        user: User!
                article: Article!
               liked: Boolean! }
Type Article {
         title: String!
         Content: String!
         lang: String!
         textVector: String!
         vote: [VoteType] }

type Recommendation {
         score: Float!
         user: UserType
         article: ArticleType
         recommended: Boolean! }
```

Además de definir el esquema, debemos definir los Queries y Mutaciones relacionadas con las operaciones que puede realizar el sistema en esta primera versión. A continuación, definimos y listamos los Queries y Mutaciones. Sin embargo, una versión ya modificada y más completa se presentará en la sección Descripción del Sistema y, de manera más detallada, en las secciones 7.3 y 7.4 en Anexos:

- **articles**
    - **Descripción:** Permite obtener una lista de artículos basándose en una búsqueda y/o mediante paginación.
    - **Arguments:**
        - **search: String**
        - **first: Int**

        - **skip: Int**
- **article**
    - **Descripción:** Buscar un artículo basado en el Id.
    - **Argumentos:**
        - **articleId: Int!**
- **recentArticles**
    - **Descripción:** Obtiene una lista de los 10 artículos más recientes.
    - **Argumentos:** n/a
- **mostVoted**
    - **Descripción:** Obtiene los 10 artículos con más votos/likes.
    - **Argumentos:** n/a
- **users**
    - **Descripción:** Retorna una lista de los usuarios registrados en la aplicación.
    - **Argumentos:** n/a
- **user**
    - **Descripción:** Retorna un usuario dado su Id
    - **Argumentos:**
        - **Id: Int!**
- **votes**
    - **Descripción:** Retorna una lista de todos los likes en conjunto con el artículo y el usuario que ha creado el like.
    - **Argumentos:** n/a
- **vote**
    - **Descripción:** Retorna si el usuario actual ha degustado o no de un artículo dado su id. Este query funciona si es que el usuario se encuentra logueado. Necesita que la cabecera authorization tenga el valor del token.
    - **Argumentos:**
        - articleId: Int!

- **recommendations**
    - **Descripción:** Obtiene una lista de los 10 artículos que pueden ser de interés para el usuario que se encuentra actualmente logueado. Necesita que la cabecera authorization tenga el valor del token.

- **Argumentos:** n/a

2.11.5. Construcción del Prototipo

Una vez definidos los modelos, comenzamos con la etapa de construcción del Prototipo. Esto se lo realizará gracias al uso de las distintas herramientas y tecnologías descritas previamente, y constará de dos partes: el Backend y la aplicación Web cliente en donde el usuario podrá acceder e interactuar con los distintos artículos presentes en la base de datos.

Backend

El desarrollo del Backend se realizó aprovechando la capacidad del framework de desarrollo Web Django, de poder dividir las distintas lógicas de entidades en su propia app, como si fuera un componente aislado de otros. Aun así, cada una de estas apps puede reutilizar lógica definida en otra. A continuación, listamos cada una de estas apps.

- Articles: app que contiene la lógica relacionada con los artículos de noticias. Define las operaciones CRUD que puede usar el cliente.
- Collector: app que contiene la lógica para llevar registros de las interacciones entre usuarios y artículos, especialmente recolectar “me gusta” realizados a noticias en el cliente. Esta app en si es la definición de perfil de usuario que se indicó anteriormente.
- Users: Es la app que contiene y define la lógica de Usuarios. Por ejemplo, en esta app se define como realizar la creación de usuarios mediante una Mutación de GraphQL.

En la Figura 15, se muestra una captura de la estructura del proyecto en donde se puede observar los componentes del Backend como son las apps, que se representan como directorios que contienen su propia funcionalidad definida. El directorio *newsgrapqhl* contiene las configuraciones globales del proyecto en Django, en cambio *staticfiles* es un directorio para archivos estáticos.

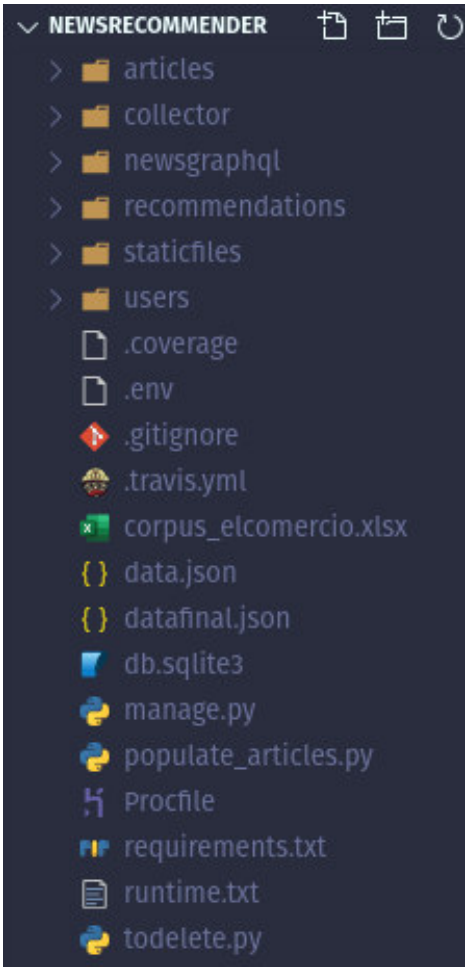


Figura 15. Estructura del Backend

Cada una de las apps de Django contiene scripts de Python, en donde se programa la funcionalidad y lógica de las APIs. Sin embargo, aquellos scripts que son de mayor interés para nosotros son los siguientes:

- **models.py:** Define entidades, Tablas de la base de datos, como si fueran clases de Python.
- **Schema.py:** definimos el esquema de GraphQL mediante clases de Python.
- **Tests.py:** define una suite de pruebas unitarias que permiten comparar los resultados obtenidos de las funciones con un valor esperado.

Cliente

El usuario deberá tener una forma de interactuar con los artículos, decir cuales le gustaron y recibir recomendaciones. Para esto definimos un cliente de aplicación Web de una sola página desarrollada mediante la librería ReactJs, la cual, permite desarrollo de aplicaciones Web basandose en componentes Web que contienen funcionalidad y operan independientemente de otros. En la Figura 16, se muestra la estructura del cliente. Sin embargo, nos enfocaremos más en los siguientes directorios y archivos:

- Components: es la sección en donde se definen los componentes que conforman una vista Web.
- Test: sección en donde se definen suites de pruebas unitarias.
- Views: sección donde se define cada una de las vistas (páginas), que el usuario visualiza al entrar a la Web app.

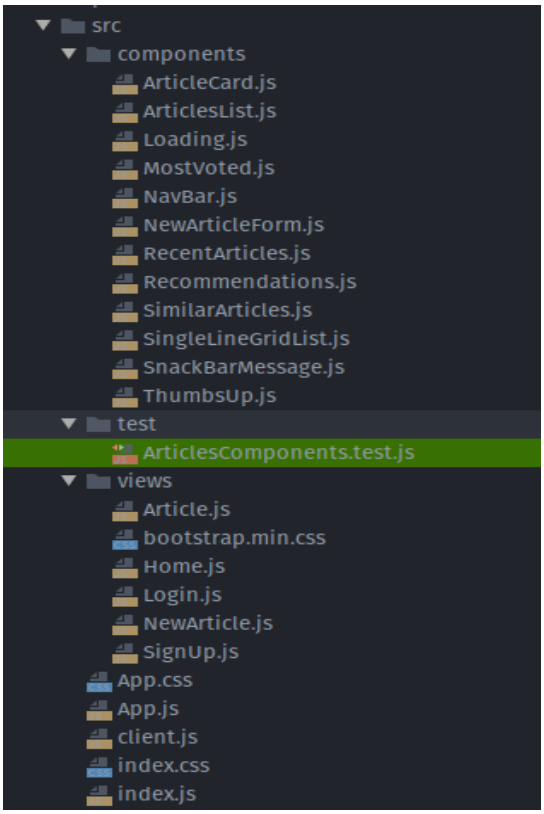


Figura 16. Estructura del código fuente del cliente.

Sin embargo, dos de los enfoques de desarrollo de software ágil usado en el presente proyecto son Integración continua y control de versiones mediante Git y GitHub. Ambos se especifican en las siguientes secciones.

Control de Versiones con Git

La manera de que funciona Git es mediante la captura de snapshoots de los archivos y directorios que conforman el proyecto. Cada snapshoot es una versión del Proyecto [35].

Integración continua

Uno de los enfoques que se especificó fue la utilización de Integración Continua (CI), la misma que consiste en incorporar nuevo código o funcionalidad a la aplicación y validarlo mediante un servidor de CI. En nuestro caso, se utilizó el servidor Travis-CI que permite configurar un ambiente para integración continua de código, de manera que, cuando se añada nueva funcionalidad, el servidor ejecutará pruebas unitarias para constatar que no exista conflictos e inconsistencias entre los componentes de la aplicación.

2.11.6 Entrega y Despliegue

Esta fase consiste en que, tanto el Backend, los servicios Web y en donde se encuentra implementado el recomendador; como el Frontend, el cliente Web, sean desplegados en un servicio en la nube independiente de cada uno. Esto se lo realizó mediante el PaaS (Plataforma como servicio) Heroku, el cual permite el despliegue de aplicaciones en la nube [36]. Esta plataforma permite configurar procesos que mantienen a la aplicación en un estado de escucha de manera que puede ser accedidas mediante una URL. A continuación, se lista las URLs asociadas al Backend y al Frontend:

- Backend: https://tranquil-mountain-27611.herokuapp.com/GraphQL/
- Frontend: https://react-newsrecommender.netlify.app/

El cliente (Frontend) utiliza la ULR del Backend para realizar consultas y operaciones CRUD mediante GraphQL. Sin embargo, al ingresar la URL podemos observar una interfaz gráfica con la cual podemos interactuar con los servicios de GraphQL y obtener información básica de todos los recursos implementados en el servidor

2.11.7 Pruebas y Validación.

Las pruebas de un sistema de información permiten validar que su funcionalidad cumple con las expectativas que el cliente y desarrolladores de sistemas esperan que el software tenga [26]. En esta fase, se define los diferentes tipos de pruebas que se realizaron al sistema de recomendación tanto al Backend como Frontend, y en dos ambientes que se listan a continuación:

1. Pruebas Offline: se incluyen pruebas unitarias, de cobertura al código en tanto como Backend como Frontend.
2. Pruebas Online: Pruebas con usuarios en un ambiente simulado y mediante el uso del TAM (Technology Acceptance Model)

2.11.8 Pruebas Offline

En desarrollo de software, se define como pruebas unitarias a aquellas en donde se realizan test a componentes o unidades individuales de código y comprueban que cada unidad se ejecute de la manera esperada. Las pruebas unitarias se realizan durante el desarrollo de una aplicación y son los desarrolladores quienes definen y ejecutan estas pruebas. Se puede aplicar a una unidad a una función, método, procedimiento, módulo o un objeto [37].

Generalmente este tipo de pruebas se realizan de manera automatizada, los pasos que se realizan son los siguientes:

- Un desarrollador escribe código que ejecutará la prueba de una función o unidad de código.
- El desarrollador puede aislar las unidades de código a ser testeadas.
- Un programador generalmente utiliza un framework para el desarrollo de pruebas automatizadas.

Pruebas unitarias en el Backend

Unittest es el principal framework de pruebas unitarias en Python, viene integrado en el framework Django utilizado en el proyecto. Para su integración, Django crea un script denominado test.py en donde debemos escribir código para automatizar las pruebas. Esto se

lo realiza mediante una clase que hereda atributos y métodos de la clase TestCase y consta de dos métodos:

1. **SetUp**: Es utilizado para establecer parámetros globales de la prueba.
2. **Métodos test_:** En cada uno de estos métodos se ejecutan las unidades de código a ser testeadas y se comparan el resultado esperado con el resultado resultante de la ejecución.

De lo descrito anteriormente, la Figura 17 muestra un ejemplo de creación de un caso de prueba para verificar que el Query de GraphQL usado para obtener artículos de la base de datos funciona de la manera esperada. En setUp se ingresan 2 artículos en la base de datos de prueba y el método test_search_articles ejecuta el query de GraphQL y compara lo esperado con lo que devolvió la operación.

```
class ArticleTestCase(TestCase):
    def setUp(self):
        Article.objects.create(
            title="Prueba de busqueda de 2 elementos",
            summary="Esta es una prueba de busqueda de dos elementos",
            lang="es",

        )
        Article.objects.create(
            title="Prueba de busqueda de 2 elementos v2",
            summary="Esta es una prueba de busqueda de dos elementos v2",
            lang="es",

        )
        #print("is setting up? ")

    def test_search_of_articles(self):
        #self.maxDiff = None
        executed = client.execute('''
        query {…

        expected = {'data': {'articles': […
        }
        }

        self.assertTrue(len(executed.get('data').get('articles')) == 2)
        self.assertDictEqual(executed, expected)
        # self.assertEqual
```

Figura 17. Caso de prueba de obtener un conjunto de artículos usando un Query de GraphQL.

Pruebas unitarias en el Frontend

Para el Frontend se utilizó Jest como el principal mecanismo de ejecución de pruebas. La herramienta utilizada para crear proyectos en ReactJs le integra automáticamente al proyecto, por lo que no hay necesidad de instalar ningún otro componente. Sin embargo, en este caso Jest no compara un resultado calculado con uno esperado, sino que, verifica que los componentes Web se montan de manera correcta para armar la aplicación Web.

En nuestro caso creamos suites de pruebas usando el formato <nombresuite>.test.js, esto para que Jest pueda ejecutar las pruebas en suites aislados. De manera general Jest comprueba que los componentes se renderizan correctamente en el DOM [38].

La implementación de las pruebas funcionales se realizó de la siguiente manera:

1. Se importa el client de GraphQL usado para realizar consultas.
2. Se crea un renderizador de pruebas.
3. Se encapsula el componente a ser testeado con un componente ApolloProvider.
4. Jest ejecuta las pruebas y comprueban que se renderizan correctamente.

La Figura 18, muestra un ejemplo de lo descrito anteriormente. En este caso, se prueba que el componente de Article, el cual representa la vista del artículo que el usuario visualiza, se renderice de manera correcta en el DOM.

