BAB III
PELAKSANAAN KERJA MAGANG

3.1. Kedudukan dan Koordinasi

Mahasiswa diberikan kesempatan untuk melakukan praktek kerja magang di instansi Bank Sentral Indonesia selama 5 bulan. Selama mahasiswa menjalani praktik kerja magang, mahasiswa diberikan posisi yang bisa disebut sebagai *website developer intern* untuk membuat proyek internal kelompok 2 pada departemen manajemen risiko. Dalam menjalani tugasnya, mahasiswa akan dipantau dan diberikan bimbingan oleh supervisor dari institusi yaitu Ibu Saadiah Ludmilla. Mahasiswa akan dilibatkan dalam proyek utama magangnya yaitu pembuatan *search engine* untuk dokumen-dokumen kelompok 2. Namun selain itu mahasiswa juga dipaparkan oleh tugas-tugas dan diskusi kelompok asesmen *major project* serta membantu tugas-tugas kelompok. *Search engine* yang akan dibuat ini nantinya akan digunakan oleh kelompok 2 untuk mendukung tugas-tugasnya yang mana salah satunya adalah mengases risiko *major project.*

Mahasiswa akan dilibatkan dalam pembuatan fitur-fitur dari *search engine* dari sisi *back-end* maupun *front-end*. *Search engine* yang akan dibuat ini sesuai dengan kebutuhan dari kelompok 2 yang mana dapat memberikan *ranking* dokumen yang paling relevan dari *query* yang diberikan oleh pengguna. Proses pencarian juga bisa digabungkan dengan filter yaitu berdasarkan satuan kerja dan tanggal dokumen tersebut dibuat sehingga hasil pencarian dapat lebih relevan. Bagaimana *search engine* ini dapat menghasilkan *result* yang relevan adalah dengan menggunakan model pencarian *vector space* yaitu model yang umum digunakan dalam pembuatan *information retrieval.*

Halaman *search engine* itu sendiri memiliki dua fitur utama dari sisi *user interface* yaitu upload dokumen dan mesin pencarian. Sedangkan dari sisi *back-end,* kita dapat mengelompokkan alur utama *search engine* ini menjadi 3 alur yaitu alur *indexing*, alur upload dokumen, dan terakhir adalah alur pencarian dan perangkingan dokumen dengan pengaplikasian awal *machine learning learning to rank.* Mahasiswa terlibat aktif dalam pembuatan *user interface,* fitur upload dokumen dan mendukung serta membantu melanjutkan pembuatan fitur *indexing* dan pencarian dokumen.

3.2. Tugas dan Uraian Kerja Magang

Kegiatan program kerja magang ini memberikan kesempatan kepada mahasiswa untuk memberikan kontribusi nyata dalam aktivitas atau pekerjaan institusi. Dalam kontribusinya, mahasiswa melakukan beberapa tugas yang diberikan oleh mentor dalam pembuatan *search engine*. Mahasiswa juga belajar mengenai bagian yang dikerjakan oleh anggota lain dalam pembuatan *search engine* ini. Secara keseluruhan, pembagian tugas antara mahasiswa cukup abstrak dan tidak menentu di masing-masing tahapan pengerjaan proyek. Walaupun begitu, mahasiswa tetap memiliki partisipasi di tiap tahap dan mencoba sebisa mungkin memahami hasil pekerjaan anggota tim lain sehingga memahami *outline* dari proses back-end yang dibuat. Berikut adalah aktivitas yang dilakukan oleh mahasiswa selama melakukan kerja magang selama lima bulan di perusahaan.

3.2.1. Tugas Kerja Magang

Pada tahap perkenalan praktek kerja nyata diawali dengan adanya pengenalan terhadap lingkungan baru serta diperkenalkannya dengan tugas-tugas yang akan dikerjakan. Selain melakukan proyek utama yaitu pembuatan *search engine,* mahasiswa juga diberikan kesempatan untuk berkecimpung di bidang lainnya salah satunya adalah mengikuti dan mendukung kegiatan kelompok dua asesmen risiko *major project*. Tugas mahasiswa disini beragam sehingga membuka kesempatan mahasiswa untuk mempelajari hal-hal baru dalam bidang asesmen risiko, ekonomi, struktur kerja perusahaan baik itu dari level eksternal atau peraturan dari departemen dengan departemen lain maupun secara internal yaitu hubungan dan kerja sama antar kelompok di departemen manajemen risiko. Mahasiswa juga mempelajari jenis-jenis *major project*, dasar-dasar dari asesmen risiko, serta pengukuran risiko *major project* untuk nantinya dirumuskan mitigasi. Mahasiswa juga dilibatkan berat ke salah satu *major project* yang ditangani oleh kelompok 2 yaitu layanan BI-FAST.

Selain menambah wawasan dan ilmu dari sisi asesmen risiko, mahasiswa juga mendapatkan pelatihan mengenai infrastruktur layanan BI-FAST, tahapan pengembangan, kekurangan dan kelebihan pengembangan layanan, hingga titik-titik ketidakstabilan dari layanan serta solusi nyata yang dilakukan untuk menyelesaikan

permasalahan tersebut. Mahasiswa diikutsertakan untuk mencari titik utama yang perlu diperhatikan sehingga bisa dikembangkan untuk didiskusikan kepada pihak pengembang yang pada konteks ini adalah departemen IT Bank Indonesia. Kegiatan ini mendorong mahasiswa untuk berpikir kritis, *attention to detail*, serta secara keseluruhan mendorong mahasiswa untuk mempelajari hal-hal yang sangat baru seperti infrastruktur dari layanan tersebut.

Diluar kegiatan mahasiswa ikut serta mendukung kegiatan kelompok dua, mahasiswa juga mengerjakan proyek utamanya yaitu pembuatan *search engine*. Mahasiswa mengawali proyek dengan mengumpulkan kebutuhan dari user dan menentukan fitur-fitur yang nantinya akan diaplikasikan ke halaman *search engine.* Dalam pembangunan *search engine* ini, mahasiswa bekerja sama dengan anggota tim dan dibantu oleh *supervisor* mahasiswa. Mahasiswa membantu serta mengerjakan tugas yang diberikan oleh *supervisor* dalam pembangunan *search engine* ini. Maka dari itu mahasiswa dapat berkontribusi dan mendukung dalam pembangunan tiga alur utama *search engine* ini yaitu alur upload dokumen, alur pengindeksan atau BSBI dan alur *searching*. Pembuatan *search engine* aktif dimulai pada bulan April ketika semua tim pengembang telah hadir dan siap untuk melanjutkan pembuatan proyek. Secara garis besar, berikut adalah uraian pekerjaan yang mahasiswa ikuti selama melakukan pembangunan *search engine* di perusahaan.

