

# Desarrollo de un sistema de recuperación de información

Alejandro Rodriguez Serrano

Universidad Autónoma de Puebla  
Posgrado en Base de Datos y Recuperación de Información  
Sistemas de recomendación

## 1 Introduction

Los Sistemas de Recuperación de Información (SRI) son herramientas esenciales en la era digital, diseñadas para localizar información específica dentro de bases de datos, colecciones de documentos o en la vasta red de Internet. Según Manning et al. (2008), estos sofisticados software tienen como objetivo principal ayudar a los usuarios a encontrar de manera eficiente y precisa información relevante y pertinente, basada en los términos de búsqueda ingresados [1].

Los SRI emplean algoritmos y técnicas avanzadas para rastrear y organizar la amplia gama de información disponible, ya sea en forma de texto, imágenes, vídeos u otros tipos de contenido. Según Baeza-Yates y Ribeiro-Neto (2011), estos algoritmos analizan criterios clave, como la coincidencia de palabras clave, la relevancia del contenido y la estructura del sitio web, entre otros, para determinar qué resultados son más pertinentes a una consulta específica [2].

Para lograr una búsqueda efectiva, los SRI también utilizan la técnica de las "Palabras Cerradas" o "stop words", que son palabras comunes y frecuentes en un idioma determinado, pero que generalmente se consideran irrelevantes para el análisis de texto. Según Zobel y Moffat (2006), estas palabras, que son ampliamente utilizadas en el lenguaje cotidiano, no aportan un valor semántico significativo a la comprensión del contenido de un texto, y por lo tanto, se excluyen en el proceso de búsqueda [3].

Además, los SRI se basan en una estructura de datos fundamental conocida como "Índice Posicional" o "posting list". Según Croft et al. (2010), esta estructura se utiliza para almacenar información sobre los términos de búsqueda y los documentos en los que aparecen. Los índices invertidos, que aprovechan los posting lists, son ampliamente utilizados en la búsqueda eficiente de información en grandes colecciones de documentos. Según Salton y McGill (1986), una posting list asocia un término de búsqueda con una lista de documentos en los que ese término aparece, permitiendo una recuperación precisa y rápida de información [4].

Otra medida ampliamente utilizada en los SRI es la "Similitud del Coseno". Según Manning et al. (2008), esta medida se utiliza para determinar la similitud entre dos vectores en un espacio vectorial y es especialmente relevante en el procesamiento del lenguaje natural y la recuperación de información. La similitud del

coseno es una herramienta eficiente y efectiva para calcular la similitud y encontrar la relevancia entre elementos en diversas aplicaciones, como la búsqueda de información, la agrupación de documentos, la recuperación de información y la recomendación de contenido relacionado [1].

En el desarrollo de los SRI, el lenguaje de programación Python se ha establecido como una opción destacada. Según Manning et al. (2008), Python es un lenguaje multipropósito, utilizado en una amplia gama de aplicaciones, como desarrollo web, análisis de datos, inteligencia artificial, aprendizaje automático, scripting y automatización, entre otros. La versatilidad y portabilidad de Python, compatible con múltiples plataformas como Windows, macOS y Linux, junto con su gran comunidad de desarrolladores y extensa documentación, facilitan el aprendizaje, la implementación y la resolución de problemas [1].

La amplia variedad de bibliotecas estándar y de terceros disponibles en Python también juega un papel crucial en los SRI. Según Baeza-Yates y Ribeiro-Neto (2011), estas bibliotecas proporcionan funcionalidades adicionales y simplifican el desarrollo de aplicaciones, brindando a los desarrolladores herramientas poderosas para construir sistemas de recuperación de información eficientes y precisos [2].

En la implementación de un sistema de recuperación de información, se puede utilizar Laravel, un popular framework de desarrollo web en PHP, como REST API para generar la interfaz de usuario de un sistema interno basado en web. Laravel proporciona un conjunto de herramientas y funcionalidades para desarrollar aplicaciones web de manera eficiente y escalable. Con Laravel, es posible crear una API RESTful que permita la comunicación entre el frontend y el backend de la aplicación, facilitando la interacción y la recuperación de información de manera segura y estructurada.