```
it('should render Article view', function () {
    renderer.create(
        <ApolloProvider client={client}>
            <Article articleId={61919}/>
        </ApolloProvider>
    )


});
```

Figura 18. Prueba unitaria del componente Article.

Pruebas de Cobertura

Este tipo de pruebas definen una métrica que indica el porcentaje de código ejecutado por un conjunto de pruebas. Incluye un reporte de la cantidad de código cubierto o ejecutado por un conjunto de pruebas y las ramas de sentencias condicionales cubiertas por la prueba [39]. Para realizar este tipo de pruebas se debe tener en cuenta lo siguiente:

1. El número (X) total de líneas de código de la unidad de código que se está sometiendo a pruebas.
2. La cantidad (Y) de líneas que actualmente se ejecutan en la prueba.
3. Dividir Y para X y multiplicar para 100 nos da el valor de la cobertura del código.

En nuestro caso, los frameworks utilizados para las pruebas unitarias tienen integrado reportes que indican la cantidad de código cubierto por las pruebas unitarias en cada unidad/módulo que al que se le han ejecutado pruebas. En el caso del Backend se pudo integrar una herramienta online conocida como Coverage para visualizar de manera más intuitiva la cantidad de código cubierto por las pruebas. En la sección de resultados se mostrará los reportes generados por las pruebas unitarias y de cobertura.

2.11.9 Pruebas Online

Technology Acceptance Model (TAM)

El Modelo de Aceptación de Tecnologías viene determinado por factores psicológicos, especialmente creencias, que tienen consecuencias sobre la utilización de una tecnología. TAM se apoya en la teoría psicológica de la “Acción Razonada” la cual tiene como objetivo predecir la conducta de las personas en función de sus intenciones y actitudes [40]. Este modelo sugiere que la actitud hacia el uso de una tecnología informática se basa en dos variables previas (Figura 19):

- **la utilidad percibida:** probabilidad de que una persona utilice la tecnología y mejore su trabajo mediante esta tecnología.
- **facilidad de uso:** el grado de que una persona cree que usar la tecnología requerirá o no de esfuerzo.

Figura 19. Technology Acceptance Model.

3. RESULTADOS

Una vez descrito el problema que queremos solucionar, la teoría referente a Sistemas de Recomendación, el proceso de desarrollo y las herramientas utilizadas; en este capítulo realizaremos una descripción detallada de nuestra solución, la cual incluye los modelos completos del sistema final, y explicaremos mediante un ejemplo la interacción que puede ser realizada por usuarios del sistema que es parte del presente proyecto integrador. Además, presentaremos los resultados de las pruebas realizadas en las dos fases de testing: online y offline.

3.1 Descripción del sistema

Nuestro sistema de recomendación tiene como objetivo recolectar artículos que un usuario considera de interés para su proceso de SScan y recomendar artículos que comparten similitud entre aquellos ya leidos con anterioridad por parte del usuario. Nuestra implementación se base en 3 partes:

1. Un Cliente (Frontend): Desarrollado en ReactJs, en el cual el usuario puede interactuar con los artículos y en donde se desplegarán las recomendaciones.
2. Backend: Web APIs diseñados con el esquema de GraphQL y Django opara proveer los artículos y las recomendaciones al cliente.
3. Base de datos: Se utiliza Sqlite3 como motor de base de datos y el ORM de Django.

La arquitectura final del sistema ha sido desarrollada en base a los lineamientos propuestos por Falk [5]. En ésta, se toma en cuenta las partes descritas anteriormente, y su interacción entre sí. En la Figura 20, se pueden observar los distintos componentes que forman parte de nuestro sistema.

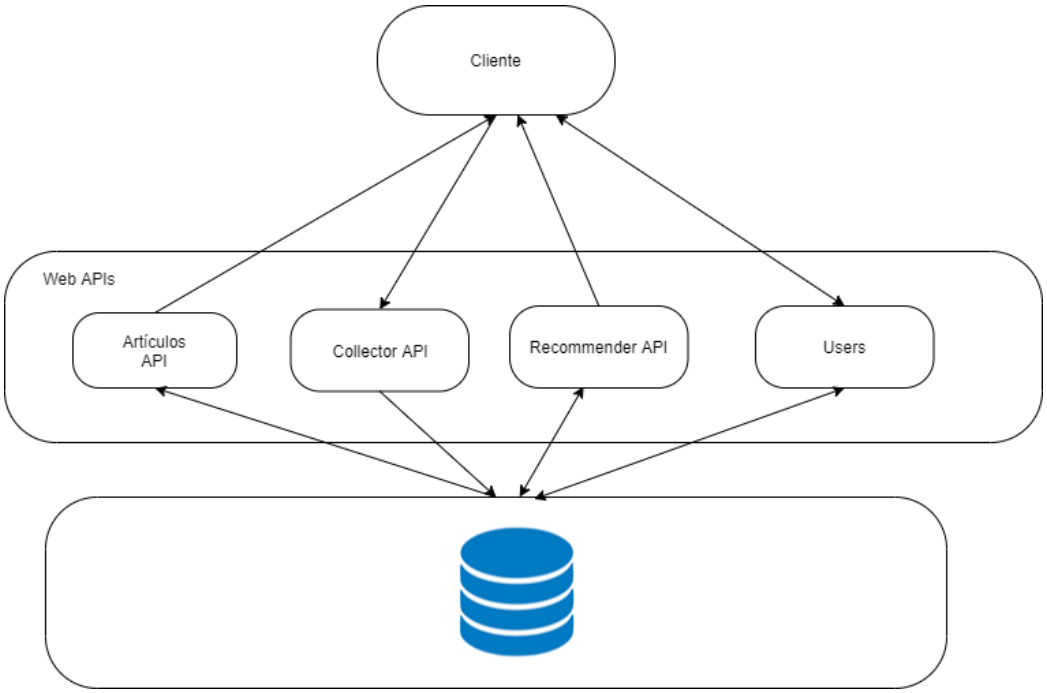


Figura 20. Estructura del Sistema de Recomendación.

3.1.1 Cliente

Es la aplicación Web en donde el usuario podrá buscar, leer artículos y obtener las recomendaciones basadas en su historial de artículos leídos previamente. Esta aplicación ha sido desarrollada con ReactJs, el cual es una biblioteca de JavaScript para construir interfaces de usuario interactivas; y a su vez está basada en componentes que en conjunto componen la aplicación como un todo. En la Figura 21, podemos observar el cliente Web con sus diferentes secciones:

- Recent Articles: artículos que han sido agregados recientenmente al sistema.
- Most Voted: artículos más votados por los usuarios.
- Recommendations: recomendaciones al usuario en base a sus gustos (esta sección es solo visible para usuarios registrados).
- Sección de articulos: lista de todos los artículos disponibles. Esta sección se encuentra inmediatamente despues de Recommendations.

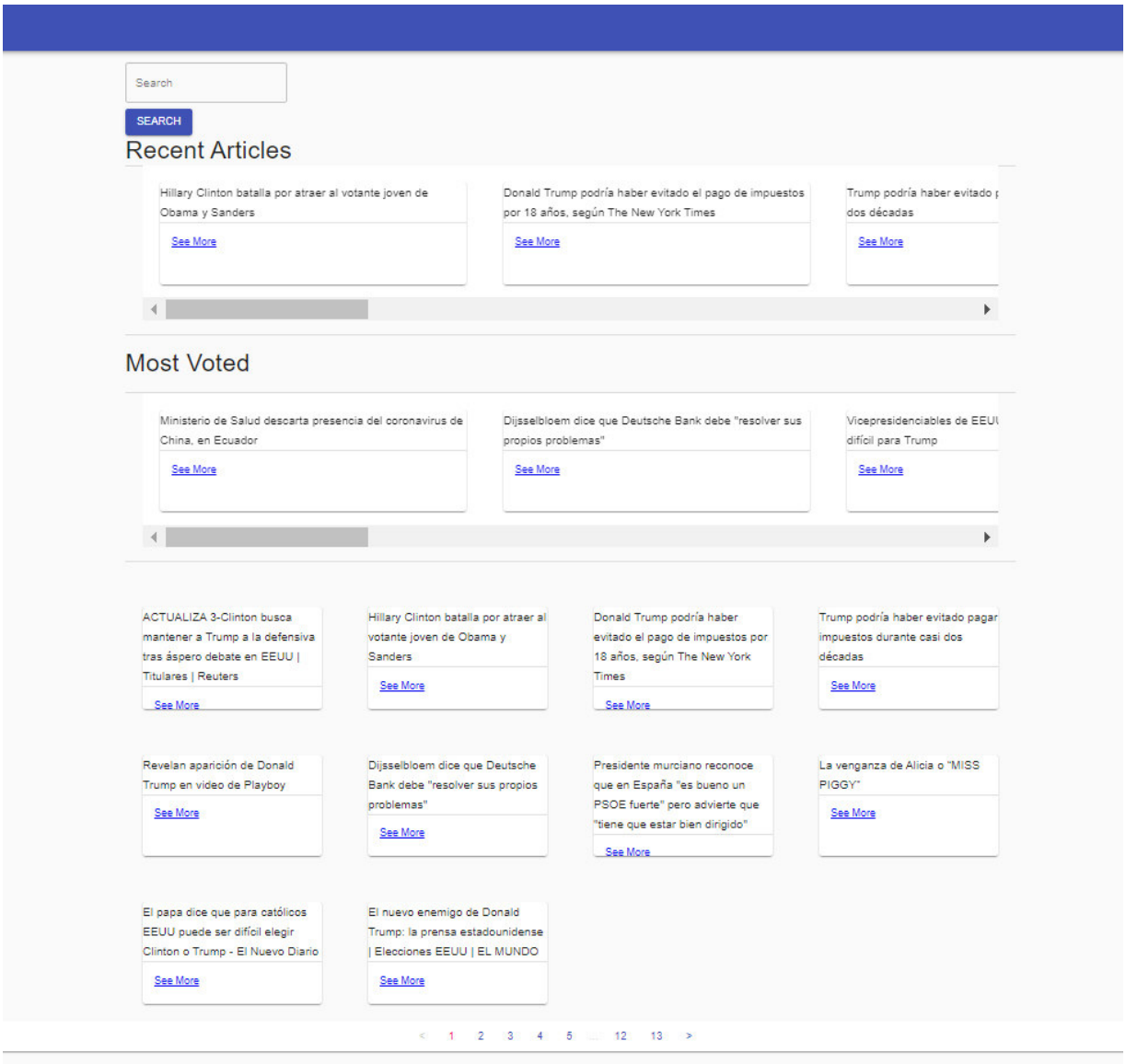


Figura 21. Cliente Web del sistema de recomendación.

3.1.2 Backend y APIS

El backend es la parte del sistema en donde se implementa lo necesario para obtener artículos, registrar usuarios y el lugar en donde se encuentra implementado el recomendador mediante un Esquema de GraphQL; usando Python como el principal lenguaje de programación, Django y graphene como librerías para la implementación en código del esquema.

Gracias a Django, podemos dividir el backend en pequeñas Apps como si fueran modulos en donde implementamos cada una de las APIs. Cada app contiene la lógica necesaría para el procesamiento de artículos en la app de Artículos, recolectar interacciones del usuario y likes

en la app Collector, registrar usuarios en la app Users, y generar recomendaciones en la app de Recommender. Para el diseño de las API usamos GraphQL, el ORM de Django y la base de datos por defecto de Django: SqlLite3. En la Figura 22 se muestra el modelo entidad-relación de la base de datos y las Tablas utilizadas.

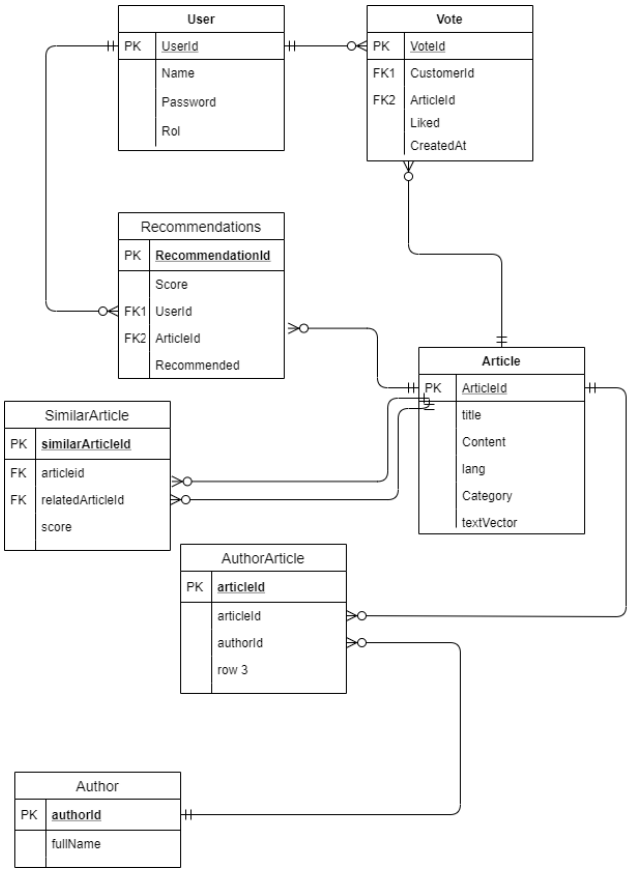


Figura 22. Esquema de la base de datos usada en el sistema.

3.1.3 APIs

Como se muestra en la Figura 20, el backend se compone de 4 APIs cada uno con la lógica necesaria de acuerdo a sus especificaciones. Gracias al framework Django, nosotros podemos dividir cada API como si fueran módulos en el backend mediante la definición de apps en el proyecto. Django como framework, tiene especificado que cada módulo es empaquetado en lo que se conoce como Apps. A continuación especificamos cada API:

- API de Artículos: Contiene todo lo necesario para realizar operaciones CRUD de artículos en la base de datos y enviar al cliente lo que requiera mediante el esquema de GraphQL.

- API de Collector: Contiene todo lo necesario para recolectar interacciones con el usuario y likes a artículos con el fin de ser usados por el recomendador. Además, el Collector se encarga de crear el pérfil del usuario que es el conjunto de artículos que el usuario ha consumido,
- API de Recomendaciones: Es en donde implementamos el recomendador y en conjunto con el collector y los artículos generamos recomendaciones para los usuarios en base a su perfil de usuario.
- API de Users: toda la lógica para crear y loguear usuarios.

3.1.4 Esquema de GraphQL

El esquema de GraphQL es el esqueleto de nuestras APIs implementadas en el sistema. Para definir un esquema de GraphQL tenemos que definir primero las estructuras de tipos (types), las cuales son representaciones de un objecto con sus respectivos atributos. A continuación, aunque la definición de los tipos se especifica en la sección 7.2 Tipos en Anexos, se listan los diferente tipos que componen el esquema utilizado en el proyecto:

- User: representa a un usuario
- Vote: representa un like de un usuario a un artículo.
- Article: representa un artículo de noticia.
- Recommendation: representa una recomendación al usuario.
- SimilarArticles: representa un artículo similar a otro.

3.1.5 Queries

Para hacer uso de las API, hemos definido Queries en GraphQL para realizar búsquedas de artículos y obtener las recomendaciones tomando en cuenta si es que se quiere obtener artículos similares a un artículo dado o basado en los gustos del usuario que se encuentra logueado. A continuación describimos cada uno de los queries. La definición completa de cada uno de los queries se listarán en la sección 7.3 Queries en Anexos:

1. **articles**
   - **1.1. Descripción:** Permite obtener una lista de artículos basandose en una búsqueda y/o mediante paginación.
2. **Tipo:** ArticleType.**article**
   - **1.1. Descripción:** Buscar un artículo basado en el Id.
   - 2.2. **Type:** ArticleType

3. **recentArticles**
    1.1. Descripción: Obtiene una lista de los 10 artículos más recientes.
    2.2. Type: ArticleTye

4. **mostVoted**
    1.1. Descripción: Obtiene los 10 artículos con mas votos/likes.
    2.2. Type: ArticleType
5. **users**
    1.1. Descripción: Retorna una lista de los usuarios registrados en la aplicación.
    2.2. Type: UserType
6. **user**
    1.1. Descripción: Retorna un usuario dado su Id
    2.2. Type: UserType
7. **votes**
    1.1. Descripción: Retorna una lista de todos los likes en conjunto con el artículo y el usuario que ha creado el like.
    2.2. Type: VoteType
8. **vote**
    1.1. Descripción: Retorna si el usuario actual ha consumido o no de un artículo dado su id. Este query funciona si es que el usuario se encuentra logueado. Necesita que la cabecera authorization tenga el valor del token.