No	Pekerjaan yang dilakukan	Minggu	Bulan
1	Mempelajari tugas-tugas dari kelompok dua asesmen risiko *major project* dan beradaptasi dengan lingkungan kantor.	1 - 2	Januari
2	Mengumpulkan kebutuhan user akan *search engine* dan menyusun fitur-fitur yang akan dibangun.	1 - 2	Februari
3	Mempelajari framework django dan model yang akan digunakan yaitu vector space model dalam pembuatan proyek.	2 - 4	Februari

4	Membuat front-end untuk search engine.	3 - 8	Februari & Maret
5	Berpartisipasi dalam pembangunan alur indexing sebagai tahap awal pemrosesan dokumen *search engine*.	9 - 15	Maret & April
6	Berpartisipasi dalam pembangunan alur Upload dokumen diantaranya pre-processing data, *index writing*, dan lainnya.	15 - 17	Mei
7	Berpartisipasi dalam pembangunan alur Searching.	16 - 18	Mei

Tabel 3. 1 Timeline Pelaksanaan Tugas Kerja Magang

3.2.2. Uraian Kerja Magang

Pada pembuatan *search engine* ini digunakan model *vector space* yang mana merupakan model yang merepresentasikan segala hal yang digunakan seperti *term*, dokumen, ataupun query kedalam vector multi dimensi. Berikut adalah uraian dari proyek yang mahasiswa ikut sertakan.

1. **Pengenalan Tugas-Tugas Kelompok 2 Asesmen Risiko Major Project**

   Mahasiswa memulai praktek kerja magang di Bank Indonesia tepat pada hari senin 15 Januari tahun 2024. Kegiatan pertama yang mahasiswa jalani adalah pembauran akan lingkungan kerja baru dan tugas-tugas dari kelompok 2 Departemen Manajemen Risiko yaitu sebagai Asesmen Risiko *Major Project.* Mahasiswa diberikan bahan-bahan yang bersangkutan dengan pekerjaan kelompok 2 seperti peraturan gubernur organisasi yang membahas peraturan-peraturan dan tata kelola proses kerja departemen, unit kerja, satuan kerja, dan lainnya dalam perusahaan, program kerja kelompok 2, jenis-jenis dan pengkategorian *major project,* cara mengklasifikasi dan menilai risiko *major project,* serta teknik perhitungan tingkat risiko dari masing-masing risiko *major project.*

   Kedepannya mahasiswa juga diikutsertakan untuk mendukung kelompok dua dalam pengerjaan tugas-tugas pokoknya. Kegiatan yang dilakukan mahasiswa

seperti membantu membuat pointers pertemuan, membantu mencari titik kemungkinan terjadinya risiko di update *major project*, dan lainnya. Mahasiswa juga diikutsertakan untuk menyimak rapat dan diskusi kelompok sehingga mendapatkan pengalaman dan pemahaman mengenai manajemen risiko di Bank Indonesia.

2. Mengumpulkan Kebutuhan User dan Penentuan Fitur-Fitur Search Engine

Kegiatan selanjutnya yang mahasiswa lakukan adalah mengumpumpulkan kebutuhan dan ekspektasi akhir dari user terhadap search engine ini. Langkah ini dilakukan dengan diskusi antara pengembang dan calon user. Dari hasil diskusi, ditemukan kendala yang dihadapi user diantaranya:

- kesulitan untuk mencari dokumen yang sesuai dalam waktu yang sedikit. Sering kali kelompok 2 melakukan rapat atau diskusi dengan departemen lain secara dadakan dan membutuhkan dokumen lampau sebagai referensi.
- Dokumen-dokumen rapat tidak tersimpan di satu tempat utama. Hal ini sering mengakibatkan pencarian dokumen memakan waktu lama hingga tidak jadi dilakukan.

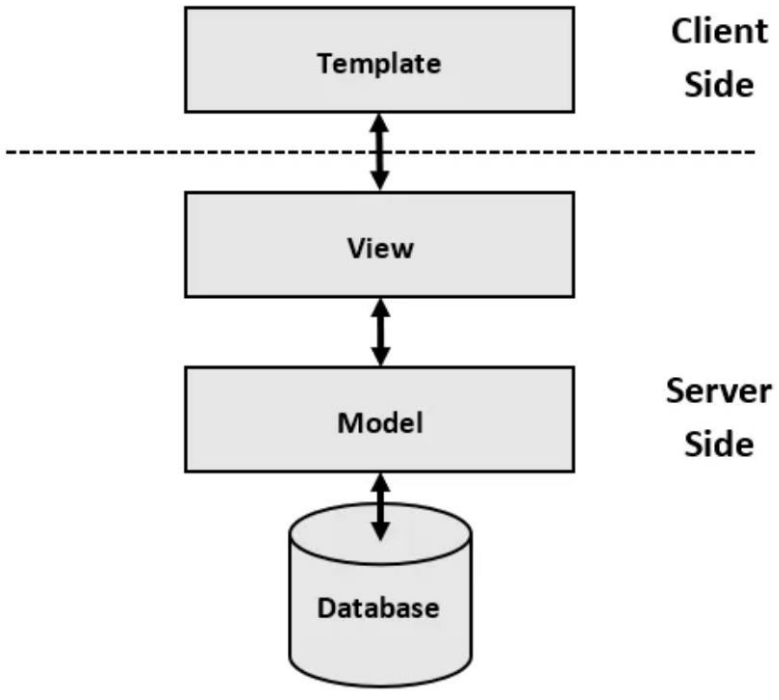
Maka dari itu ditarik hasil akhir atau harapan akhir dari user yaitu:

- Search engine bisa menampung dokumen-dokumen risalah rapat kelompok 2 dan disertai mesin pencari untuk pencarian dokumen berdasarkan bahasan yang relevan dari query.
- Tersedia fitur upload untuk mendukung keefektifan mesin pencarian. Karena dokumen-dokumen risalah rapat ini akan terus bertambah seiring berjalannya waktu, maka diperlukan juga fitur upload dokumen agar user bisa menyimpan dokumen baru di search engine ini.
- Tersedia fitur filter untuk mendukung keefektifan mesin pencarian. Ditemukan bahwa user sering kali mencari dokumen berdasarkan departemen pengadaan rapat maka dari itu fitur filter dalam pencarian dokumen dianggap dapat membantu proses pencarian dokumen. Filter tersebut adalah filter satuan kerja dan filter waktu diadakannya rapat.

Jadi ditentukan bahwa *search engine* ini akan terdiri dari dua fitur utama yaitu upload dokumen dan mesin pencarian dokumen.

3. Mempelajari Framework Django & Model Information Retrieval Vector Space

Pembuatan *search engine* ini akan berbasis website dilakukan dengan menggunakan *framework* Django versi 5.0.2 dan bahasa pemrograman Python versi 3.12.2. *Framework* Django ini adalah sebuah *framework* yang bersifat open-source yang umum digunakan dalam pengembangan aplikasi berbasis *website* dengan menggunakan bahasa pemrograman Python [9]. Sebagai sebuah *framework*, Django memiliki arsitekturnya sendiri yang disebut Model-Template-View atau MVT dan dapat digambarkan sebagai berikut.



Gambar 3. 1 Arsitektur Django Framework

Model dalam MVT memiliki fungsi yang sama dengan model yang diterapkan pada Model-View-Controller atau MVC. Pada model Django MVT juga terdapat bagian yang disebut sebagai Template. Bagian template ini memiliki fungsi yaitu menyediakan tampilan informasi yang mempermudah user melihat dan

membacanya. Dari gambar 3.1 Tampak bahwa arsitektur ini terdiri dari 4 bagian utama, yaitu dari sisi user ada Template, dan dari sisi server ada View, Model, dan Database.