La combinación de un SRI con Laravel como REST API ofrece una solución completa para construir un sistema de recuperación de información robusto y accesible a través de una interfaz de usuario

## **2 Desarrollo**

### **2.1 Planteamiento del problema**

Se requiere un sistema que cuente con una interfaz de usuario implementada en un entorno web. Esta interfaz debe permitir a los usuarios ingresar una frase o consulta de búsqueda. Al enviar la consulta, el sistema deberá realizar un análisis y comparación exhaustiva en la base de datos de documentos disponibles, identificando aquellos que presenten la mayor coincidencia con la consulta ingresada.

Este sistema busca proporcionar una solución eficiente y precisa para la recuperación de información relevante. Al ingresar una frase, los algoritmos y técnicas avanzadas del sistema analizarán y compararán la consulta con los documentos almacenados, utilizando criterios como la coincidencia de palabras clave y la relevancia del contenido. El objetivo es identificar los documentos que presenten una mayor similitud y relevancia con la consulta del usuario.

## 2.2 Metodología

El desarrollo del presente trabajo se basa en una metodología sistemática, que se presenta en la figura 1. Esta metodología consta de varios pasos claramente definidos para lograr los objetivos establecidos. A continuación, se detallan cada uno de estos pasos en un lenguaje más formal:

1. **Preprocesamiento del texto en los documentos:** En esta etapa inicial, se realiza un proceso de limpieza del texto de los documentos. Se eliminan símbolos y caracteres que no son relevantes para la realización de las búsquedas. El resultado de este paso es un texto limpio y con una estructura uniforme para cada documento.
2. **Generación del posting list:** En esta fase, se genera el posting list, que es una estructura de datos fundamental para el sistema. Es necesario definir la estructura en la que se almacenará la información. El posting list relaciona los términos de búsqueda con la lista de documentos en los que aparecen.
3. **Implementación de algoritmos de búsqueda en Python:** En esta etapa, se implementan los algoritmos necesarios en Python para llevar a cabo la búsqueda por frase, así como las operaciones lógicas de búsqueda AND y OR. Estos algoritmos serán fundamentales para encontrar los documentos relevantes en función de las consultas realizadas por los usuarios.
4. **Implementación de la interfaz y el módulo de carga de posting list:** En esta última etapa, se lleva a cabo la implementación de la interfaz de usuario, que permitirá a los usuarios ingresar sus consultas de búsqueda. Además, se desarrollará un módulo para cargar y gestionar los posting list generados anteriormente. Esto facilitará la búsqueda eficiente y precisa de información.

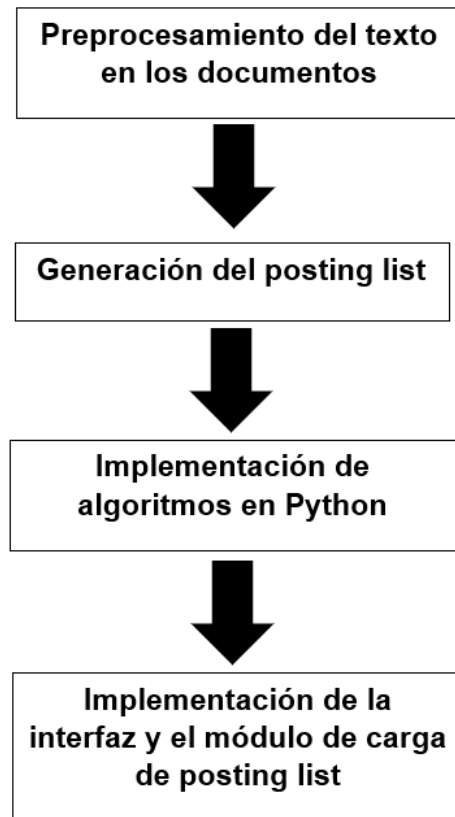


Fig. 1: Metodología

Siguiendo esta metodología, se logro los objetivos del trabajo de manera organizada y eficiente. Cada paso se lleva a cabo de manera secuencial, asegurando la calidad y funcionalidad del sistema de recuperación de información desarrollado.