    1.1. Type: VoteType
9. **similarArticles**
    1.1. Descripción: Dado el id de un artículo retorna una lista de artículos relacionados basado en el coseno de la similitud.
    2.2. Type: SimilarArticlesType
10. **Recommendations**
    1.1. Descripción: Obtiene una lista de lo 10 artículos que pueden ser de interés para el usuario que se encuentra actualmente logueado. Necesita que la cabecera authorization tenga el valor del token.
    2.2. Type: RecommendationType

3.1.6 Mutations

Las mutaciones nos permiten realizar operaciones de eliminación, actualización e ingreso de datos en la base de datos. En nuestras APIs hemos definido las siguientes mutaciones que forman parte tanto del cliente como del sistema de recomendación. Se lista cada mutation, pero la descripción completa se puede encontrar en la sección 7.4 Mutaciones en Anexos:

1. **addArticle:** Permite añadir un nuevo artículo a la base.
2. **createUser :** Mutación para registrar un nuevo usuario en el sistema
3. **tokenAuth :** Loguea el usuario al sistema y retorna un token para autorizar el acceso a recursos no disponibles para usuarios tipo visitante.
4. **createVote:** Crea un voto de un usuario a un artículo, si el voto ya existe se cambiá el valor del campo liked. Necesita que la cabecera authorization tenga el valor del token.

3.2 Funcionamiento del Sistema

El funcionamiento del sistema se puede resumir en 2 etapas:

1. Construir el perfil del usuario.
2. Recomendar artículos en base al perfil del usuario.

3.2.1 Construyendo el Perfil del usuario

En la primera etapa, el perfil del usuario se definió como el conjunto de artículos que el usuario ha consumido mediante la interacción con la aplicación Web. Cada interacción incluye especialmente lo que son likes a un artículo dado, los cuales son recolectados por la API de Collector, la cual se encarga de almacenar estas interacciones en la Tabla Vote en la base de datos.

Una vez que el usuario ha comenzado a interactuar con los artículos mediante un boton de “Me Gusta”, estas interacciones son recolectadas por el Collector el cuál creal el perfil del usuario con los artículos y datos que obtiene de la base de datos. La Figura 23 resume el proceso de recolección y creación del pérfil del usuario.

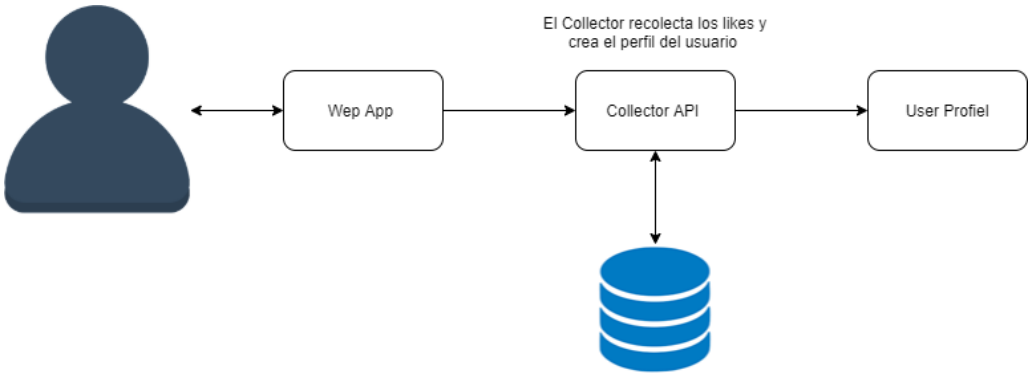


Figura 23. Creación del Pérfil del Usuario.

3.2.2 Similitud de Documentos

El enfoque usado para nuestro sistema de recomendación es el basado en contenidos (conten-based). Este enfoque trata de buscar similitudes entre los ítems que ha consumido el usuario y aquellos que pueden resultar de su interés [5]. Para ello, se debe encontrar similitud entre el contenido de los artículos, y como se explicó en el capítulo 2, existen diferentes técnicas de procesamiento de texto que nos permiten encontrar similitud entre cadenas de texto. En el presente proyecto se utilizó los modelos NER (Name Entity Recognition) de la librería SpaCy, este modelo es generado utilizando Redes Neuronales Convolucionales, las cuales toman un vector de cada palabra presente en el documento y tratan de hallar si cada una de estas pertenece a una entidad. Durante este proceso también se calcula un vector del texto, el cual es conocido como vector de resumen (summary vector) [33].

Gracias a la representación vectorial obtenida por el Modelo de SpaCy, podemos encontrar similitud entre documentos usando el Coseno de la Similitud. El tamaño del vector depende del modelo utilizado por SpaCy. Para artículos en español, usamos el modelo es_core_news_md, el cual ha sido entrenado con un corpus de noticias en español. Estos vectores pertenecen a un espacio vectorial de 50 dimensiones, es decir cada vector consta de 50 elementos.

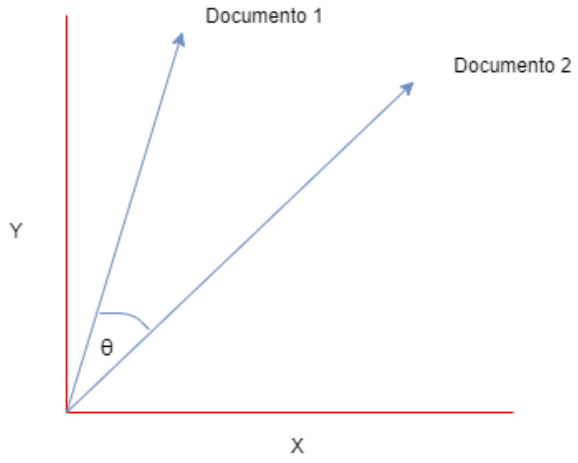


Figura 24. Representación vectorial de dos documentos en R2 y el ángulo entre ellos.

Ahora podemos usar el coseno de la similtud para encontrar si dos documentos están relacionados en base al ángulo que existe entre dos vectores A y B, que son similares entre sí [5] (Figura 24). Si el valor del coseno de este ángulo es cercano a 1, los dos documentos comparten un similitud en el contexto de las palabras y el tópico del cual trata, en cambio, un valor cercano a 0 indica que dos documentos tienen contenido diferente o el tópico de cada uno es diferente.

Cabe destacar que las similitudes entre artículos se han calculado previamente de manera que, cuando un nuevo usuario comienza a degustar artículos, el recomendador construye las recomendaciones basado en el perfil de usuario y los artículos similares a los ya degustados.

3.2.3 Generación de Recomendaciones

Una vez que se tiene creado el perfil del usuario, empieza la generación de recomendaciones en base a los gustos del usuario. El Recomendador extrae el perfil del Usuario y recomienda artículos que comparten una notable similitud con aquellos que han sido de ínteres previamente para el usuario. Cada vez que el usuario ingrese al aplicativo Web, nuestra aplicación se comunicará con el Recomendador mediante la API que hemos definido y obtendrá de ésta los artículos que se recomendarán al usuario. La Figura 25 muestra un flujo del proceso generador de recomendaciones.

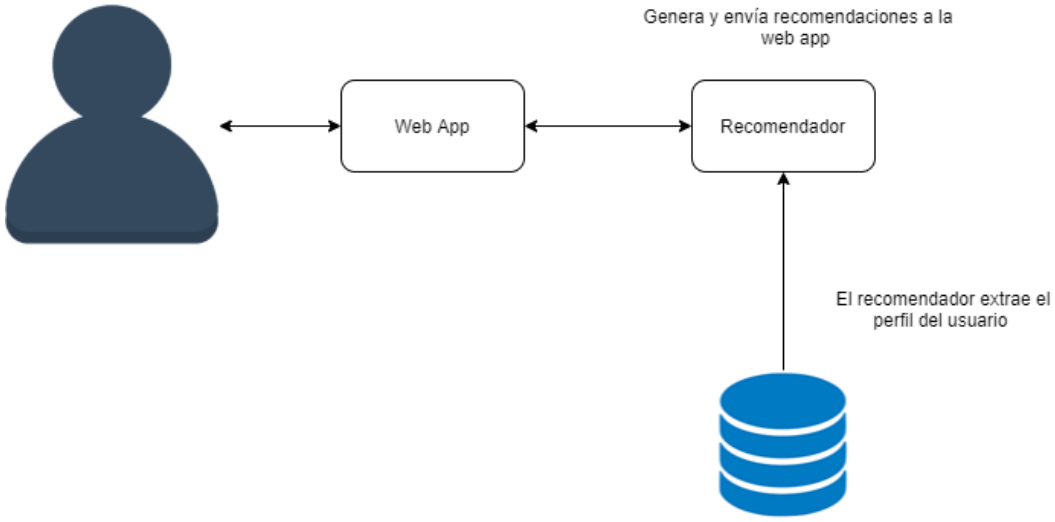


Figura 25. Generación de recomendaciones.

3.3 Ejemplo de demostración

Para comprender de mejor manera lo descrito anteriormente, a continuación, se detallará un ejemplo de uso. Primeramente, un usuario se registra en la aplicación Web mediante un formulario como se muestra en la Figura 26. En este formulario se tiene que especificar el correo electrónico, una contraseña y un nombre del usuario.

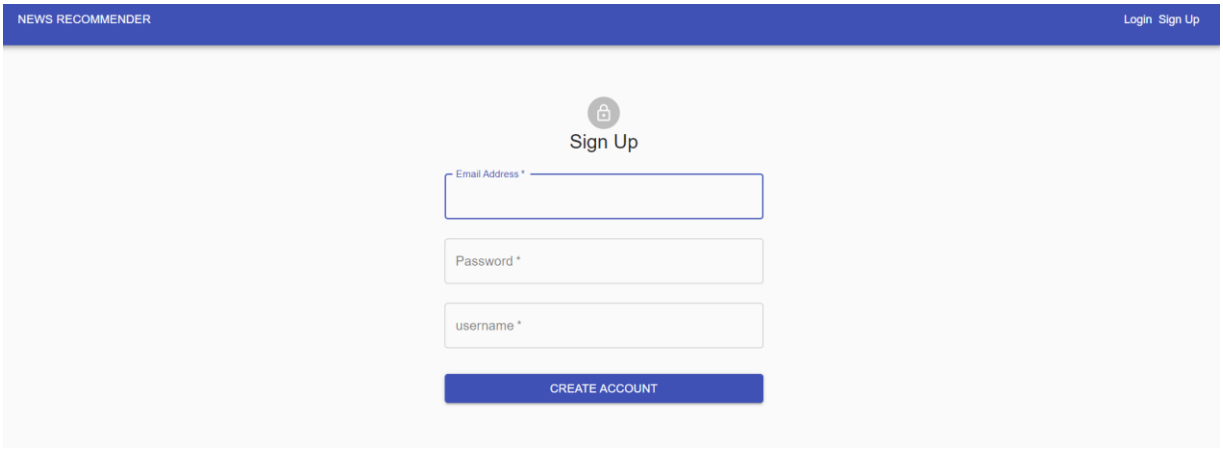


Figura 26. Pantalla de registro de usuario.

Una vez ya registrado el usuario, la aplicación Web redireccionará a un formulario de login para el ingreso (Figura 30).

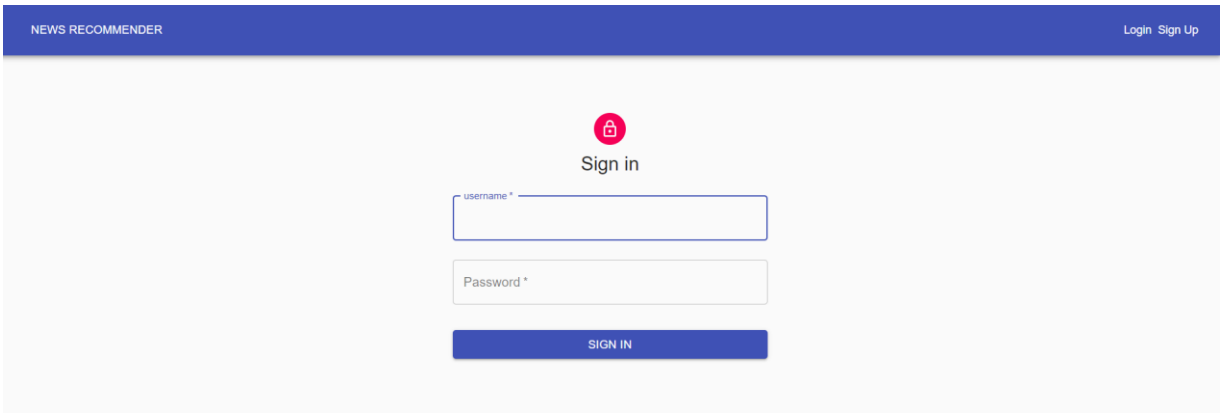


Figura 27. Pantalla de login de usuario.

Una vez que el usuario ha ingresado al sistema, éste podrá observar 3 bandas de artículos (Figura 28), las cuales corresponden a los artículos más recientes, los más votados y los recomendados para el usuario, debido a que es un usuario nuevo, aún no existen artículos recomendados.

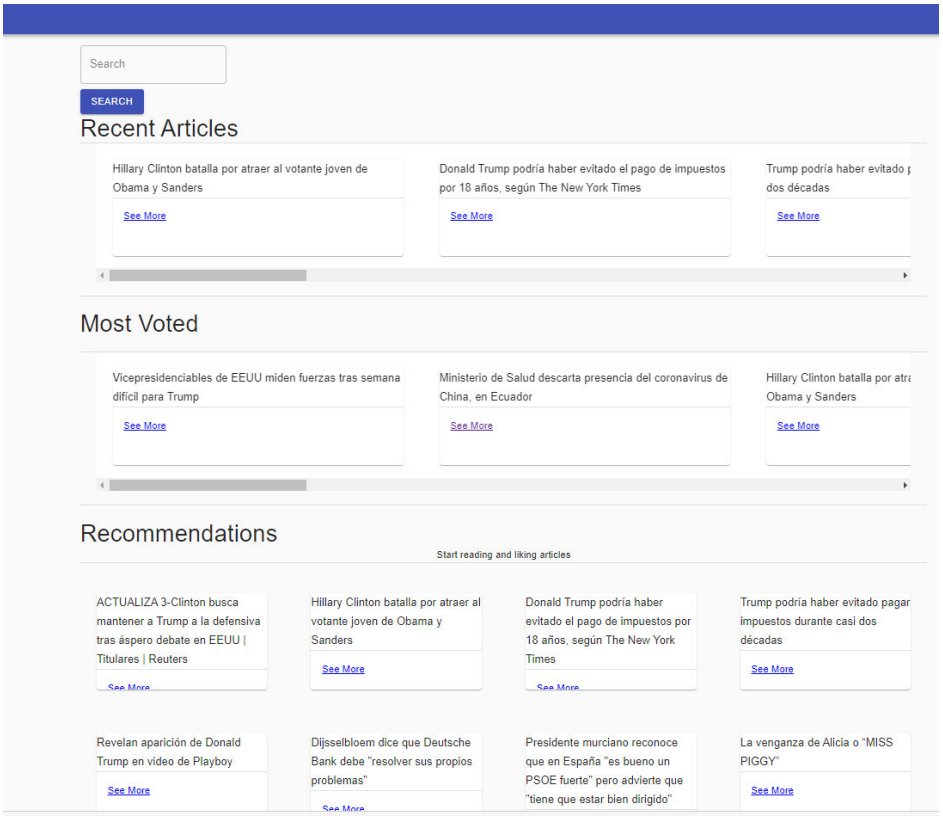


Figura 28. Home del cliente Web.

Una vez que ha ingresado al aplicativo, el usuario comenzará a leer e indicar lo artículos que le resultaron más interesantes mediante un click al botón de “Me Gusta”. Estas interacciones serán recolectadas por el Collector API para crear el perfil del usuario.

Al ingresar a un artículo, el usuario podrá visualizar todo el contenido en texto, un botón de “Me Gusta” y una banda de artículos relacionados (Figura 29).

Figura 29. Artículo sobre Covid-19.

Para ilustrar la creación del perfil del usuario, daremos “Me Gusta” a dos artículos: uno relacionado al coronavirus (ver Figura 29) y otro relacionado con política de Estados Unidos, en especial Donald Trump. Una vez que el usuario ha interactuado con los artículos, el recomendador proveerá artículos similares a los que el usuario haya señalado como de su interés. De manera que, cuando el usuario regrese al Home podrá observar los artículos que se recomiendan en base a su perfil de usuario. Las recomendaciones del ejemplo anterior pueden ser observadas en la Figura 30.

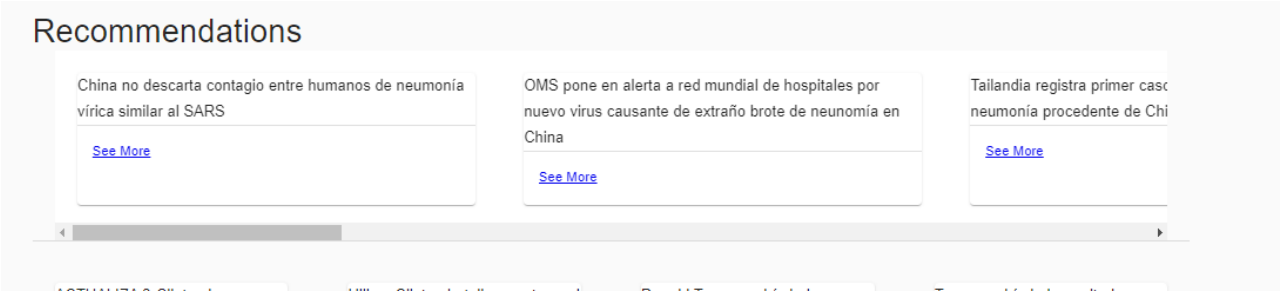


Figura 30. Recomendaciones de artículos relacionados con COVID-19.

3.4 Resultados de Pruebas

En la presente sección se presentará y analizará los resultados de las pruebas realizadas durante las fases de testing online y offline, que se describieron en el anterior capítulo, como parte de la metodología aplicada al presente proyecto. Destacamos dos subdivisiones, las cuales son:

1. Resultados de pruebas offline: incluyen los resultados de pruebas unitarias, de cobertura realizadas en el cliente y en el Backend.
2. Resultados de pruebas online: pruebas realizadas con usuarios en un ambiente simulado basándose en la metodología TAM.

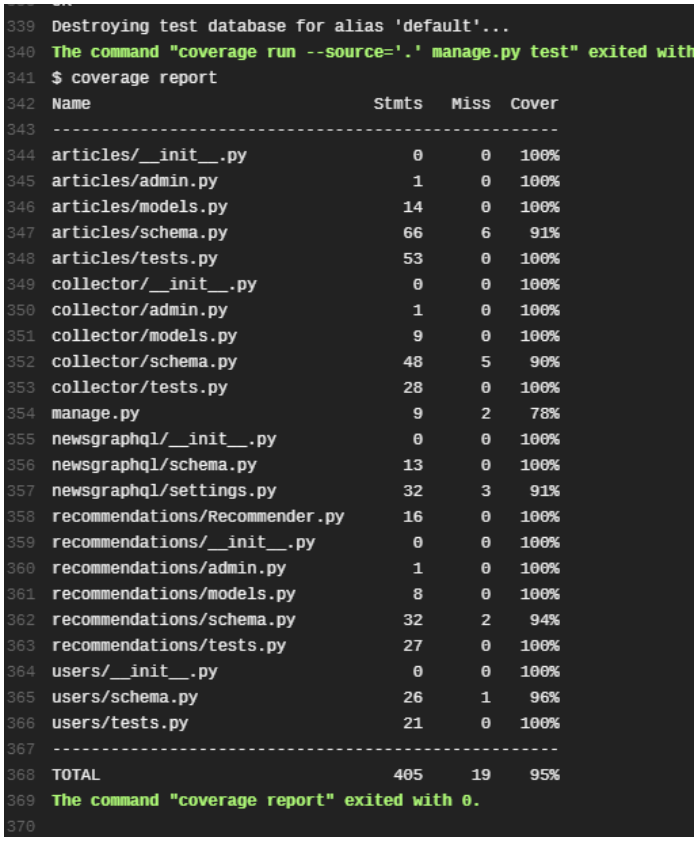
3.4.1 Pruebas Offline

Como se indicó anteriormente, las pruebas offline se realizan en un ambiente simulado y mediante ayuda de herramientas de testing. En el presente proyecto se realizaron pruebas unitarias y de cobertura, ambas se relacionan entre sí, por lo que en las siguientes secciones se presentan de manera general los resultados de estas pruebas tanto en Backend como Frontend.

Backend

En esta sección se presentan los resultados de pruebas unitarias y de cobertura para el Backend. Estas pruebas nos permiten determinar si el código es ejecutado de manera adecuada y retorna los valores esperados, cubriendo todo el código escrito sin generar resultados no deseados ni incoherencias en el sistema. Para estas pruebas se utilizaron dos servicios en la nube: *travis ci* como servidor de integración continua en donde se ejecutan las pruebas, y *codecov* para presentar los resultados de las pruebas de cobertura ejecutadas en *travis ci*.

La Figura 31, muestra un ejemplo de ejecución de comandos y una salida de resumen de las pruebas. De los resultados podemos observar que un 95% del código tomado en cuenta que ha sido cubierto en la ejecución de pruebas unitarias del Backend.