Dari sisi server, Model bertanggung jawab untuk mengelola data dan logika bisnis dari aplikasi. Dengan kata lain model merepresentasikan tabel-tabel dari database django. Model bertanggung jawab dalam menangani pengambilan data dari database, penyimpanan data, dan memastikan data-data tersebut sesuai dengan ketentuan yang dibuat user seperti tipe data dan panjang maksimal data. Seperti pada proyek ini, model digunakan untuk membuat tabel dokumen yang mana menyimpan nama dokumen, dan id dokumen. Tabel dokumen ini juga memiliki ekstensi atau foreign key ke tabel Risalah Rapat yang meng-inherit tabel Dokumen. Tabel risalah rapat dibuat untuk menampung dokumen-dokumen yang memiliki jenis risalah rapat sehingga memiliki kolom tambahan yaitu tanggal rapat dan satuan kerja.

Bagian kedua dari sisi server yaitu View memiliki fungsi mengendalikan apa yang terjadi ketika pengguna mengunjungi URL tertentu. View bisa dipahami sebagai jembatan atau yang menghubungi antara model dan template ataupun antar halaman template. Ketika user mengirim permintaan, view akan mengambil data dari model yang sudah dibuat dan mengirimnya ke template. Pada proyek kali ini, view memiliki beberapa function yang berfungsi untuk menangani masing-masing request yang berada di halaman search engine yang dibuat. Diantaranya ada function untuk mengupload dokumen, function pengambilan data dari tabel untuk filter, function feedback, dan function melakukan indexing.

Terakhir adalah arsitektur Django di sisi user yaitu template. Template berfungsi untuk menyajikan data atau menampilkan front-end kepada pengguna. Template ini adalah file HTML yang bisa berisi placeholder untuk data-data dinamis yang dikelola oleh view. Template menerima data dari view dan menampilkannya di tempat yang sudah disediakan pada html yang dibuat. Contohnya pada proyek ini, template search result akan menampilkan dokumen-dokumen yang sudah diambil oleh view dari models.

Proyek berbasis web yang dibuat dengan framework Django ini memiliki konfigurasi-konfigurasi dari aplikasi pembentuk website tersebut. File-file konfigurasi tersebut memiliki fungsi masing-masing diantaranya adalah sebagai berikut:

1) __init__.py : Pada proyek Django, __init__.py ini adalah direktori kosong namun bisa berfungsi untuk memastikan modul-modul didalam direktori dapat diakses oleh framework Django.
2) aesgi.py dan wsgi.py : asgi.py atau Asynchronous Server Gateway Interface berfungsi sebagai titik masuk untuk aplikasi Django yang akan dijalankan secara asynchronous. ASGI memungkinkan aplikasi Django untuk mendukung protokol web seperti HTTP. Sedangkan wsgi.py adalah titik masuk standar untuk menjalankan aplikasi Django di server web yang mendukung WSGI atau protokol standar untuk komunikasi antara server web dan aplikasi Python dan file wsgi.py memastikan aplikasi Django dapat berjalan di server. Secara garis besar kedua file ini berfungsi sebagai file yang digunakan untuk melakukan *deployment* pada proyek yang dikerjakan dengan melibatkan *web server* yang sesuai dengan protokol ASGI atau WSGI.
3) settings.py : file settings ini berisi pengaturan dan konfigurasi utama dari proyek Django. Pengaturan ini mencakup informasi tentang database, aplikasi yang digunakan atau terinstall, middleware, template, dan berbagai pengaturan lainnya yang mengontrol alur berjalannya aplikasi Django.
4) urls.py : File ini bertanggung jawab untuk menentukan pola atau jalur URL yang digunakan dalam aplikasi Django. Dengan menggunakan URLconf(URLconfiguration), urls.py memetakan URL yang diminta user ke view yang tepat yang mana selanjutnya akan dilanjutkan atau ditampilkan pada templates yang tepat. Ini memungkinkan Django untuk menavigasi permintaan pengguna ke fungsi atau kelas view yang akan memproses dan merespons permintaan tersebut. urls.py dapat dikonfigurasi di tingkat proyek maupun di tingkat aplikasi individu.
5) manage.py : File ini digunakan untuk berbagai tugas manajemen proyek Django seperti menjalankan server, membuat aplikasi baru, melakukan migrasi database,

membuat superuser, mengumpulkan file statis, dan berbagai perintah lainnya yang memudahkan pengelolaan proyek Django. File manage,py ini memastikan bahwa perintah Django dapat dieksekusi dengan mudah.

Proyek search engine yang dibuat pada kegiatan kerja magang ini diberi nama search-engine yang kemudian dibuat juga folder atau app yang masing-masing memiliki fungsi dalam website. App atau folder utama proyek ini diberi nama search_engine dan memiliki file-file konfigurasinya diantaranya adalah sebagai berikut:

1) admin.py : File ini berfungsi untuk mendaftarkan model-model yang ada pada proyek ke dalam Django ke dalam Django admin interface. Seperti pada proyek ini, admin.py berisi kode yang mendaftarkan model dari tabel database yang digunakan diantaranya mendaftarkan model Document, RisalahRapat, SatuanKerja, dan Query. Berikut adalah isi dari file admin.py yang berfungsi untuk mendaftarkan model pada proyek ini.

```
1   from django.contrib import admin
2   from .models.document_model import Document, RisalahRapat, Query
3   from .models.satker_model import SatuanKerja
4
5
6   # Register your models here.
7   admin.site.register(Document)
8   admin.site.register(RisalahRapat)
9   admin.site.register(SatuanKerja)
10  admin.site.register(Query)
```

Gambar 3. 2 File admin.py

2) apps.py : File ini berisi konfigurasi untuk aplikasi Django yang mendefinisikan pengaturan dasar aplikasi. Pada proyek ini, file apps.py digunakan untuk mendefinisikan dan memuat model ranker dan lsi dari file pickle yang mana berfungsi pada alur searching search engine.
3) urls.py : File urls.py pada aplikasi yang dinamakan search_engine ini berfungsi untuk mengatur alur URL dalam proyek Django.File ini akan menentukan bagaimana permintaan URL yang berbeda-beda akan dipetakan ke view yang

sesuai di aplikasi. Berikut adalah lampiran kode dari file ini pada proyek search_engine yang dibangun.

```
15  urlpatterns = [
16      path('', home, name="home"),
17      path('search', ask, name='search'),
18      path('bsbi/do_indexing', do_bsbi, name='do_bsbi'),
19      path('docs/<int:id>', read_file, name='read_file'),
20      path('submit_feedback', submit_feedback, name='submit_feedback'),
21      path('create-lsi-model', create_lsi_model),
22      re_path(r'^search_engine/(?P<path>.*)$', serve, {'document_root': settings.MEDIA_ROOT}),
23  ]
24
```

Gambar 3. 3 urlpatterns pada File urls.py

Di gambar 3.3 terlihat ada 6 path yang dibuat yang mana masing-masing path merupakan informasi pemetaan URL yang terjadi di aplikasi. Sebagai contoh pada path pertama, memetakan URL root atau (‘’) ke view ‘home’ yang artinya adalah ketika aplikasi dijalankan maka root atau halaman pertama yang akan ditampilkan server adalah halaman home. Begitu pula dengan path lainnya yang mana bila URL dipanggil sebagai contoh lebih detail yaitu URL ‘bsbi/do_indexing’ maka akan dialihkan ke function do_bsbi di file views.py yang nantinya pada file views.py akan berperan sebagai jembatan untuk operasi selanjutnya yang akan aplikasi lakukan.