### 3 Implementación

#### 3.1 Preprocesamiento del texto en los documentos

Para la implementación de la etapa de preprocesamiento del texto en los documentos, se han desarrollado dos clases en Python: "ReadFile" y "GeneratePostIn-List".

La clase "ReadFile" se encarga de buscar los documentos en una carpeta específica y los indexa de forma individual. Posteriormente, extrae el contenido de cada documento y lo prepara para el procesamiento adicional.

La clase "ReadFile" también se encarga de realizar la limpieza del texto extraído. Esta etapa implica eliminar símbolos y caracteres innecesarios, así como normalizar el texto para asegurar que tenga una estructura consistente y uniforme. Al finalizar el proceso de preprocesamiento, el texto limpio y preparado se almacena en una estructura adecuada para su posterior análisis y búsqueda, a continuación el código de la clase.

```
class ReadFile:

    def __init__(self, path='./documents', from_text_special=None):
        self.buff_files = {}
        self.path = path
        self.text_in_files = {}
        self.from_text_special = from_text_special
        if from_text_special == 'wines':
            self.load_corpus()
            print('información de vinos cargada...')
            return

        self.init_read_files()
        self.process_text_in_file()
        if from_text_special:
            self._generate_post_in_list = GeneratePostInList()

    def get_names_file_in_dir(self):
        return glob.glob(self.path + "/*.*")

    def init_read_files(self):
        for file in self.get_names_file_in_dir():
            ext = file.split('.')
            self.buff_files[file] = {'ext': ext[len(ext) - 1], 'process': False}

    def check_files_and_update(self):
        for file in self.get_names_file_in_dir():
            if len(self.buff_files[file]) > 0:
                continue
            else:
                ext = file.split('.')
                self.buff_files[file] = {'ext': ext[len(ext) - 1], 'process': False}

    def check_file_process(self, files):
        for file in files:
            if len(self.buff_files[file]) > 0:
                self.buff_files[file]['process'] = True

    def process_text_in_file(self):
        for file in self.buff_files:
            print('process file --> ', file, ' ....\n')
            if self.buff_files[file]['ext'] == 'pdf' and not self.buff_files[file]['process']:
                with fitz.open(file) as doc:
                    text = []
                    for page in doc:
                        text.append(str.lower(page.get_text()))
                    self.text_in_files[file] = " ".join(text)
            if self.buff_files[file]['ext'] == 'txt' and self.from_text_special == 'wikipedia' and not \
                self.buff_files[file]['process']:
                self.process_text_txt_wikipedia(file)
                self._generate_post_in_list.set_test_in_files(self.text_in_files)
                self._generate_post_in_list.generate()
                self.text_in_files = {}

    def get_text_in_file(self):
        return self.text_in_files

    def get_buff_files(self):
        return self.buff_files
```

```

def process_text_txt_wikipedia(self, file):
    with open(file, encoding='utf-8') as doc:
        lines = doc.readlines()
        content = ''
        title = ''
        for line in lines:
            if line.find('|*+|') > 0:
                textLine = line.split('|*+|')
                title = textLine[0]
                content = ' '.join(textLine[1])
                self.text_in_files[file] = {}
            elif line.find('*+|*+') > 0:
                self.text_in_files[file][title] = content
                print('title --> process ', title, '\n')
                content = ''
                title = ''
            else:
                content = ' '.join(lines)
        doc.close()
        del title, content, lines

def get_corpus_post_in_list(self):
    self._generate_post_in_list.print_post_in_list_in_file()
    return self._generate_post_in_list.get_post_in_list()

def get_text_from_file(self, file, init_pos=0, dim=400):
    if file in self.text_in_files:
        return self.text_in_files[file][init_pos:init_pos + dim]
    return None

def load_corpus(self):
    try:
        with open('./documents/vinos/corpus/wiki.txt', 'r', encoding='utf-8') as file:
            print('cargando información del corpus')
            for line, linea in enumerate(file):
                document = linea.split('|*+|')
                self.text_in_files[linea] = {
                    'title': document[0],
                    'text': document[1]
                }
    except UnicodeDecodeError:
        print('error al cargar corpus')

def get_text_from_file_wine(self, pos_document, init_pos):
    return self.text_in_files[int(pos_document)][['title'],
    self.text_in_files[int(pos_document)][['text']][int(init_pos[0]):
    init_pos[len(init_pos) - 1]]