```
Destroying test database for alias 'default'...
The command "coverage run --source='.' manage.py test" exited with
$ coverage report
Name                                Stmts   Miss  Cover
-------------------------------------------------------
articles/__init__.py                    0      0   100%
articles/admin.py                       1      0   100%
articles/models.py                     14      0   100%
articles/schema.py                     66      6    91%
articles/tests.py                      53      0   100%
collector/__init__.py                   0      0   100%
collector/admin.py                      1      0   100%
collector/models.py                     9      0   100%
collector/schema.py                    48      5    90%
collector/tests.py                     28      0   100%
manage.py                               9      2    78%
newsgraphql/__init__.py                 0      0   100%
newsgraphql/schema.py                  13      0   100%
newsgraphql/settings.py                32      3    91%
recommendations/Recommender.py         16      0   100%
recommendations/__init__.py             0      0   100%
recommendations/admin.py                1      0   100%
recommendations/models.py               8      0   100%
recommendations/schema.py              32      2    94%
recommendations/tests.py               27      0   100%
users/__init__.py                       0      0   100%
users/schema.py                        26      1    96%
users/tests.py                         21      0   100%
-------------------------------------------------------
TOTAL                                 405     19    95%
The command "coverage report" exited with 0.
```

Figura 31. Ejecución de pruebas unitarias y de cobertura en el Backend.

Sin embargo, los resultados de *travis ci* no son muy descriptivos, por lo que realizamos integración con *codecov* para tener mejores reportes acerca de la cobertura de pruebas. El dash board de *codecov* se divide en 4 secciones. La primera es una gráfica de ejecución de las pruebas, en donde el eje X representa la fecha de ejecución; y el eje Y, el porcentaje de cobertura (Figura 32). Obsérvese que la línea ha ido creciendo conforme se realizaron más pruebas al Backend.

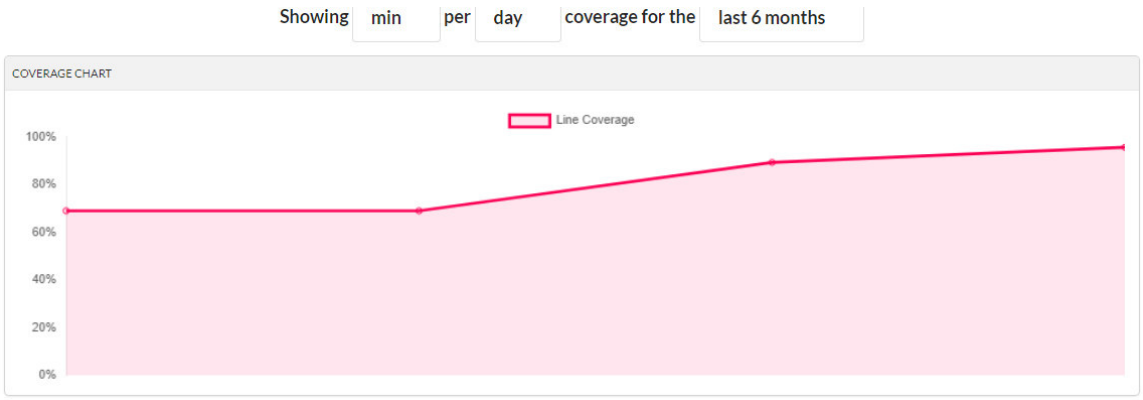


Figura 32. Línea de cobertura de pruebas en el tiempo.

Las siguientes dos secciones de *codecov* se dividen en una gráfica en forma de pastel, la cual se subdivide a su vez en unidades que indican los módulos y submódulos del Backend. Mediante el color verde de la gráfica se indica si un módulo ha sido cubierto por las pruebas, caso contrario se muestra un color anaranjado si se ha cubierto de manera parcial o rojo si no se han sido cubierto por las pruebas. Esta gráfica se conoce como COVERAGE SUNBURST. En cambio, la sección de RECENT COMMIT muestra el historial de *commit* realizado por el equipo de desarrollo, es decir las pruebas ejecutadas. Estas dos secciones se muestran en la Figura 33.

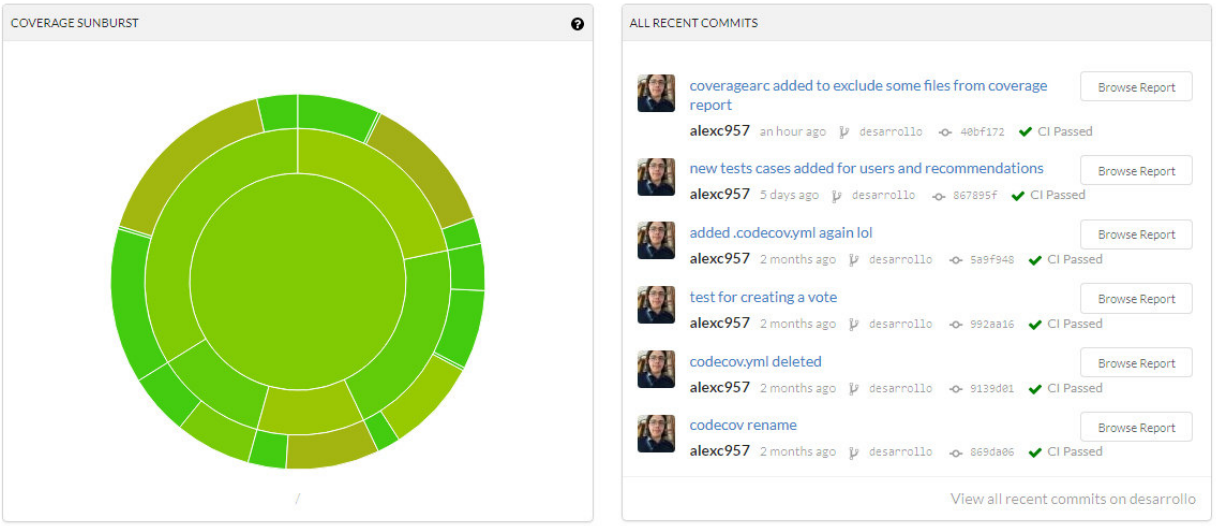
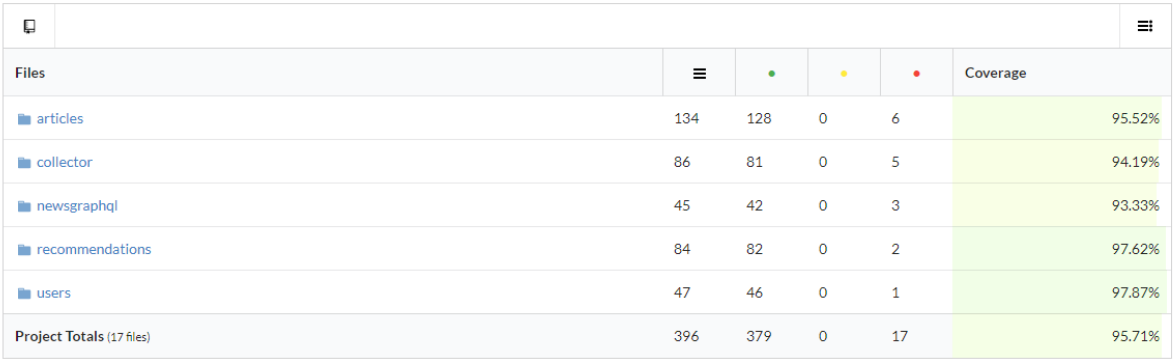


Figura 33. COVERAGE SUNBURST y RECENT COMMITS.

La última sección nos permite ver el porcentaje de cobertura de cada submódulo, permitiendo verificar los archivos de cada submódulo para revisar el porcentaje de cobertura por archivo o script. La Figura 37 muestra esta sección, en donde se puede observar el porcentaje por cada submódulo.

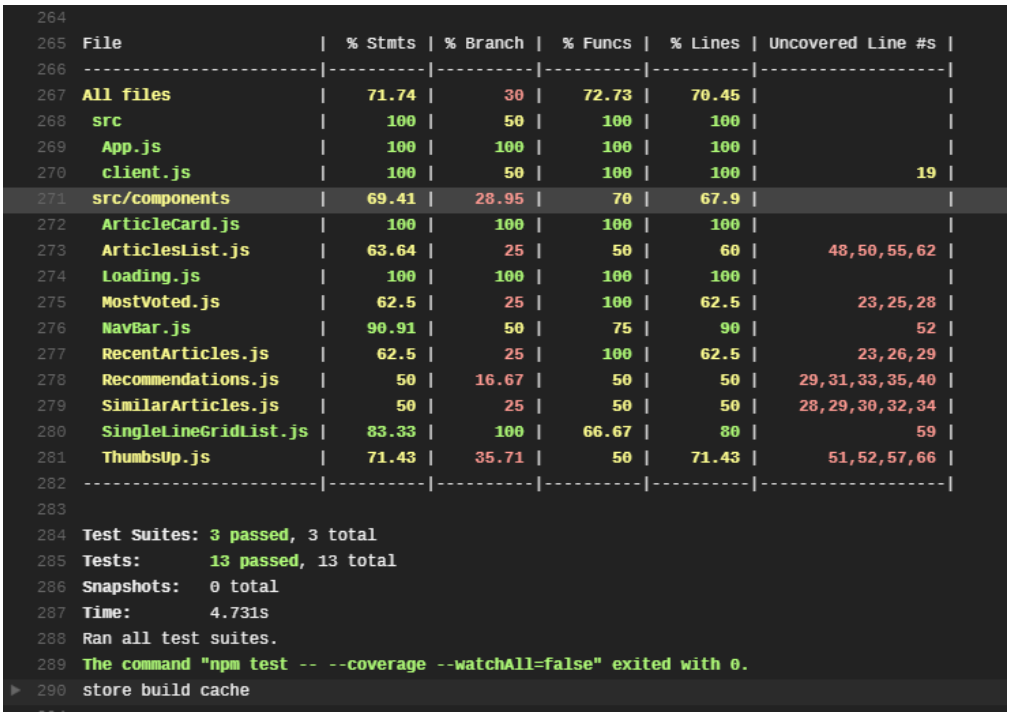


Files					Coverage
articles	134	128	0	6	95.52%
collector	86	81	0	5	94.19%
newsgraphql	45	42	0	3	93.33%
recommendations	84	82	0	2	97.62%
users	47	46	0	1	97.87%
Project Totals (17 files)	396	379	0	17	95.71%

Figura 34. Sección de resumen de resultados de pruebas de cobertura por modulo.

Frontend

En el Frontend se realizaron pruebas de componentes para determinar que cada componente de interfaz gráfica se carga sin ningún problema. Estas pruebas unitarias se realizaron en conjunto con las de cobertura. Sin embargo, debido a problemas de compatibilidad con las tecnologías usadas en el Frontend, solo se utilizó el servicio en nube de *travis CI* para ejecutar las pruebas y mostrar el reporte mediante un comando (Figura 35), en donde nos muestra que todos los componentes se cargaron sin ningún problema, pero que solo el 70% del código fue cubierto en la ejecución de las pruebas.