4) views.py : File ini berfungsi untuk menangani permintaan atau *request* dari pengguna dan menghasilkan respons. Setiap fungsi pada file ini memiliki informasi respons yang berbeda-beda untuk panggilan yang berbeda. Berikut adalah salah satu contoh function dari file views.py.

```
80  def do_bsbi(request):
81      BSBI_instance = BSBIIndex(data_dir='search_engine\data\documents',
82                                postings_encoding=VBEPostings,
83                                output_dir='search_engine\data\index')
84      BSBI_instance.do_indexing()
85      return HttpResponse("nice")
```

Gambar 3. 4 Function do_bsbi pada File views.py

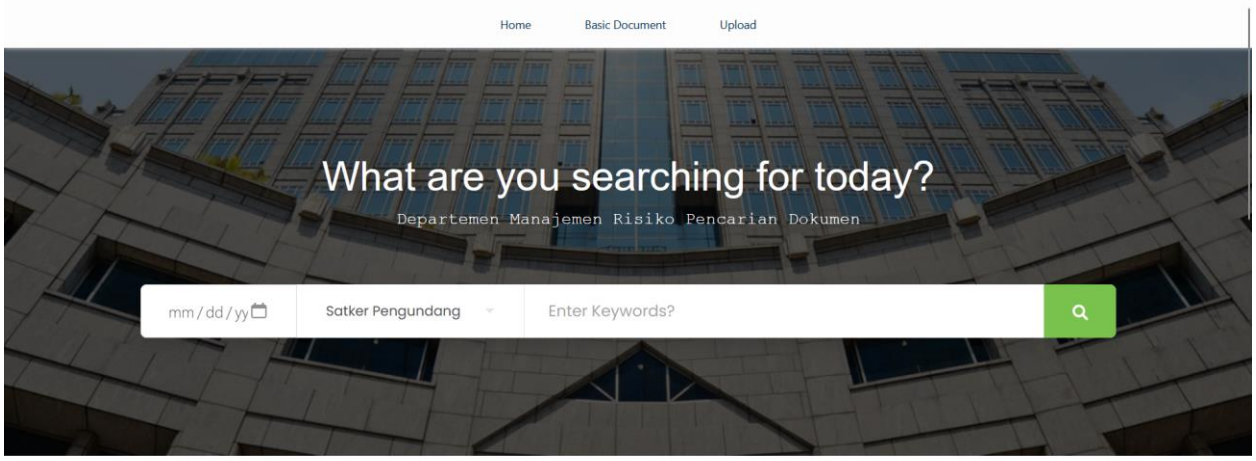
Kode diatas adalah contoh function home yang menerima request panggilan. Isi dari function ini terdapat variabel beserta parameter yang akan digunakan pada proses indexing atau fungsi pada function ini. Pada function ini juga mengarahkan ke class do_indexing yang berapa di bsbi.py dan di akhir dari function ini setelah

sudah berhasil menjalankan function do_indexing, akan dikembalikan response yang berjenis HTTP atau akan menampilkan halaman http dengan pesan “nice”. Jadi secara keseluruhan, urls dan views saling bekerja sama untuk mengkoordinasikan alur dan request dari user pada aplikasi Django.

4. Pembuatan User Interface

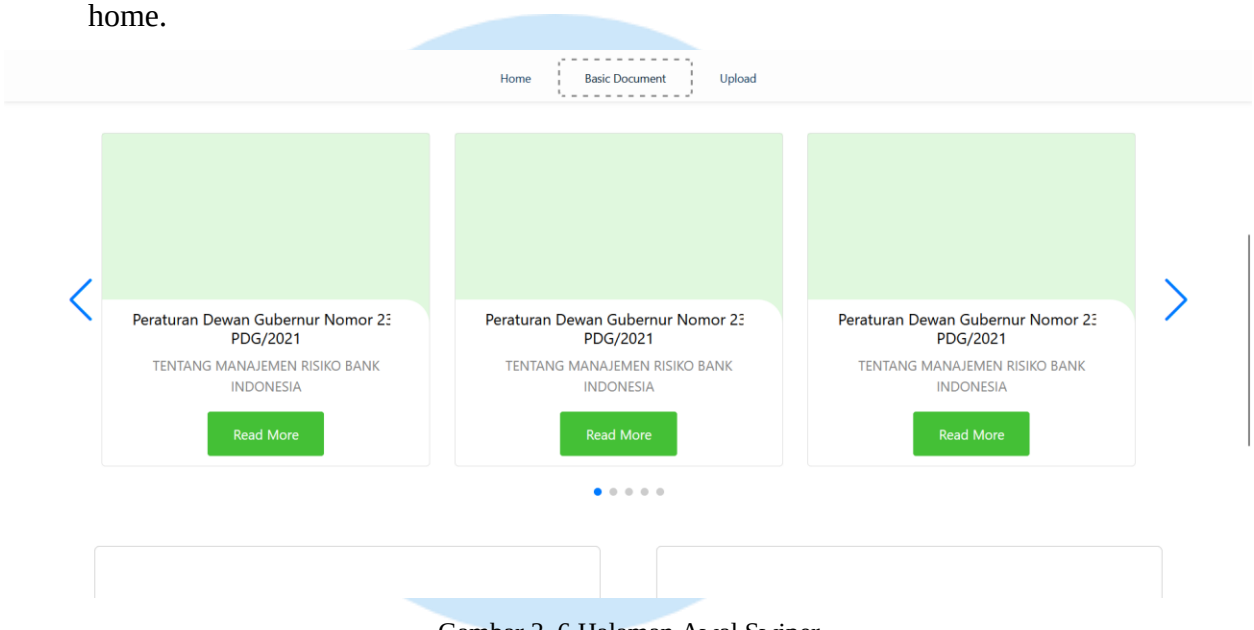
Dalam pembuatan *interface*, mahasiswa menggunakan software visual studio code. UI dari *search engine* ini sendiri terdiri dari dua halaman utama yaitu halaman home dan halaman *search result*. Halaman home terbagi menjadi tiga bagian utama yaitu *search bar, swiper* untuk dokumen-dokumen utama kelompok, dan terakhir adalah tempat untuk mengupload dokumen.

Bagian awal dari halaman home adalah *search bar* yang berupa form tempat input query dari pengguna. Query juga bisa dilengkapi dengan dua filter tambahan yaitu berdasarkan tanggal dan berdasarkan satuan kerja. Penggunaan filter bersifat opsional untuk pencarian lebih spesifik namun *search engine* tetap dapat digunakan tanpa mengisi filter tersebut. Berikut adalah tampilan dari bagian awal halaman home.