```

## 3.2 Generación del posting list

Por otro lado, se ha desarrollado la clase "GeneratePostInList" encargada de generar el posting list con el objetivo de mejorar la velocidad de las búsquedas realizadas por los usuarios. Esta clase recibe como parámetro el texto extraído previamente por la clase "ReadFile".

El proceso de generación del posting list consiste en analizar el texto proporcionado y extraer los términos relevantes presentes en cada documento. Estos términos se relacionan con la lista de documentos en los que aparecen, creando así una estructura que permite una búsqueda más rápida y eficiente.

Al utilizar la clase "GeneratePostInList", se optimiza la velocidad de las consultas realizadas por los usuarios, ya que se reduce significativamente el tiempo

de búsqueda al acceder directamente a los documentos que contienen los términos deseados.

La implementación de esta clase complementa el proceso de preprocesamiento del texto y contribuye a mejorar la eficiencia del sistema de recuperación de información, garantizando respuestas más rápidas y precisas para los usuarios, a continuación el código de la clase.

```
class GeneratePostInList:

    def __init__(self, text_in_files=None):
        self.post_in_list = {}
        if text_in_files == 'wines':
            print('generando post in list para vinos')
            self.process_text_txt_vinos()

        elif text_in_files:
            self.text_in_files = text_in_files
            self.generate()
            self.print_post_in_list_in_file()

    def set_text_in_files(self, text_in_files):
        self.text_in_files = text_in_files

    def generate(self):
        for file in self.text_in_files:
            text = self.clear_text(self.text_in_files[file])
            words = text.split(' ')
            for pos, word in enumerate(words):
                if word not in self.post_in_list:
                    self.post_in_list[word] = {}
                    self.post_in_list[word][file] = [pos]
                else:
                    if file in self.post_in_list[word]:
                        self.post_in_list[word][file].append(pos)
                    else:
                        self.post_in_list[word][file] = [pos]

    def print_post_in_list_in_file(self):
        with open("./outputs/post_in_list.txt", 'w', encoding='utf-8') as f:
            for key, item in self.post_in_list.items():
                f.write('%s:%s\n' % (key, item))

    def clear_text(self, text):
        text = re.sub(r"[^\w\d?,:./<>=;!-_()#@{}[\]\n]", '', text)
        text = " ".join(text.split())
        return text

    def process_text_txt_vinos(self):
        with open('documents/vinos/postingList1.txt', encoding='utf-8') as f:
            self.post_in_list = json.load(f)

    def get_post_in_list(self):
        return self.post_in_list

    def sorter_post_in_list(self):
        pass
```

### 3.3 Implementación de algoritmos de búsqueda en Python

Además, se ha desarrollado una tercera clase llamada "Operators" para implementar los algoritmos de búsqueda. En esta clase se encuentra la lógica necesaria para realizar búsquedas utilizando los operadores lógicos AND, OR y NEAR. Como parámetro, la clase "Operators" recibe el posting list generado por la clase "GeneratePostInList".