```
264
265 File                    |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
266 ------------------------|----------|----------|----------|----------|-------------------|
267 All files               |    71.74 |       30 |    72.73 |    70.45 |                   |
268  src                    |      100 |       50 |      100 |      100 |                   |
269   App.js                |      100 |      100 |      100 |      100 |                   |
270   client.js             |      100 |       50 |      100 |      100 |                19 |
271  src/components         |    69.41 |    28.95 |       70 |     67.9 |                   |
272   ArticleCard.js        |      100 |      100 |      100 |      100 |                   |
273   ArticlesList.js       |    63.64 |       25 |       50 |       60 |       48,50,55,62 |
274   Loading.js            |      100 |      100 |      100 |      100 |                   |
275   MostVoted.js          |     62.5 |       25 |      100 |     62.5 |          23,25,28 |
276   NavBar.js             |    90.91 |       50 |       75 |       90 |                52 |
277   RecentArticles.js     |     62.5 |       25 |      100 |     62.5 |          23,26,29 |
278   Recommendations.js    |       50 |    16.67 |       50 |       50 |    29,31,33,35,40 |
279   SimilarArticles.js    |       50 |       25 |       50 |       50 |    28,29,30,32,34 |
280   SingleLineGridList.js |    83.33 |      100 |    66.67 |       80 |                59 |
281   ThumbsUp.js           |    71.43 |    35.71 |       50 |    71.43 |       51,52,57,66 |
282 ------------------------|----------|----------|----------|----------|-------------------|
283
284 Test Suites: 3 passed, 3 total
285 Tests:       13 passed, 13 total
286 Snapshots:   0 total
287 Time:        4.731s
288 Ran all test suites.
289 The command "npm test -- --coverage --watchAll=false" exited with 0.
290 store build cache
```

Figura 35. Resultado de las pruebas de cobertura en el Frontend.

3.4.2 Pruebas Online

Para esta evaluación, se utilizó un método de encuestas para medir el nivel de aceptación del sistema. Las encuestas fueron realizadas de manera individual por un grupo de 8 personas. Dicha encuesta utilizó preguntas directamente relacionadas con la metodología TAM. Además, existieron preguntas que requerían al usuario su opinión acerca de la funcionalidad del sistema.

Los resultados de las encuestas son los siguientes:

Facilidad de Uso

Una de las características de un sistema informático es su facilidad de uso e interacción de usuario con la interfaz gráfica. En sección de las pruebas, un 75% de los participantes estuvo de acuerdo que la aplicación es de fácil interacción, mientras que un 12.5% estuvieron parcialmente de acuerdo frente a un 12.5 neutral. En otra pregunta relacionada con la estructura de la aplicación, un 87.5% indicó que la estructura de la aplicación es entendible sin mayor esfuerzo, frente a un 12.5 que estuvo parcialmente de acuerdo.

Además, se especificó una pregunta de opinión sobre la facilidad del sistema, en esta todos los usuarios indicaron que el sistema cumple con la característica de facilidad de uso.

Los resultados de las preguntas se muestran a continuación:

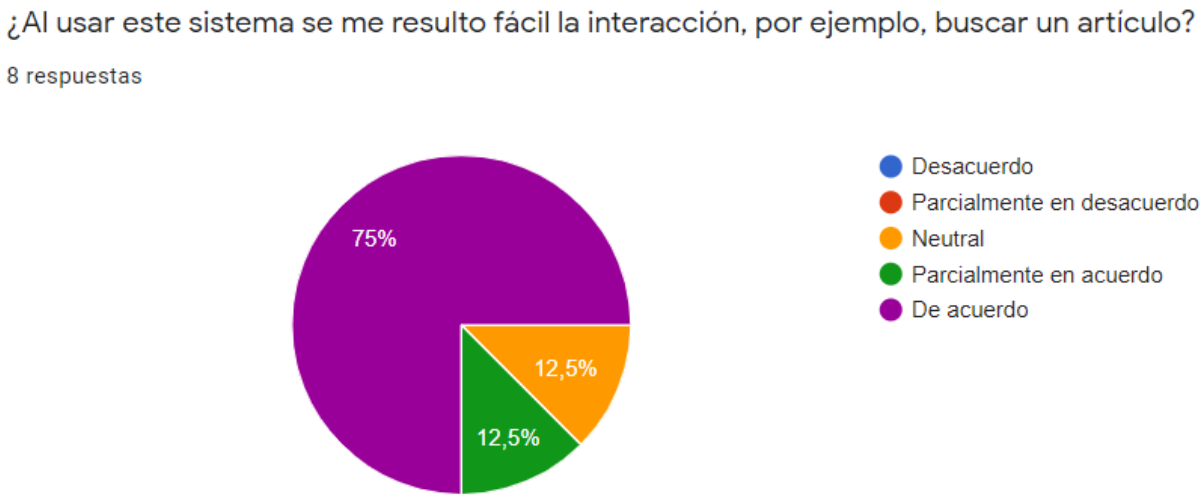


Figura 36. Resultados de la pregunta 1 de facilidad de uso.

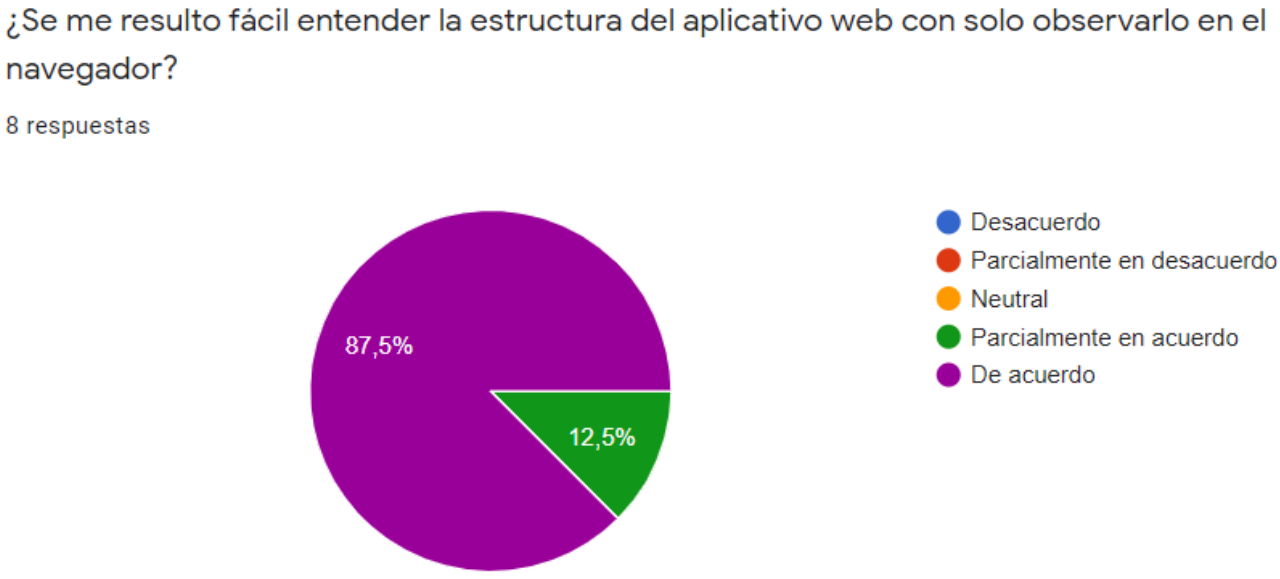


Figura 37. Resultados de la pregunta 2 de facilidad de uso.

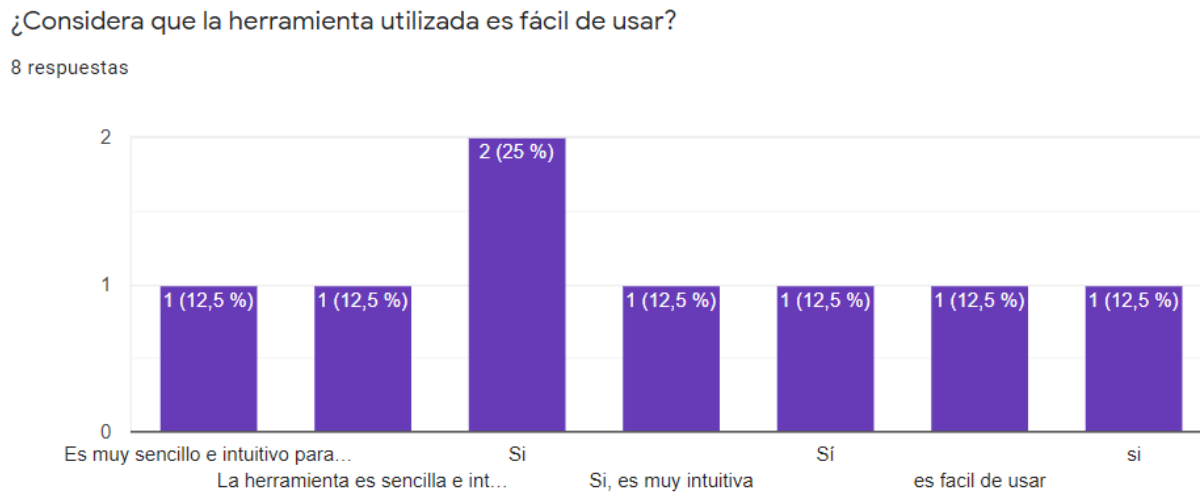


Figura 38. Resultados de la pregunta de opinión.

Utilidad percibida

En este apartado se analiza la utilidad percibida del sistema por parte de los usuarios quienes realizaron las pruebas. A continuación, se describe los resultados de las pruebas: un 100% de los usuarios indicó que los artículos recomendados eran similares a aquellos que habían indicado con el “botón me gusta”, en relación con esta pregunta un 62.5% indicó que el sistema era novedoso en generar las recomendaciones de los artículos de noticias teniendo en cuenta las temáticas propuestas de futbol, política y coronavirus. En otra pregunta relacionada con el potencial uso de este sistema, un 87.5% indicó que el sistema podrá ser usado e integrado en distintas plataformas. En la pregunta de opinión los usuarios indicaron que el sistema es útil y cumple con su propósito.

Los resultados en gráficos de las preguntas se muestran a continuación:

¿Al usar este sistema pudo Ud notar que los artículos similares tienen relación entre sí?

8 respuestas

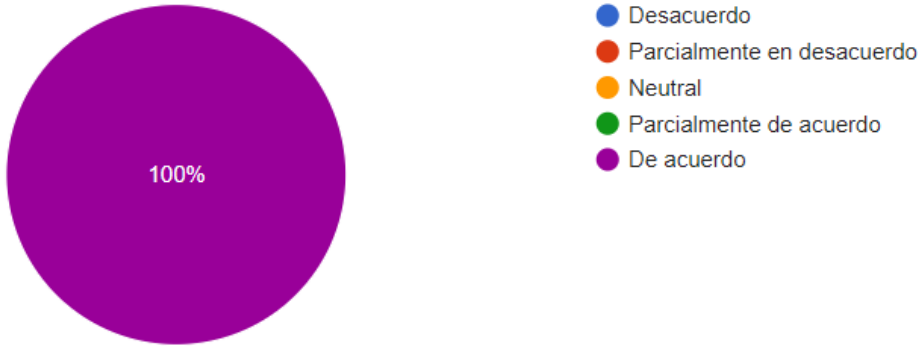


Figura 39. Pregunta 1 de Utilidad percibida

Una propiedad de los sistemas de recomendación es la "novedad" ¿Considera que los artículos recomendados son novedosos en relación a la temática tomada en cuenta (política, coronavirus, deportes)?

8 respuestas

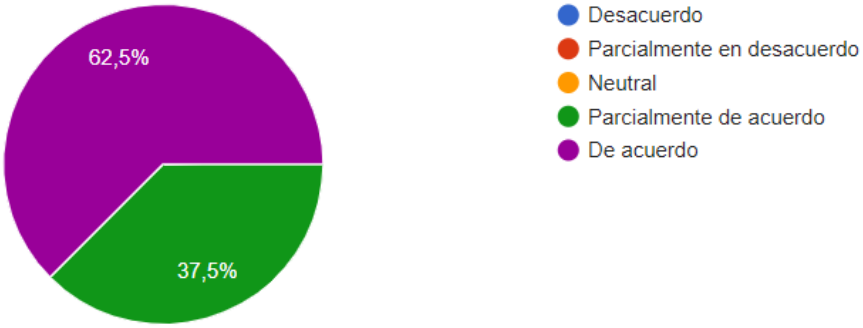


Figura 40. Pregunta 2 de Utilidad percibida

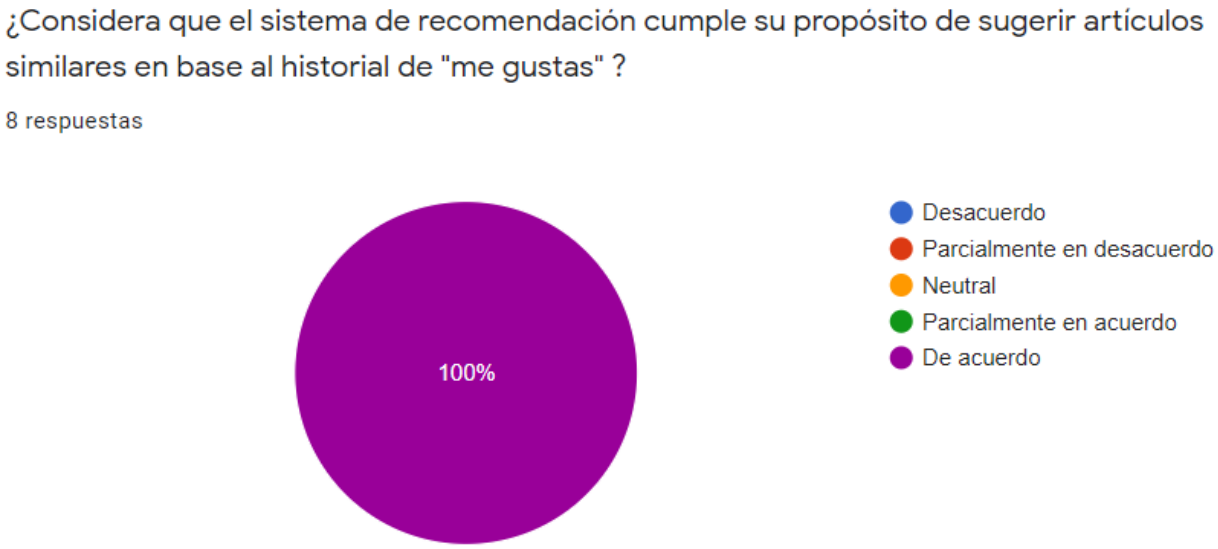


Figura 41. Pregunta 3 de Utilidad percibida

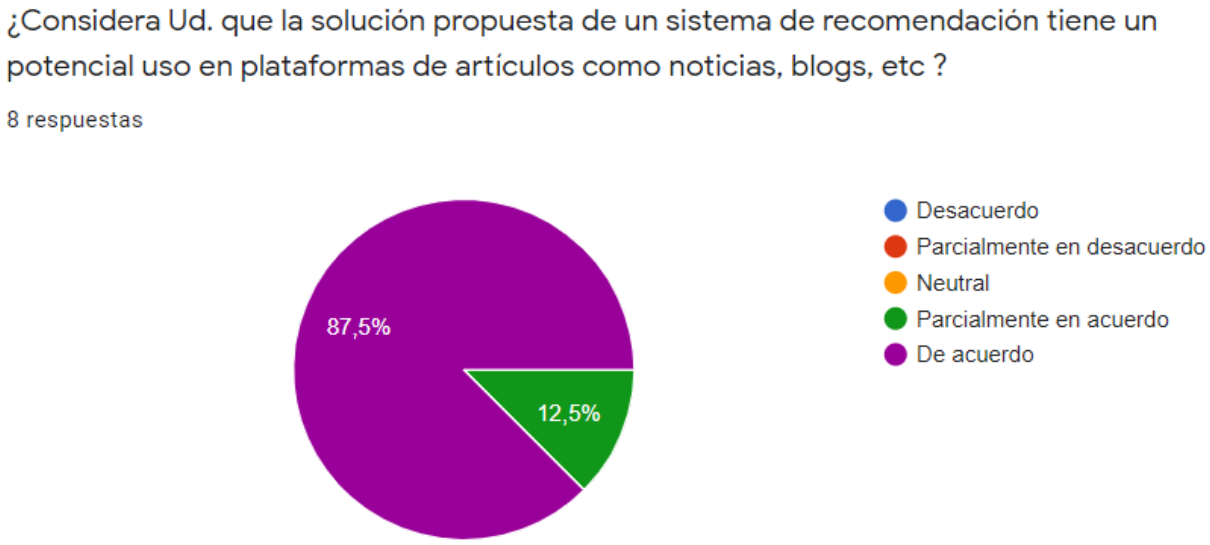


Figura 42. Pregunta 4 de Utilidad percibida.

¿Al usar este aplicativo web pudo observar que tanto los artículos similares como las recomendaciones fueron variadas, es decir diferentes artículos están relacionados pero su contexto es diferente?

8 respuestas

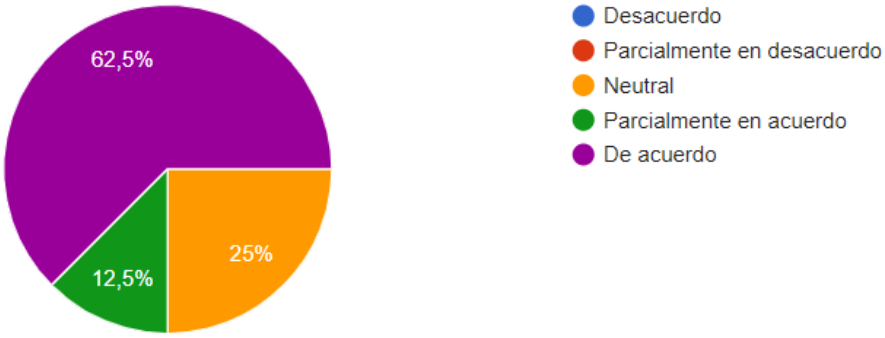


Figura 43. Pregunta 5 de Utilidad percibida.

Tomando en cuenta el objetivo de permitir explorar con mayor eficiencia un corpus documental. ¿Considera usted que esta herramienta es útil?

8 respuestas

Si

Si ya que me ayuda a navegar por noticias relacionadas.

La herramienta cumple su objetivo, por tanto creo que es bastante útil