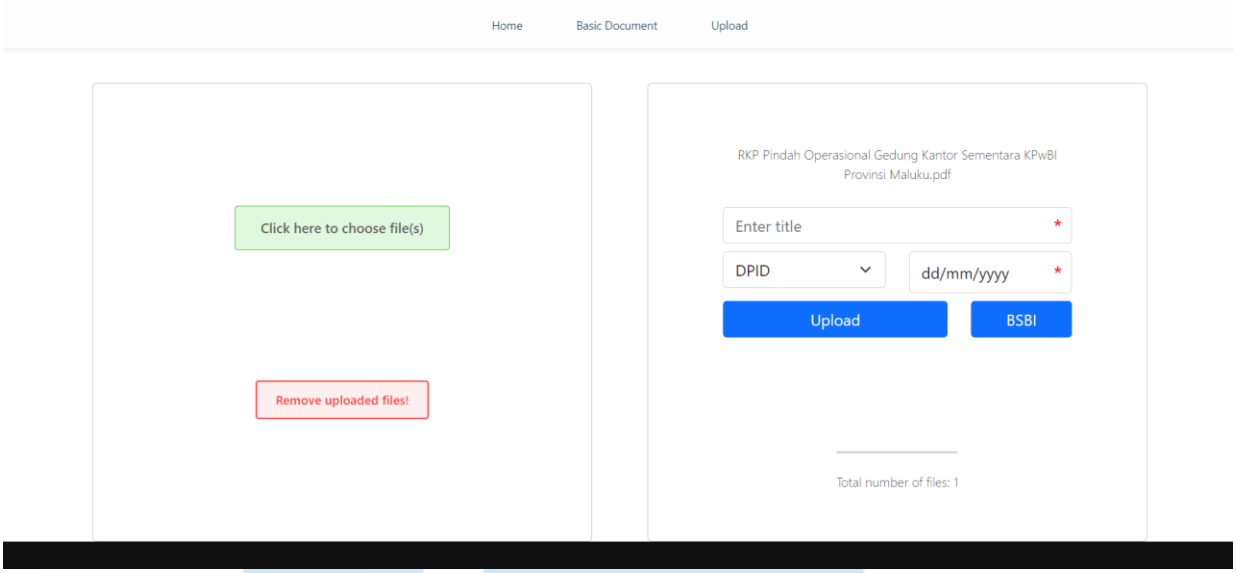
Gambar 3. 5 Halaman Awal Search Bar

Bagian kedua adalah swiper untuk menampilkan dokumen-dokumen yang menjadi dasar kegiatan kelompok 2 atau dokumen-dokumen yang sering digunakan oleh kelompok 2 ini. Untuk saat ini *swiper* bersifat *static* karena hanya dapat diubah lewat kode nya langsung. Berikut adalah tampilan dari bagian kedua dari halaman home.



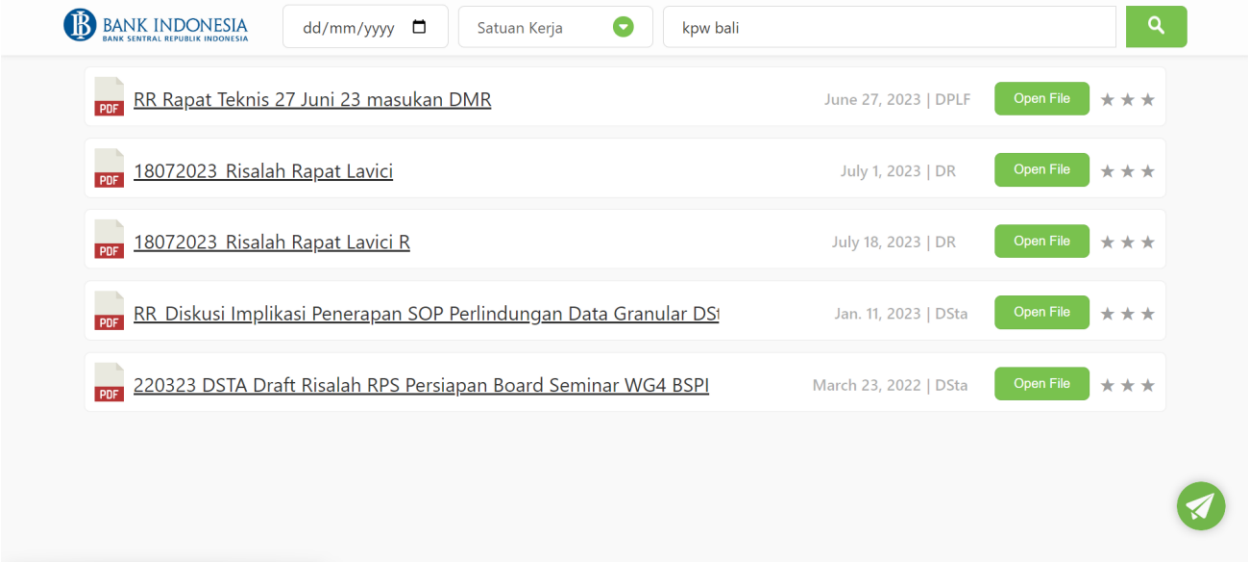
Gambar 3. 6 Halaman Awal Swiper

Bagian terakhir adalah tempat untuk meng-upload dokumen baru oleh pengguna. Karena tujuan dari *search engine* tahap awal ini terfokus pada risalah-risalah rapat kelompok 2, maka dokumen yang diupload harus dilengkapi dengan beberapa detail. Detail tersebut adalah satuan kerja pengundang rapat, tanggal diadakannya rapat, dan judul dari dokumen tersebut.



Gambar 3. 7 Halaman Awal Upload Dokumen

Halaman selanjutnya adalah *result page* atau halaman hasil. Disini akan ditampilkan dokumen-dokumen yang diurutkan dari yang memiliki skor relevansi tertinggi hingga yang paling rendah. Di halaman ini juga disediakan *search bar* sehingga user bisa langsung melanjutkan pencarian tanpa perlu kembali ke halaman utama. Masing-masing *result* dilengkapi dengan judul dokumen, tanggal rapat, satuan kerja pengundang, *feedback*, dan tombol *preview* dokumen. *Feedback* ini digunakan untuk pelatihan *machine learning* yang masih dikembangkan untuk *learning to rank search engine* ini. *Feedback* akan ditampung seiring *search engine* ini digunakan. Untuk tombol *preview* ini akan memunculkan dokumen di halaman baru tanpa mengunduh dokumen tersebut. Namun pengguna juga bisa mengunduh langsung dokumen dengan menekan judul dari dokumen yang diinginkan. Berikut adalah halaman hasil pencarian dari website.



Gambar 3. 8 Halaman Hasil Pencarian

5. Pembuatan alur upload

a) Gambaran Singkat Alur Upload

Upload dokumen secara garis besar mencakup lima file python yaitu document_model, query_model, satker_model, upload_service, dan uploader. Sedangkan untuk user interface tempat input file yang akan diupload berada pada home.html. Masing-masing file memiliki fungsi dan peranannya sendiri, maka dari itu selanjutnya adalah uraian dari alur upload dokumen pada *search engine* ini.

b) Uraian Alur Upload

1) Home.html

Pada home.html tempat user menginput dokumen yang ingin di upload diawali dengan penggunaan *function* runner. *Function* ini akan dimulai dengan mengambil daftar file yang dipilih oleh pengguna dari elemen dengan ID file_drop, kemudian memeriksa apakah file yang dipilih melebihi batasan yaitu 1 file dengan menggunakan kondisi if(0 > 1). Bila file sesuai dengan batasan yaitu hanya 1 file yang dipilih, akan dideklarasikan *variable* selectOptions untuk nantinya menampung opsi *dropdown* input filter satuan kerja. Selanjutnya akan dilakukan perulangan untuk file yang dipilih oleh

user. Dalam setiap perulangan, informasi tentang setiap file, seperti nama file, judul, satuan kerja, dan tanggal diadakannya rapat akan dimasukkan ke dalam elemen HTML yang sesuai, yang nantinya ditambahkan ke dalam elemen dengan ID 'file_list_container'. Berikut adalah kode dari input-input yang akan diproses berkaitan dengan dokumen yang diupload.

```
var file_name;
function runner(){
    var fileList = document.getElementById('file_drop').files;
    if(0 > 1){
        alert("Not more than one file can be uploaded!!!");
    }else{

        document.getElementById('file_list_container').innerHTML = "";
        var selectOptions = '';
        // populate the dropdown
        {% for satker_obj in satker %}
            selectOptions += '<option value="{{ satker_obj.id }}">{{ satker_obj.nama }}</option>';
        {% endfor %}
        console.log(selectOptions)
```