La clase "Operators" permite llevar a cabo búsquedas más complejas y precisas, utilizando los operadores lógicos mencionados. El algoritmo de búsqueda AND devuelve aquellos documentos que contienen todos los términos de búsqueda ingresados por el usuario. Por otro lado, el algoritmo de búsqueda OR devuelve aquellos documentos que contienen al menos uno de los términos de búsqueda. Finalmente, el operador NEAR encuentra documentos donde los términos de búsqueda se encuentran cercanos entre sí en el texto.

La implementación de la clase "Operators" complementa el funcionamiento del sistema de recuperación de información, permitiendo a los usuarios realizar búsquedas más sofisticadas y obtener resultados más precisos y relevantes. Esta clase proporciona una funcionalidad esencial para el procesamiento de consultas y la extracción de información relevante de los documentos almacenados.

A continuación el código de la clase:

```
class Operators:

    def __init__(self):
        print('Near operator init')

    def near_operator(self, post_in_list_word1, post_in_list_word2, range=3):
        answer = {}
        for element in zip(post_in_list_word1, post_in_list_word2):
            if element[0] == element[1]:
                for pos_a, pos_b in zip(post_in_list_word1[element[0]], post_in_list_word2[element[1]]):
                    if range:
                        if (pos_a - pos_b) <= range:
                            if element[0] not in answer:
                                answer[element[0]] = [pos_a]
                                answer[element[1]] = [pos_b]
                            else:
                                answer[element[0]].append(pos_a)
                                answer[element[1]].append(pos_b)
                        else:
                            if element[0] not in answer:
                                answer[element[0]] = [pos_a]
                                answer[element[1]] = [pos_b]
                            else:
                                answer[element[0]].append(pos_a)
                                answer[element[1]].append(pos_b)
                    else:
                        if element[0] not in answer:
                            answer[element[0]] = [pos_a]
                            answer[element[1]] = [pos_b]
                        else:
                            answer[element[0]].append(pos_a)
                            answer[element[1]].append(pos_b)
                return answer

    def phrase_near_operator(self, phrase, post_in_list, operator_and=False):
        words = phrase.split(' ')
        pos_word = 0
        result = None
        print('>---->> palabras ', words)
        if len(words) < 1:
            print('error, no estás ingresando ninguna palabra \n')
            return
        if len(words) == 1:
            print(words[0], post_in_list[words[0]])
            if words[0] in post_in_list:
                return post_in_list[words[0]][1]
            else:
                return

        result = []
        next_post = 0
        while next_post < len(words) - 1:
            print('-->>> while', next_post)
            # Verificar si existe la palabra sino buscar por
            if words[pos_word] in post_in_list:
                print('buscando --> ', words[pos_word])
```



```

        if pos_word == 0:
            next_post = pos_word + 1
            while next_post < len(words):
                print('buscando la siguiente coincidencia --> ', words[next_post])
                if words[next_post] in post_in_list:
                    print('coincidencia encontrada se busca con el operador y se rompe el ciclo')

                    if not operator_and:
                        result = self.near_operator(post_in_list[words[pos_word]][1],
                                                    post_in_list[words[next_post]][1])
                    else:
                        result = self.near_operator(post_in_list[words[pos_word]][1],
                                                    post_in_list[words[next_post]][1], 0)

                    break
                else:
                    next_post += 1

            if next_post >= (len(words) - 1):
                result = post_in_list[words[pos_word]][1]

            print('siguientes posiciones --> ', pos_word, next_post, len(words))
            pos_word = next_post

        else:
            print('buscando sobre el resultado --> ')
            print('resultado anterior --> ', result)

            if not operator_and:
                result = self.near_operator(result, post_in_list[words[pos_word]][1])
            else:
                result = self.near_operator(result, post_in_list[words[pos_word]][1], 0)

            pos_word = pos_word + 1

    else:
        next_post += 1
        continue
    return result

def distanceLevenshtein(self, str1, str2):
    d = dict()
    for i in range(len(str1) + 1):
        d[i] = dict()
        d[i][0] = i
    for i in range(len(str2) + 1):
        d[0][i] = i
    for i in range(1, len(str1) + 1):
        for j in range(1, len(str2) + 1):
            d[i][j] = min(d[i][j - 1] + 1, d[i - 1][j] + 1, d[i - 1][j - 1] + (not str1[i - 1] == str2[j - 1]))
    return d[len(str1)][len(str2)]