Si ya que esta herramienta ayuda a disminuir el tiempo que se gastaria buscando en varias fuentes diferentes ya que esto se haria en una sola aplicacion. Ya que se podria empezar solo con un tema y ir a los articulos relacionados de una manera muy facil.

si

de acuerdo

Si, es muy útil

Figura 44. Pregunta 6 de opinión Utilidad percibida.

En adición a las preguntas de facilidad de uso y utilidad percibida, se agregaron preguntas de opinión y recomendaciones por parte de los usuarios que formaron parte de las pruebas realizadas. A continuación, se muestra los resultados de dichas preguntas.

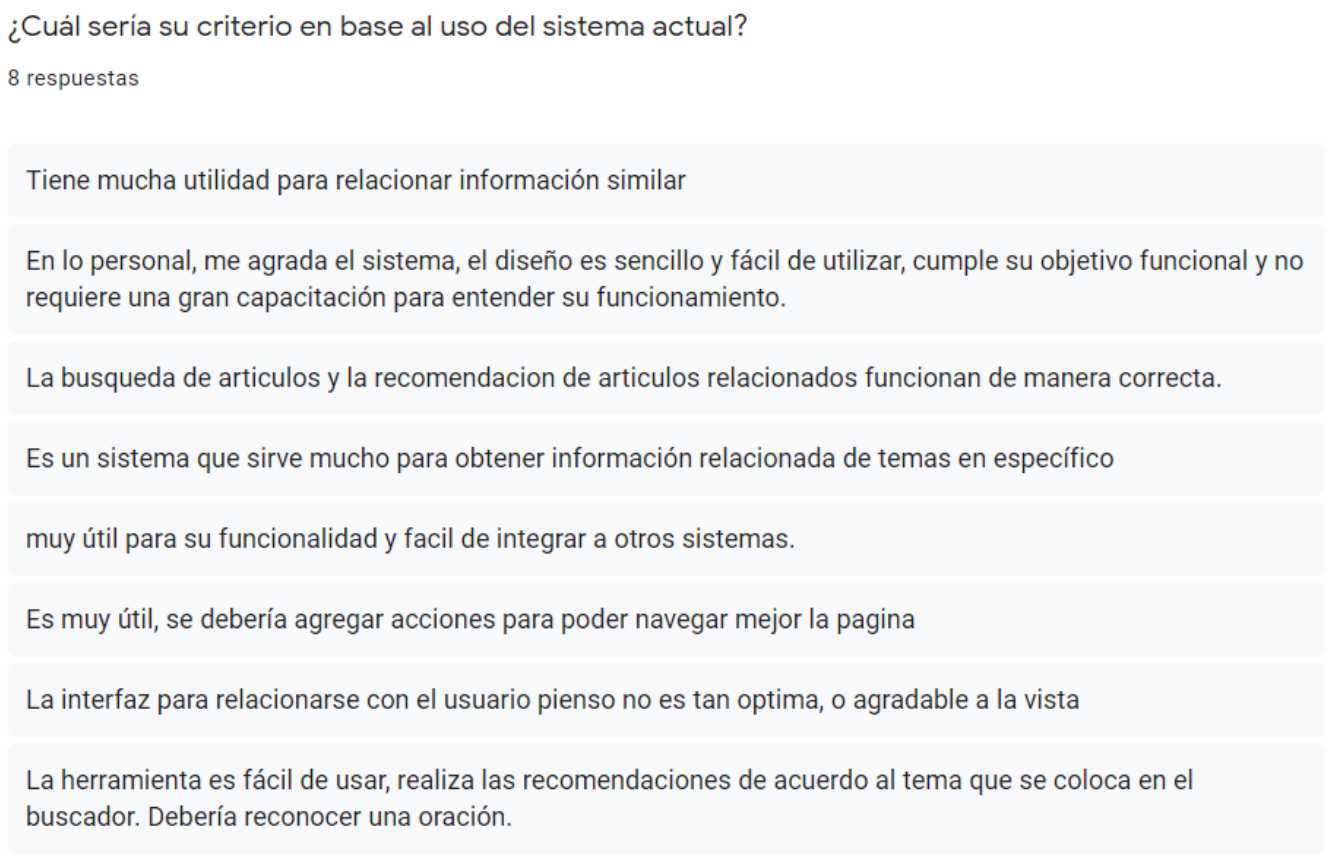


Figura 45. Criterio personal sobre el uso actual del sistema.

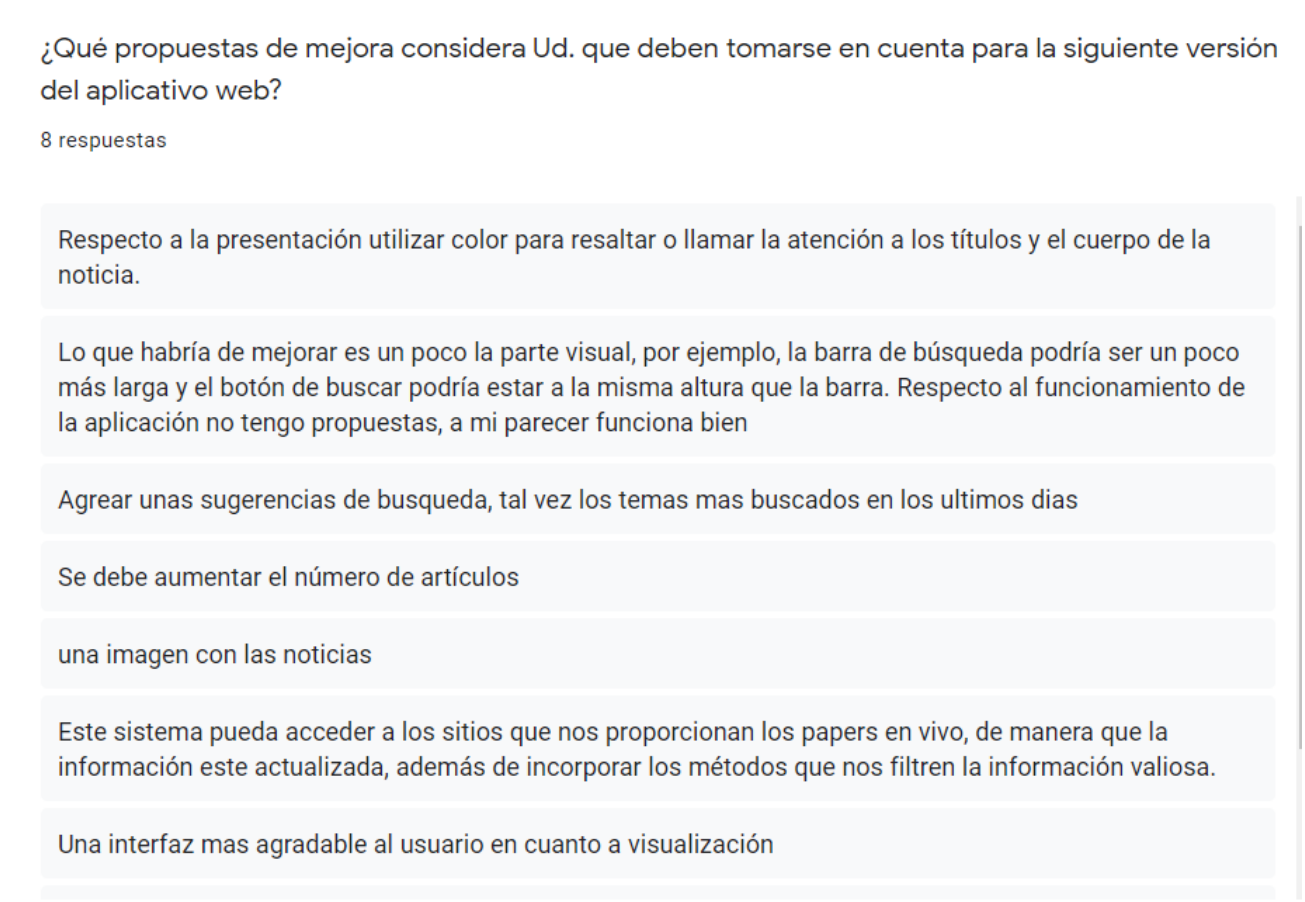


Figura 46. Criterio personal sobre el uso actual del sistema.

3.5 Nueva versión del Sistema.

En las pruebas realizadas con usuarios, se estableció que, para una nueva versión de sistema, debería mejorarse la página inicial con cambios como: arreglar la barra de búsqueda para que cubra toda la sección top a lo ancho, o que el botón de búsqueda se encuentre a la derecha de la barra. Además, se cambió la presentación de las noticias en la página de inicio para simplificar su presentación (Figura 47).

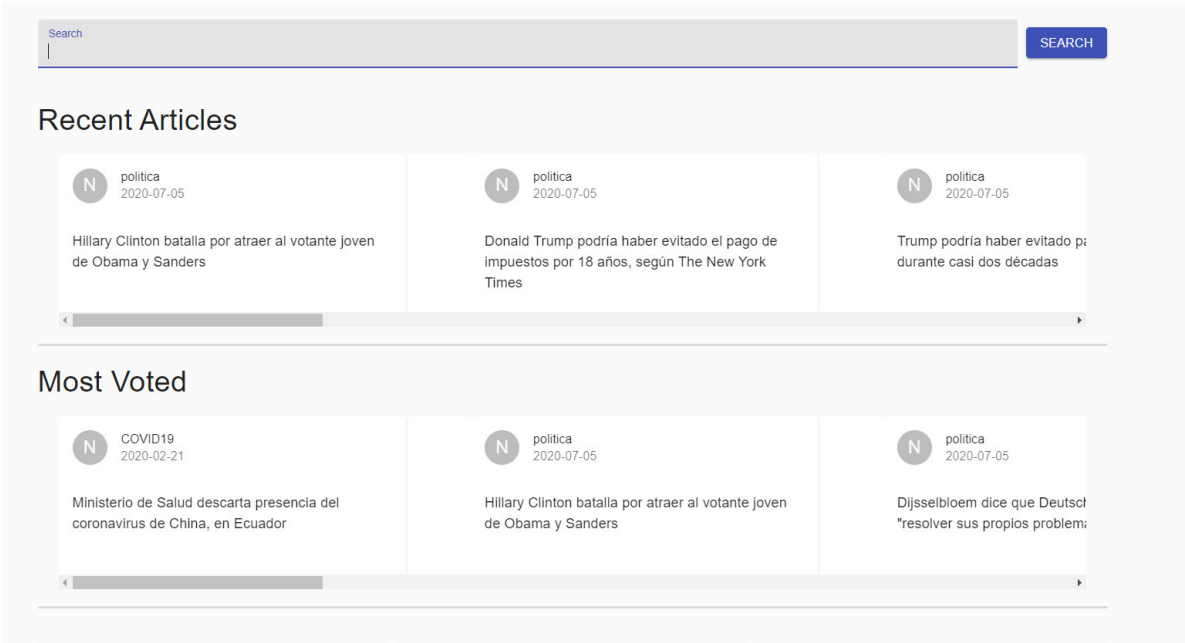


Figura 47. La nueva página de inicio modificada de acuerdo con la retroalimentación proporcionada por usuarios en las pruebas.

4. CONCLUSIONES

4.1 Conclusiones

- El sistema se desarrolló mediante la implementación de un algoritmo de recomendación basado en contenidos usando la librería Spacy en Python debido a facilidad de uso y disponibilidad de modelos de NLP en Español preentrenados con representaciones vectoriales para documentos de noticias.
- Los Sistemas de Recomendación basados en contenidos siempre tienen en cuenta los atributos que identifican a un ítem para calcular similitudes y generar sugerencias. Además, estos Sistemas de Recomendación son útiles en los casos en los que no se tenga ratings o información detallada de relaciones entre usuarios e ítems. Sin embargo, pueden presentar problemas de arranque en frío cuando no se tiene información sobre las preferencias de los usuarios. Una alternativa complementaria puede residir en perfilar a un usuario con usuarios similares de los cuales se conozca ya sus preferencias.
- La similitud entre documentos realizado mediante el modelo de SpaCy nos ha permitido obtener documentos similares y gracias a esto podemos construir una relación entre ítems. Esta relación permite aproximar noticias o información que, en un contexto de Vigilancia Estratégica, permitirá facilitar el enlace entre informaciones de interés conectadas con una temática específica, o incluso, desarrollar el proceso de inteligencia colectiva.
- El Sistema está diseñado para proveer recomendaciones basándose en artículos de noticias. Esto debido a que el alcance de nuestro proyecto se limitaba artículos noticiarios como demostrador. Sin embargo, su aplicación puede darse con cualquier colección textual. Así, el sistema permitirá a las organizaciones obtener artículos que tengan similitud con aquellos ya utilizados en su proceso de Vigilancia Estratégica.
- La implementación del sistema de recomendación tiene su núcleo en el Backend y ha sido implementado gracias a un esquema de GraphQL. De nuestra experiencia podemos afirmar que GraphQL nos permite implementar APIs que son intuitivas en un uso y fácil de integrar con un cliente Web.

- De acuerdo con las pruebas realizadas con los usuarios, el sistema cumple con su propósito y pudiera ser incorporado como parte de un sistema más complejo que abarque las otras actividades de Vigilancia Estratégica.

4.2. Recomendaciones

- Para un futuro es probable que se pueda integrar ratings en el sistema para calcular recomendaciones basándose en algoritmos de collaborative-filtering.
- Es posible incorporar mecanismos de recomendación grupales que permita que el sistema pueda ofrecer recomendaciones a grupos de trabajo que estén inmersos en un proyecto de Vigilancia Estratégica.
- El esquema de bases de datos usado en este proyecto es de entidad-relación. Sin embargo, se pudiera mejorar este esquema o investigar, de ser posible, otro esquema NoSql para la base de datos.
- Utilizar distintos algoritmos para calcular similitud de documentos permitirá al sistema sugerir artículos más relevantes para el usuario, reforzando las fortalezas y reduciendo las limitaciones de cada sistema.
- Mejorar la administración de usuarios permitirá al sistema entender mejor quienes usan el sistema y sus así mejorar el sistema de recomendación. En las pruebas de usuario se indicó que el sistema puede ser mejorado en la interfaz de usuario, agregando imágenes a los artículos y la barra de búsqueda tenga sea más notoria para el usuario. Para esto se debe tener en cuenta las recomendaciones de los usuarios que se presentó en la sección de resultados de pruebas.

6. REFERENCIAS BIBLIOGRÁFICAS

[1] S. J. Yelena Islen y F. I. Romero Rodríguez, «Modelos y herramientas para la vigilancia tecnológica,» *Ciencias de la Información,* pp. 2-3, 2016.

[2] «Vigilancia Estratégica,» Camara de comercio de España, [En línea]. Available: https://www.camara.es/innovacion-y-competitividad/como-innovar/vigilacia-estrategica. [Último acceso: 27 Octuber 2019].

[3] N. Lesca y M.-L. Caron-Fasan , «Strategic scanning project failure and abandonment factors: lessons learned,» *European Journal of Information Systems,* pp. 371-386, 2008.