Gambar 3. 9 Kode Input dari File home.html

Input kedua adalah file_tags_upload_entry dan dilengkapi dengan id yang diambil terlebih dahulu dari id terakhir file pada fileList. File_tags_upload_entry adalah input judul dari dokumen yang diupload. Di sisi user, bagian ini akan berbentuk form dengan *placeholder* "Enter title for file" sehingga user mengetahui apa yang harus diinput pada form ini. Berikut adalah kode dari File_tags_upload_entry input.

```
document.getElementById('file_list_container').innerHTML +=
'<div class="row">' +
    '<div class="col-md-6">' +

    // input file name
    // id = file_list_i
    '<p id="file_list_i">' + fileList[i].name + '</p>' +
    '</div>' +
    '<div class="col-md-6">' +

    // input title
    // name = files_tags_upload_entry_
    '<div class="input-group">' +
    '<input type="text" placeholder="Enter title for file..." class="form-control files_tags_upload_entry"
    '</div>' +
    '</div>' +
    '<div class="col-md-6 mt-2">' +
```

Gambar 3. 10 Kode JavaScript Input File home.html

Selanjutnya adalah input satuan kerja dan tanggal diadakannya rapat. Input satker diambil dari *dropdown* dari sisi user yang diambil dari

database admin django. Satuan kerja yang terpilih, id nya akan tersimpan di *variable* selectOptions. Untuk tanggal, di sisi user akan menampilkan *calendar picker* dimana user hanya perlu memilih hari yang diinginkan dan nantinya akan tersimpan pada date_input_. Terakhir adalah button untuk menjalankan proses upload ini. *Button* dengan *class* btn ketika di klik akan menjalankan *function* submitForm yang mana akan meneruskan nilai-nilai yang sudah tersimpan di variabel-variabel sebelumnya. Dengan demikian, *function* runner bertanggung jawab untuk menampilkan informasi tentang setiap file yang dipilih oleh pengguna dalam bentuk elemen HTML yang sesuai. Berikut adalah kode dari kedua input dan tombol upload tersebut.

```
                // input satker dropdown
                // name = option_select_
                '<select class="form-select" name="option_select_' + i + '" id="option_select_' + i + '">' +
                selectOptions +
                '</select>' +
                '</div>' +
                '<div class="col-md-6 mt-2">' +

                // input date
                // name = date_input_
                '<input class="form-control" type="date" name="date_input_' + i + '" id="date_input_' + i + '">' +
                '</div>' +
                '<div class="col-md-6 mt-2">' +
                '<input class="form-control" type="text" placeholder="Enter title..." name="judul_' + i + '">' +
                '</div>' +
                '<div class="col-md-12 mt-2">' +
                '<button class="btn btn-primary w-100" type="button" onclick="submitForm()">Upload</button>' +
                '</div>' +
                '</div>';

            file_name = fileList[i].name;
            //console.log(file_name);
            temp_array.push(file_name); //temp_array holds name of file
        }
        document.getElementById('total_no_files').innerHTML = "Total number of files: " + fileList.length;
        }
}
```

Gambar 3. 11 Kode JavaScript Date dan Button

Ketika tombol upload di klik, maka *function* submitForm akan dijalankan. *Function* submitForm() dimulai dengan membuat objek FormData baru bernama formData untuk menyimpan data formulir. Selanjutnya, variabel fileList diinisialisasi dengan daftar file yang dipilih oleh pengguna dari elemen dengan ID 'file_drop'. Daftar ini kemudian di print untuk tujuan *debugging*. Selanjutnya, fungsi melakukan perulangan melalui daftar file yang dipilih. Dalam setiap perulangan, variabel fileName diinisialisasi dengan nama file saat ini dari daftar. Variabel tags, option, date,

dan title diinisialisasi dengan nilai-nilai yang diambil dari elemen formulir dengan atribut name yang sesuai dengan indeks iterasi saat ini. Nilai-nilai ini adalah tag, options, date, dan title yang dimasukkan oleh pengguna untuk setiap file. Setiap pasangan kunci-nilai ini ditambahkan ke objek formData menggunakan metode append(), dengan kunci yang sesuai dengan nama kunci yang diharapkan oleh server.

Setelah mengumpulkan semua data form, objek formData di print untuk tujuan debugging. Selanjutnya, dilakukan pengiriman data form ke server menggunakan AJAX atau XMLHttpRequest. XMLHttpRequest digunakan untuk mengirim permintaan HTTP ke server tanpa harus memuat ulang halaman dan pada langkah ini, objek XMLHttpRequest diinisialisasikan untuk membuat koneksi dengan server. Objek XMLHttpRequest diinisialisasi dengan metode open(). Metode ini digunakan untuk menentukan jenis permintaan HTTP (dalam hal ini, POST), URL tujuan permintaan (dalam hal ini, root URL '/'), dan apakah permintaan harus dilakukan secara *asynchronous* (true). Setelah objek XMLHttpRequest diinisialisasi dan pengaturan awalnya ditetapkan, metode send() digunakan untuk mengirim data ke server. Dalam konteks ini, objek formData yang berisi data formulir yang dikumpulkan dikirim sebagai *payload* permintaan.

Dengan menggunakan langkah-langkah ini, data formulir yang dikumpulkan oleh pengguna akan dikirim ke server secara *asynchronous*, yang berarti halaman web tidak perlu dimuat ulang saat permintaan dikirim. Ini memungkinkan antarmuka pengguna untuk tetap responsif dan terus berfungsi tanpa mengganggu pengalaman pengguna. Berikut adalah *function* dari submitForm.

```
411     function submitForm() {
412         // Loop through the file list to gather values of date and option select fields
413         var formData = new FormData(); // Create a new FormData object to store form data
414         console.log(document.querySelector('files'))
415         var fileList = document.getElementById('file_drop').files;
416         console.log(fileList)
417 
418         for (var i = 0; i < fileList.length; i++) {
419             var fileName = fileList[i].name;
420             var tags = document.querySelector('[name="files_tags_upload_entry_' + i + '"]').value;
421             var option = document.querySelector('[name="option_select_' + i + '"]').value;
422             var date = document.querySelector('[name="date_input_' + i + '"]').value;
423             var title = document.querySelector('[name="judul_' + i +'"]').value;
424 
425             // Append each file data along with tags, option select, and date to FormData object
426             formData.append('file_' + i, fileList[i]);
427             formData.append('tags_' + i, tags);
428             formData.append('option_' + i, option);
429             formData.append('date_' + i, date);
430             formData.append('title_'+i, title);
431         }
432 
433         console.log(formData)
```

Gambar 3. 12 Kode Function submitForm

2) views.py

Pada Django, request dari satu halaman ke halaman lain dikendalikan pada file views.py termasuk pada pemanfaatannya kali ini. Sebelumnya pada *function* submitForm, kita melakukan POST yang mana mengirimkan request untuk POST ke server. Karena function submitForm berada pada file home.html, maka request POST tersebut akan berkoresponden dengan function home yang dibuat pada views.py. Terlebih dahulu berikut adalah kode dari function home pada file views.py.

```
22  upload_service = UploadService()
23 
24  @csrf_exempt
25  def home(request):
26 
27      if request.method == 'POST':
28          upload_service.save(request.POST.get('option_0'),
29                              request.POST.get('date_0'),
30                              request.POST.get('title_0'),
31                              request.FILES.get('file_0'))
32          print(request.POST)
33          print(request.FILES)
34      list_satker = SatuanKerja.objects.filter()
35      return render(request, "home.html", {"satker":list_satker})
36 
```