```

### 3.4 Implementación de la interfaz y el módulo de carga de posting list

En el desarrollo del módulo de carga del posting list, se ha creado la clase "Main" que desempeña un papel fundamental al unir y coordinar todas las clases anteriores. Esta clase es responsable de crear instancias de las clases mencionadas anteriormente y mantener la información relevante en memoria durante la ejecución del sistema.

Al ser ejecutada, la clase "Main" establece una comunicación bidireccional entre el sistema desarrollado en Python y la plataforma Laravel utilizada para la interfaz de usuario en el entorno web. Esta interacción se realiza a través de

archivos en disco duro, en los cuales se intercambian consultas provenientes del frontend y respuestas generadas por el script.

La clase "Main" se encarga de procesar las consultas recibidas desde Laravel, utilizando los objetos previamente creados de las clases "ReadFile", "GeneratePostInList" y "Operators". Estos objetos permiten realizar el procesamiento de las consultas y generar respuestas adecuadas, las cuales son escritas en un archivo específico. Por su parte, Laravel se mantiene escuchando ese archivo para detectar cuando se han obtenido los resultados y mostrarlos en la interfaz de usuario.

La implementación de la clase "Main" garantiza la sincronización entre el sistema desarrollado en Python y la plataforma Laravel, permitiendo una interacción fluida y eficiente. Además, esta clase centraliza el control de las clases y objetos relacionados, asegurando un manejo adecuado de la información en memoria y facilitando la integración entre las distintas partes del sistema.

A continuación el código de la clase main:

```
class MainProgram:

    def __init__(self):
        # self.files = ReadFile(path='./outputWikipedia', from_text_special='wikipedia')
        # self.files = ReadFile(path='./documents', from_text_special='wikipedia')
        self.files = ReadFile(path='./documents/vinos', from_text_special='wines')
        self.corpus = GeneratePostInList('wines')
        # self.corpus = GeneratePostInList(self.files.get_text_in_file())
        # self.corpus = self.files.get_corpus_post_in_list()
        print('se genero la información')
        self.operator = Operators()
        self.core_data = LoadDataCore()

    def valid_search(self, search):
        error = False
        frase = []
        for word in search:
            if word not in self.core_data.get_dictionary():
                error = True
                list_t = [word[i:i + 2] for i in range(len(word) - (2 - 1))]
                list_proposals = []
                proposals = set()
                for grama in list_t:
                    proposals = proposals.union(self.core_data.get_dictionary_gramas(grama))
                    list_proposals.append(self.core_data.get_dictionary_gramas(grama))

                dMin = 999
                pMin = ""
                for prop in proposals:
                    aux = self.operator.distanceLevenshtein(word, prop)
                    if aux < dMin:
                        dMin = aux
                        pMin = prop
                    print(dMin)
                frase.append(pMin)
            else:
                frase.append(word)
        return error, " ".join(frase)

    def main(self):
        while True:
            lines = []
            with open("../storage/app/input.txt", 'r', encoding='utf-8') as f:
                try:
```

```

        lines = f.readlines()
    except PermissionError:
        print('writing another script')

print(lines, len(lines))
if len(lines) > 0:
    outputs = []
    error = False
    for line in lines:
        if len(line) > 0:
            line = str.lower(line)
            line = line.split('|')
            operator_use = line[1]
            phrase = line[0]
            response = None
            print(phrase, operator_use)
            error, frase = self.valid_search(phrase.split(" "))
            if error:
                outputs.append({'suggestions': '¿quizo decir: ' + frase + ' ?'})
            else:
                print('-->', phrase)
                if operator_use == 'near':
                    print('-->--> uso operador near')
                    response = self.operator.phrase_near_operator(phrase,
                                                                self.corpus.get_post_in_list())
                if operator_use == 'and':
                    response = self.operator.phrase_near_operator(phrase,
                                                                self.corpus.get_post_in_list(), True)

            if response:
                outputs.append(response)
            print('*****>', outputs, len(outputs))
if len(outputs) > 0:
    if error:
        with open("../storage/app/output.txt", 'w', encoding='utf-8') as f:
            f.write(json.dumps(outputs))
    else:
        outputs = dict(sorted(outputs[0].items(), key=lambda x: len(x[1]), reverse=True))
        number_response = 0
        responses = {}
        for file in outputs:
            title, text = self.files.get_text_from_file_wine(pos_document=file, init_pos=outputs[file])
            responses[file] = {
                'text': text,
                'title': title
            }
            number_response += 1
            if number_response == 50:
                break

        with open("../storage/app/output.txt", 'w', encoding='utf-8') as f:
            f.write(json.dumps(responses))
    else:
        print('salí en else')
        with open("../storage/app/output.txt", 'w', encoding='utf-8') as f:
            f.write('not results')
        with open("../storage/app/input.txt", 'r+') as file:
            file.truncate(0)

start = MainProgram()
start.main()