[4] Keyul, «10 Most Popular Web Frameworks in 2020,» Medium, 27 November 2019. [En línea]. Available: https://medium.com/front-end-weekly/10-most-popular-Web-frameworks-in-2020-167b9103e08a. [Último acceso: 30 August 2020].

[5] K. Falk, Practical Recommender Systems, Manning, January.

[6] F. Isinkaye, Y. Folajimi y B. Ojokoh, «Recommendation systems: Principles, methods and evaluation,» *Egyptian Informatics Journal,* pp. 261-273, 2015.

[7] «Trained Models & Pipelines,» SpaCy, [En línea]. Available: https://spacy.io/models. [Último acceso: 5 Septiembre 2020].

[8] C. Aggarwal, Recommender Systems: The Textbook, Springer International Publishing, 2016.

[9] S. Bergamaschi y L. Po, «Comparing LDA and LSA Topic Models for Content-Based Movie Recommendation Systems,» de *International Conference on Web Information Systems and Technologies*, 2015.

[10] E. F. Loza Aguirre y A. F. Buitrago Hurtado, «Qualitative assessment of user acceptance within Action Design Research and Action Research: two case studies,» *LATIN AMERICAN JOURNAL OF COMPUTING–LAJC,* vol. I, 2014.

[11] H. Chen, M. Chau y D. Zeng, «CI Spider: a tool for competitive intelligence on the Web,» 2002.

[12] S. Blanco y N. Lesca, «From weak signals to anticipative information : learning from the implementation,» de *Proceedings of the International Conference*, Palermo, 2003.

[13] A. Casagrande, E. F. Loza Aguirre y L. Vuillon, «Improving Strategic Scanning Information Analysis: An Alternative Measure for Information Proximity Evaluation,» de *2015 International Conference on Enterprise Systems (ES)*, Basel, 2015.

[14] P. Hemp, «Death by Information Overload,» *Harvard Business Review,* September 2009.

[15] G. Miller, «The magical number seven, plus or minus two: Some limits on our capacity for processing information.,» *Psychological Review,* p. 343–352, 1956.

[16] D. Nations, «What Is a Web Application?,» Lifewire, 25 June 2020. [En línea].

[17] F. Copes, «What is a Single Page Application?,» Flaviocopes, 11 November 2018. [En línea]. Available: https://flaviocopes.com/single-page-application/.

[18] R. Greenlaw y E. Hepp, «Internet, Overview,» *Encyclopedia of Information Systems,* pp. 667-681, 2003.

[19] A. Groom, «What Is a Single-Page Application?,» Dzone, 18 July 2018. [En línea]. Available: https://dzone.com/articles/what-is-a-single-page-application.

[20] «ReactJs,» ReactJs, [En línea]. Available: https://reactjs.org.

[21] «What is an API? (Application Programming Interface),» MuleSoft, [En línea]. Available: https://www.mulesoft.com/resources/api/what-is-an-api. [Último acceso: 18 September 2020].

[22] «Basic Tutorial - Introduction,» How To GraphQL, [En línea]. Available: https://www.howtographql.com/basics/0-introduction/. [Último acceso: 18 September 2020].

[23] «Core Concepts,» How To GraphQL, [En línea]. Available: https://www.howtographql.com/basics/2-core-concepts/. [Último acceso: 28 September 2020].

[24] «The Django Project,» The Django Project, [En línea]. Available: https://www.djangoproject.com/. [Último acceso: 5 October 2020].

[25] «Graphene Python,» Graphene Python, [En línea]. Available: https://graphene-python.org/. [Último acceso: 10 October 2020].

[26] R. Pressman, Ingeniería de Software: Un Enfoque Práctico, Mexico: McGraw-Hill, 2010.

[27] I. Sommerville, Software Engineering, Pearson, 2016.

[28] S. Ambler, «User Stories: An Agile Introduction,» Agile Modeling, [En línea]. Available: http://www.agilemodeling.com/artifacts/userStory.htm. [Último acceso: 20 October 2020].

[29] D. Radigan, «Kanban: How the kanban methodology applies to software development,» Atlassian, [En línea]. Available: https://www.atlassian.com/agile/kanban. [Último acceso: 20 October 2020].

[30] M. Rehkopf, «What is Continuous Integration?,» Atlassian, [En línea]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration. [Último acceso: 30 October 2020].

[31] «Spanish news articles from the top 10,000 (based on the ranking provided by Alexa) news sites,» Webhose, October 2016. [En línea]. Available: https://Webhose.io/free-datasets/spanish-news-articles/. [Último acceso: 2 November 2020].

[32] A. Thompson, «All the news: 143,000 articles from 15 American publications,» Kaggle, [En línea]. Available: https://www.kaggle.com/snapcrack/all-the-news. [Último acceso: 10 November 2020].

[33] M. Honnibal, «Embed, encode, attend, predict: The new deep learning formula for state-of-the-art NLP models,» Explosion AI, 16 November 2016. [En línea]. Available: explosion.ai/blog/deep-learning-formula-nlp. [Último acceso: 15 November 2020].

[34] «SpaCy 101: Everything you need to know,» Spacy, [En línea]. Available: https://spacy.io/usage/spacy-101.

[35] «1.3 Getting Started - What is Git?,» Git, [En línea]. Available: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F. [Último acceso: 3 December 2020].

[36] «What Is Heroku,» Heroku, [En línea]. Available: https://www.heroku.com/about. [Último acceso: 15 December 2020].

[37] «Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE,» Guru99, [En línea]. Available: https://www.guru99.com/unit-testing-guide.html. [Último acceso: 8 December 2020].

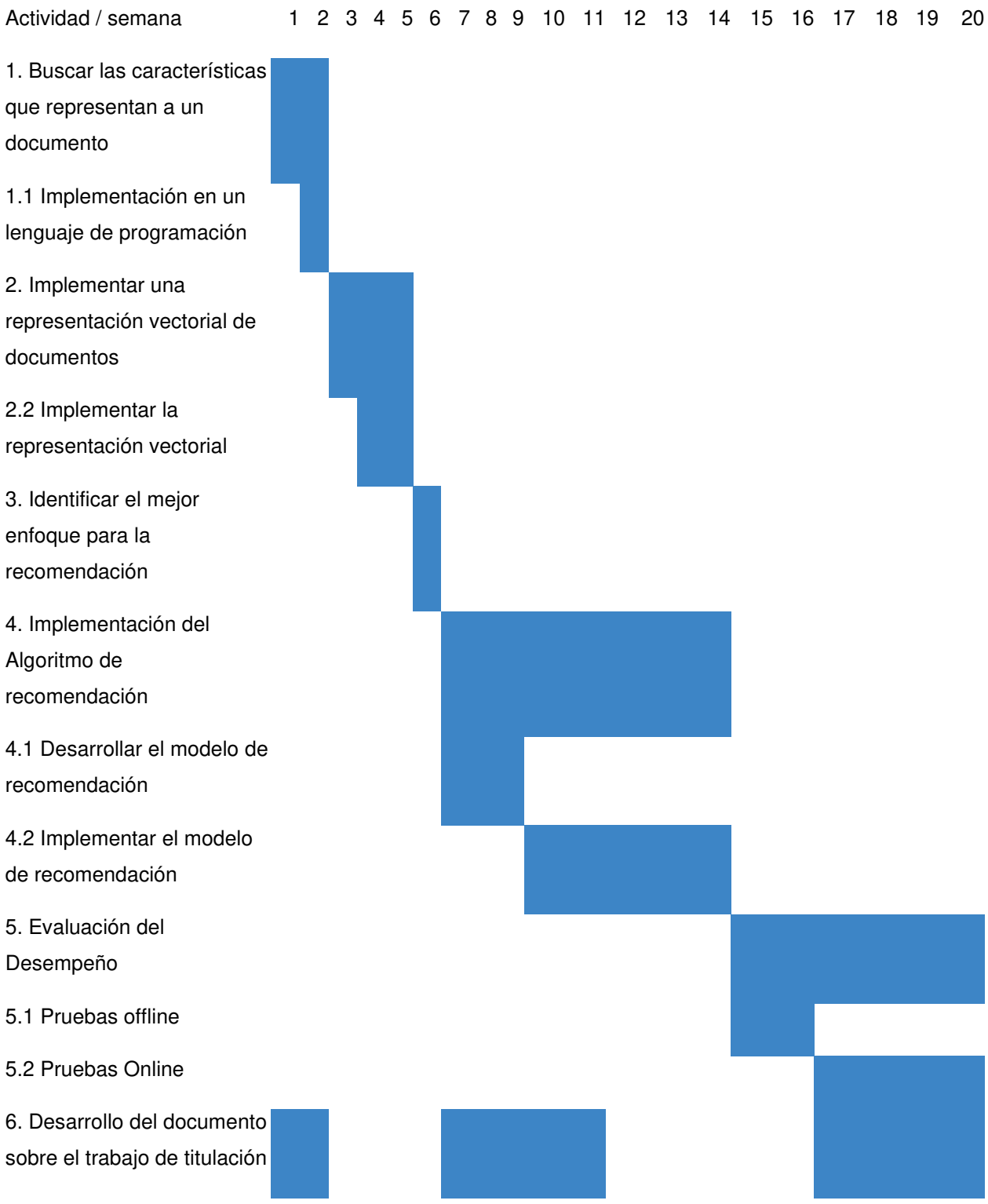
[38] «Running Tests,» Create React App, [En línea]. Available: https://create-react-app.dev/docs/running-tests/. [Último acceso: 11 December 2020].

[39] «Test Coverage in Software Testing,» Guru99, [En línea]. Available: https://www.guru99.com/test-coverage-in-software-testing.html. [Último acceso: 15 December 2020].

[40] J. C. Almenara, J. B. Osuna y . M. d. C. L. Cejudo, «Technology acceptance model & realidad aumentada: estudio en desarrollo,» *Revista Lasallista de Investigación,* vol. II, 2016.

7. ANEXOS

7.1 Cronógrama de Trabajo



7.2 Tipos