Gambar 3. 13 Function home pada File views.py

Dalam function home, bila request method yang diterima berjenis POST, maka akan dipanggil upload_service function save yang mana akan mengolah parameter-parameter yang didapati dari request POST dari home.html tersebut atau dari function submitForm. Parameter tersebut akan di GET untuk di proses menggunakan upload_service, parameter tersebut ialah option atau pilihan dari dropdown satuan kerja, date atau tanggal rapat, title atau judul, dan file atau nama file dan disimpan ke tabel RisalahRapat. Setelah tersimpan, maka akan mencetak atau print POST pada command prompt beserta mencetak semua file yang dikirim dalam permintaan tersebut dengan request.FILES ke command prompt. Untuk alur upload dokumen ke database terjadi di dalam pengkondisian 'if'.

3) upload_service.py

UploadService adalah *class* yang bertanggung jawab untuk menyediakan layanan yang diperlukan dalam proses pengunggahan file atau upload file. Kali ini, terdapat sebuah metode statis bernama save yang mana mengambil parameter-parameter input yaitu satker_pengundang_id, tanggal, judul, dan file. Dalam function save ini sebuah variabel obj yang mana akan membuat object menggunakan *class* panggilan yang bernama RisalahRapat. Secara singkat, *class* RisalahRapat bertujuan untuk menyimpan data yang terkait dengan risalah rapat ke dalam basis data atau database django. Dalam pembuatan object ini, judul, satuan kerja, file dan waktu rapat akan berisi sesuai dengan nilai-nilai yang diterima dari parameter. Maka dari itu isi dari parameter object ada judul=judul, file=file, dan seterusnya yang artinya menerima kiriman nilai. Setelah objek RisalahRapat dibuat, akan dilakukan print(obj) yang digunakan untuk megeprint objek yang baru saja dibuat. Jadi, function UploadService ini menyediakan sebuah metode untuk menyimpan

informasi tentang input dokumen yang diunggah oleh user. Berikut adalah kode dari function UploadService.

```
4   class UploadService:
5
6       @staticmethod
7       def save(satker_pengundang_id, tanggal, judul, file):
8           obj = RisalahRapat.objects.create(judul=judul,
9                                             satker_pengundang=SatuanKerja.
10                                            objects.filter(id=satker_pengundang_id).first(),
11          file=file, waktu_rapat=tanggal)
12          print(obj)
```

Gambar 3. 14 Class UploadService pada File upload_service.py

4) document_model.py

Pada django, telah tersedia interface admin secara otomatis pada setiap proyek nya. Admin interface ini merupakan database lokal django yang membaca metadata dari proyek lokal kita. Django admin memungkinkan user admin untuk mengelola data yang disimpan dalam database dengan mudah tanpa menulis kode tampilan/interface. Untuk membuat tabel database di django admin sendiri perlu dilakukan secara manual dan biasa disebut sebagai models. Untuk proyek ini, memiliki tabel utama yang bernama dokumen dan tabel turunan yang bernama RisalahRapat. Berikut adalah model dari tabel Document.

```
10  # Create your models here.
11  def upload_document_with_last_id(instance, filename):
12      last_document_id = Document.objects.last().pk
13      return upload_document(instance, filename, last_id=last_document_id)
14
15  class Document(models.Model):
16      # buat atribut input
17      judul = models.TextField(max_length=512)
18      file = models.FileField(upload_to=upload_document_with_last_id)
19      created_at = models.DateTimeField(auto_now_add=True, null=True)
20      last_modified = models.DateTimeField(auto_now=True, null=True)
21
22      def save(self, *args, **kwargs):
23          judul, file_ext = os.path.splitext(self.file.name)
24          if self.pk:
25              self.file_name = str(self.pk) + file_ext
26          else:
27              last_document = Document.objects.last()
28              self.file.name = str(last_document.id + 1 if last_document else 1) + f"_{judul}" + file_ext
29          super(Document, self).save(*args, **kwargs)
30
31      def __str__(self):
32          return self.judul
```

Gambar 3. 15 Model Class Document pada File document_model.py

Pertama-tama dibuat sebuah function yang bernama upload document_with_last_id yang mana berfungsi untuk menetapkan alamat dari dokumen yang akan diupload. Hal ini dilakukan dengan menggunakan *class* upload_document. Pada function upload_dokumen sendiri terlihat dilakukan return path yang mana merupakan path tempat dokumen-dokumen yang diupload akan disimpan.dan berikut kodenya. Berikut adalah kode dari function upload_document ini.

```python
import os

# menentukan lokasi penyimpanan file yang disimpen
def upload_document(instance, filename, last_id):
    block = (last_id + 1) // 100
    # membagi last_id + 1 dengan 100 dan mengambil nilai pembulatan ke bawah.
    # Ini berfungsi untuk membagi penyimpanan file ke dalam blok-blok tertentu,
    # mungkin untuk mengorganisir penyimpanan file dengan lebih baik.

    return os.path.join('data','documents',str(block), filename)
    # buat path yang artinya di folder data/documents/block(jenis string)/filename
```

Gambar 3. 16 Function Upload_document

Selanjutnya kita memasuki *class* Document atau model yang mengatur bentuk dari database django admin. Pertama kita buat terlebih dahulu atribut input beserta jenis dari masing-masing input yaitu judul dengan jenis TextField, file dengan jenis FileField dan tambahan path yang dibuat dengan function upload_document_with_last_id, lalu created_at yang berjenis DateTimeField, dan last_modified yang juga berjenis DateTimeField. Di Dalam *class* ini juga dibuat dua function yaitu save dan __str__. Function save ini berfungsi untuk meyimpan objek dokumen ke database django yang mana terjadi pada super(Document, self).save(*args, **kwargs) yang artinya menyimpan isi dari nilai-nilai yang di *parsing* ke *class* Document tersebut.

Tabel turunan dari tabel Document adalah tabel RisalahRapat dan diinisialisasikan tabelnya dengan *class* RisalahRapat. Karena tabel ini diturunkan dari Document, maka atribut-atribut tabel Dokumen berlaku dan juga dimiliki oleh tabel RisalahRapat. Bedanya adalah pada tabel RisalahRapat kita menambahkan dua atribut baru yaitu satker_pengundang

dan waktu_rapat. Sama seperti fungsi dari *class* Document, *class* ini juga akan menyimpan nilai-nilai yang di *parsing* ke *class* RisalahRapat dan menambahkannya ke tabel, dan untuk kali ini akan menyimpan atribut judul, dile, created_at, last_modified, satker_pengundang, dan waktu_rapat. Berikut adalah *class* dari RisalahRapat.

```
34  class RisalahRapat(Document):
35      satker_pengundang = models.ForeignKey(SatuanKerja, on_delete=models.SET_NULL, null=True)
36      waktu_rapat = models.DateField(null=True, blank=True)
37
38      def save(self, *args, **kwargs):
39          super(RisalahRapat, self).save(*args, **kwargs)
40
41      def __str__(self):
42          return self.judul
43
44  class Query(models.Model):
45      query = models.CharField(max_length=256)
46
```

Gambar 3. 17 Model Class RisalahRapat pada File document_model.py

c) Kesimpulan Alur Upload

Alur *upload* dokumen pada *search engine* ini dimulai dari halaman Home.html, di mana pengguna diberi kemampuan untuk upload dokumen dengan mengisi formulir yang telah disediakan. Ketika pengguna mengklik tombol untuk memilih file, sebuah *function* yang disebut runner akan dijalankan. *Function* ini bertugas untuk mengambil daftar file yang dipilih oleh pengguna dari elemen dengan ID file_drop. Dalam pengambilan ini, *function* akan memeriksa apakah pengguna telah memilih satu file saja, sesuai dengan batasan yang ditentukan. Jika batasan terpenuhi, maka variabel selectOptions akan dideklarasikan untuk menampung opsi *dropdown* input untuk memilih satuan kerja terkait.