```

Para la implementación de la interfaz de usuario, se utilizó el motor de plantillas Blade, que es el motor de plantillas predeterminado en Laravel. Además,

se siguió el patrón de diseño Modelo-Vista-Controlador (MVC) para asegurar una estructura organizada y modular del código.

Laravel, como framework, proporciona una amplia documentación oficial que detalla sus características, funcionalidades y cómo utilizar el motor de plantillas Blade. Recomendamos consultar la documentación oficial de Laravel en [5] para obtener más información sobre el framework.

A continuación, se muestra un ejemplo de código que representa la lógica de la petición en el contexto de la implementación:

```
public function executeProcess(Request $request)
{
    if ($request->get('optionSearch') == 'corpus') {
        $search = trim($request->get('search'));
        if (
            ($search[0] == '\\' || $search[0] == '"') &&
            ($search[strlen($search) - 1] == '\\' || $search[strlen($search) - 1] == '"')
        ) {
            $operator = 'near';
            $search = str_replace('\\', '', $search);
            $search = str_replace('"', '', $search);
        } else {
            $operator = 'and';
        }

        $results = $this->executeSearchCorpus($search, $operator);
    } elseif ($request->get('optionSearch') == 'recommendation') {
        $results = $this->executeSearchRecommendationSystem($request->get('search'));
    } else {
        $results = null;
    }
    return view('results')->with('results', $results);
}

public function executeSearchCorpus($search, $operator)
{
    Storage::disk('local')->put('input.txt', $search . '|' . $operator);
    $response = null;
    while (empty($response)) {
        $response = Storage::disk('local')->get('output.txt');
    }
    if ($response !== 'not results')
        $response = json_decode($response);
    else
        $response = null;

    Storage::disk('local')->put('output.txt', '');
    // print_r($response);
    return $response;
}
```

En la Figura ?? se muestra un ejemplo de cómo luce la interfaz de usuario implementada. Pueden observarse los elementos visuales diseñados utilizando el motor de plantillas Blade y siguiendo las directrices del patrón de diseño MVC. La figura es representativa de la apariencia y la disposición de los elementos en la interfaz de usuario del sistema implementado.