```
type User{
        username: String!
        pasword: String!
        rol: String
        votes: [Vote]
        recommendations [Recommendation]

}
type Vote {
        user: User!
        article: Article!
        liked: Boolean!
        createAt: Date!
}

Type Article {
        title: String!
        summary: String!
        lang: String!
        category: String
        dateUploaded: Date!
        textVector: String!
        similarArticles: [SimilarArticleType]
        vote: [VoteType]
}
type Recommendation {
        score: Float!
        user: UserType
        article: ArticleType
```

```
        recommended: Boolean!
        }
type SimilarArticle {
        principalArticle: ArticleType!
        relatedArticle: ArticleType!
        score: Float!

}
```

7.3 Queries

1. a**rticles**
   - **1.1. Descripción:** Permite obtener una lista de artículos basandose en una búsqueda y/o mediante paginación.
   - **1.2. Tipo:** ArticleType
   - **1.3. Arguments:**
     - **1.3.1. search: String**
     - **1.3.2. first: Int**
     - **1.3.3. skip: Int**
     - 1.3.4. Los argumentos first y skip nos sirven para la paginación en el cliente. Los tres argumentos son opcionales
   - **1.4. Ejemplo:**

```
query {
articles {
id
title
summary
lang
textVector
similarArticles {
        relatedArticle {
        id
        title
        }
}
}
}
```

2. **article**
   - **2.1. Descripción:** Buscar un artículo basado en el Id.

2.2. Type: ArticleType

2.3. Argumentos:

2.3.1. articleId: Int!

2.4. Ejemplo

```
query {
article(articleId:50) {
id
title
summary
lang
textVector
similarArticles {
        relatedArticle {
        id
        title
        }
}
}
}
```

3. **recentArticles**

3.1. Descripción: Obtiene una lista de los 10 artículos más recientes.

3.2. Type: ArticleTye

3.3. Argumentos: n/a

3.4. Ejemplo

```
query {
recentArticles {
id
title
summary
}
}
```

4. **mostVoted**

4.1. Descripción: Obtiene los 10 artículos con mas votos/likes.

4.2. Type: ArticleType

4.3. Argumentos: n/a

4.4. Ejemplo:

```
query {
mostVoted {
id
title
summary
}
}
```

5. **users**

   5.1. Descripción: Retorna una lista de los usuarios registrados en la aplicación.

   5.2. Type: UserType

   5.3. Argumentos: n/a

   5.4. Ejemplo

```
query {
users {
email
username
}
}
```

6. **user**

   6.1. Descripción: Retorna un usuario dado su Id

   6.2. Type: UserType

   6.3. Argumentos:

      6.3.1. Id: Int!

   6.4. Ejemplo

```
query {
user(id: 5) {
email
username
}
}
```

7. **votes**

   7.1. **Descripción:** Retorna una lista de todos los likes en conjunto con el artículo y el usuario que ha creado el like.

   7.2. **Type:** VoteType

   7.3. **Argumentos:** n/a

      7.3.1. **Ejemplo**

```
query {
votes {
liked
article {
      title
}
user {
      username
}
}
}
```

8. **vote**

   8.1. **Descripción:** Retorna si el usuario actual ha degustado o no de un artículo dado su id. Este query funciona si es que el usuario se encuentra logueado. Necesita que la cabecera authorization tenga el valor del token.

   8.2. **Type:** VoteType

   8.3. **Argumentos:**

      8.3.1. articleId: Int!

8.4. Ejemplo

```
query {
vote(articleId:50) {
user {
      username
}
article {
      title
}
}
}
```

9. similarArticles

9.1. Descripción: Dado el id de un artículo retorna una lista de artículos relacionados basado en el coseno de la similitud.

9.2. Type: SimilarArticlesType

9.3. Argumentos:

9.3.1. articleId: Int!

9.4. Ejemplo:

```
query {
similarArticles(articleId:50) {
relatedArticle {
      title
}
score
}
}
```

10. recommendations

10.1. Descripción: Obtiene una lista de lo 10 artículos que pueden ser de interés para el usuario que se encuentra actualmente logueado. . Necesita que la cabecera authorization tenga el valor del token.

10.2. Type: RecommendationType

10.3. Argumentos: n/a

10.4. Ejemplo:

```
query {
recommendations {
article {
      title
      summary
}
}
```

```
}
```

7.4 Mutations

1. **addArticle**
   - **1.1. Descripción:** Permite añadir un nuevo artículo a la base.
   - **1.2. Argumentos:**
     - **1.2.1. title:** String!
     - **1.2.2. summary:** String!
     - **1.2.3. lang:** String
     - **1.2.4. textVector:** String
     - **1.2.5. category:** Char
       - **1.2.5.1. dateUploaded:** Date
   - **1.3. Ejemplo**

```
mutation {
    addArticle(
        title: "title",
        summary:"summary or full content",
        category: "news",
        lang: "es"
){
  title
  sumary
}
}
```

2. **createUser**
   - **2.1. Descripción:** Mutación para registrar un nuevo usuario en el sistema.
   - **2.2. Argumentos**
     - **2.2.1. email:** String!
     - **2.2.2. password:** String!
     - **2.2.3. username:** String!

   - **2.3. Ejemplo**

```
mutation {
```

```
createUser(
email:  "mail@mail.com",
username:  "alex",
password: "alex123"
){
user {
    username
    email
}
}
}
```

3. **tokenAuth**
   - **3.1. Descripción:** Loguea el usuario al sistema y retorna un token para autorizar el acceso a recursos no disponibles para usuarios tipo visitante.
   - **3.2. Argumentos:**
     - **3.2.1. Username:** String!
     - **3.2.2. Password:** String!
   - **3.3. Ejemplo**

```
mutation {
tokenAuth(username: "alex", password: "alex123"){
token
}
}
```

4. **createVote**
   - **4.1. Descripción:** Crea un voto de un usuario a un artículo, si el voto ya existe se cambiá el valor del campo liked. Necesita que la cabecera authorization tenga el valor del token.
   - **4.2. Argumentos:**
     - **4.2.1. articleId:** Int!

4.3. Ejemplo

```
mutation {
createVote(articleId:  50) {
liked
}
}
```