Vino tinto

Q


#	Documento	Posiciones
	19935 Vino de hielo	<p>El vino de hielo (en alemán Eiswein, en inglés icewine o ice wine y en francés vin de glace) es un vino hecho de uva helada con una fuerte concentración en azúcar. La técnica para conseguir esta uva consiste en dejar sobremadurar la uva en la cepa, que no se cosecha hasta que se produce la primera helada. Cuando el grano se hiela, el agua se expande y rompe la cascarilla de la uva. Así, se pierde más agua y el azúcar es más concentrado. Los vinos de hielo son extraordinarios de riqueza y de persistencia aromática gracias a su concentración y a una acidez fuera de lo común. Es una técnica que se utiliza en zonas frías donde se cultiva la viña, sobre todo en Alemania, Canadá (donde se le llama, en francés, con el término vin de glace) y Francia y se hace con variedades como Gewürztraminer y Riesling, pero también con chardonnay, cabernet franc y vidal (en Canadá). Los que tienen más reputación se encuentran en Alemania, sobre todo a lo largo del Mosela, el Sarre y el Ruwer. Hay igualmente Eiswein austriacos. Son a menudo muy caros (250 euros por botella). Quebec tiene también una buena reputación en este dominio, donde también se elabora sidra de hielo. En Canadá se produce también en Columbia Británica y en Ontario, en la región de los Grandes Lagos. En España en la Valladolid, La Seca, (Rueda), de uva verdejo. Bodegas Vidal Soblechero en 2006 fue la primera en elaborarlo de forma natural la congelación de la uva fue en la cepa en la propia viña. (Ice Clavador) y en Aragón, Miedes (Denominación de Origen Calatayud), mediante uva Macabeo, o en Palencia, Torquemada, (D. O. Arlanza) a partir de uva tempranillo y también</p>
	5575 Cabernet sauvignon	<p>n es una de las uvas tintas más conocidas del mundo. Crece en casi todas las grandes zonas vitícolas, en un diverso espectro de climas, desde el valle del Okanagan (Canadá) al valle de la Beca (Líbano). La cabernet sauvignon se hizo famosa por su presencia en el vino de Burdeos, donde es mezclada a menudo con la merlot y con la cabernet franc. Desde Francia, la uva se ha extendido por Europa y por el Nuevo Mundo</p>

Fig. 2: Iterfaz

Para obtener más detalles sobre la implementación específica de la interfaz y las funcionalidades del sistema, se recomienda consultar la documentación oficial de Laravel y examinar el código fuente del proyecto.

## 4 Conclusión

En conclusión, se ha desarrollado un sistema de recuperación de información que utiliza algoritmos y técnicas avanzadas para buscar y proporcionar resultados relevantes a partir de una base de datos de documentos. El proceso de búsqueda se ha optimizado mediante la implementación de un posting list que acelera la velocidad de recuperación de información.

El sistema se ha estructurado siguiendo una metodología que abarca las etapas de preprocesamiento de los documentos, generación del posting list y la implementación de algoritmos de búsqueda basados en los operadores lógicos AND, OR y NEAR. Estos algoritmos permiten a los usuarios realizar consultas más sofisticadas y obtener resultados más precisos.

Además, se ha implementado una interfaz de usuario utilizando el motor de plantillas Blade en el framework Laravel, siguiendo el patrón de diseño Modelo-Vista-Controlador (MVC). Esto ha permitido crear una interfaz intuitiva y amigable, facilitando la interacción del usuario con el sistema.

La integración entre el sistema desarrollado en Python y la plataforma Laravel se ha logrado mediante la comunicación a través de archivos en disco duro. Esta interacción garantiza la transferencia adecuada de consultas y resultados entre ambas partes.

En resumen, el sistema de recuperación de información desarrollado ha demostrado ser eficiente en la búsqueda y recuperación de información relevante. La implementación de las clases "ReadFile", "GeneratePostInList", "Operators" y "Main" ha permitido un manejo estructurado y coherente de las diferentes etapas del sistema. La interfaz de usuario desarrollada en Laravel ha proporcionado una experiencia agradable para los usuarios, permitiendo realizar consultas de manera fácil y obtener resultados de forma rápida y precisa.

## References

1. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology behind Search. 2nd edn. Addison-Wesley (2011)
3. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys **38** (2006) Article 6
4. Croft, W.B., Metzler, D., Strohman, T.: Search Engines: Information Retrieval in Practice. Pearson Education (2010)
5. : Laravel documentation. (<https://laravel.com/docs>) Accessed on [Fecha de acceso].
6. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill (1986)