Setelah memastikan bahwa hanya satu file yang dipilih, informasi terkait file tersebut seperti nama file, judul, satuan kerja, dan tanggal diadakannya rapat akan dimasukkan ke dalam elemen HTML yang sesuai. Informasi-informasi ini kemudian akan ditampilkan kepada pengguna di halaman Home. Selanjutnya, pengguna akan diminta untuk mengisi judul dokumen melalui input dengan ID file_tags_upload_entry. Input ini akan menampilkan *form* dengan *placeholder*

"Enter title for file" untuk memberi petunjuk kepada pengguna mengenai apa yang harus diinput.

Selanjutnya, pengguna akan diminta untuk memilih satuan kerja terkait dan tanggal diadakannya rapat. Pilihan satuan kerja akan diambil dari *dropdown* yang berasal dari database admin Django. Setelah pengguna memilih satuan kerja dan tanggal rapat, informasi tersebut akan disimpan dalam variabel dan akan digunakan dalam proses pengiriman data.

Ketika pengguna mengklik tombol upload, *function* submitForm akan dijalankan. *Function* ini akan membuat objek FormData untuk menyimpan data formulir yang telah diisi oleh pengguna, termasuk file yang dipilih, judul dokumen, satuan kerja terkait, dan tanggal diadakannya rapat. Data formulir ini akan dikirim ke server secara *asynchronous* menggunakan XMLHttpRequest. Di sisi server, permintaan POST dari halaman Home akan ditangani oleh *function* home dalam views.py. *Function* ini akan memproses data yang diterima dari formulir, termasuk file yang diunggah, judul dokumen, satuan kerja terkait, dan tanggal diadakannya rapat. Data ini kemudian akan disimpan menggunakan layanan yang disediakan oleh *class* UploadService.

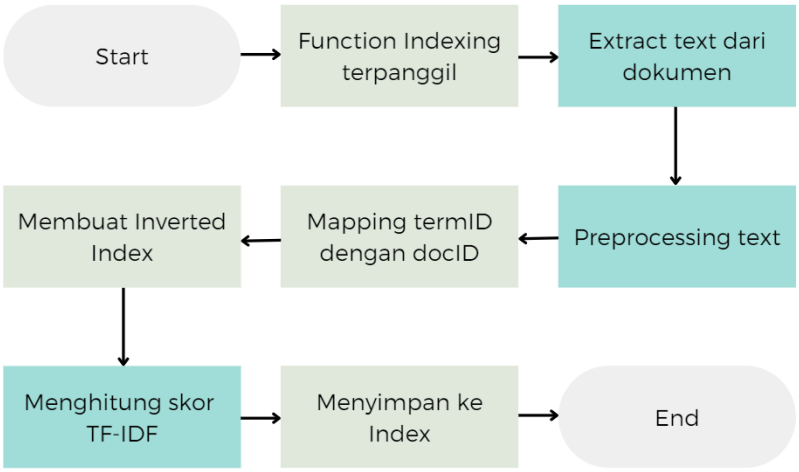
Class UploadService bertanggung jawab untuk memproses data yang diterima dari formulir dan menyimpannya ke dalam basis data Django. Dengan demikian, proses upload dokumen ini memungkinkan pengguna untuk dengan mudah mengunggah dokumen ke dalam sistem, dengan memastikan bahwa informasi terkait dokumen tersebut tersimpan dengan benar dalam database.

6. Pembuatan alur Indexing

a) Gambaran Singkat Alur Indexing

BSBI atau Block Sort-Based Indexing adalah algoritma atau metode untuk pembuatan *inverted index*. Penggunaan BSBI dalam pembuatan *inverted index* meningkatkan efisiensi karena memungkinkan untuk mengelola dan mengindeks koleksi dokumen berukuran besar dengan memori yang terbatas. Hal ini dapat diraih karena dengan menggunakan BSBI, indeks dari dokumen disimpan dalam

blok-blok terstruktur yang nantinya akan di *compress*. Pada search engine yang dibuat, kegiatan ini dilakukan setiap kali dokumen baru diupload atau masuk ke database. Metode BSBI ini akan menggunakan 3 *function* utama yaitu do_indexing, parsing_block, dan write_to_index. Secara ringkas, metode ini akan *scan* semua data di koleksi, memanggil parsing_block untuk *parsing* dokumen dan memanggil write_to_index yang melakukan *inversion* di setiap block dan menyimpannya di index baru. Berikut penjabaran singkat dari masing-masing *function*. Berikut ini adalah workflow secara garis besar alur upload.



Gambar 3. 18 Workflow alur indexing

Kontribusi mahasiswa dalam alur ini ditandai dengan cell berwarna biru diantaranya adalah:

- Tahapan *extract text* dari dokumen mahasiswa membantu dalam pembuatan *function* parsing_block yang berfungsi untuk membaca dan mengambil teks dari dokumen.
- Pada tahap *preprocessing text,* mahasiswa membuat bagian untuk membersihkan teks yang telah di *extract* diantaranya *stemming*, penghapusan *stopwords,* dan *tokenization.*
- Pada tahap Menghitung skor TF-IDF, mahasiswa membuat function untuk menghitung TF-IDF yang mencakup rumus pengukuran skor TF-IDF.

b) Uraian Alur Indexing

1) do_indexing

Function ini adalah *function* utama yang terpanggil ketika akan dilakukan BSBI atau *indexing* dokumen. Pertama, *function* ini akan meng-*scan* setiap sub-direktori dalam folder tempat penyimpanan dokumen. Dari direktori-direktori yang sudah dibaca akan di *sort* atau diurutkan secara alfabetis. Masing-masing sub-direktori pada daftar ini selanjutnya akan diproses menggunakan *function* parsing_block yang mana menghasilkan pasangan antara term-document yang kemudian akan ditulis ke dalam indeks sementara. Berikut adalah bagian pemanggilan parsing_block dan meng-assign nya ke td_pairs.

```
474     def do_indexing(self):
484         # loop untuk setiap sub-directory di dalam folder collection (setiap block)
485         for block_dir_relative in tqdm(sorted(next(os.walk(self.data_dir))[1])):
486             td_pairs = self.parsing_block(block_dir_relative)
487
```

Gambar 3. 19 Function do_indexing

Selanjutnya td_pairs hasil dari penggunaan parsing_block akan diproses menggunakan write_to_index yang mana juga akan menggunakan *class* InvertedIndexWriter. Penggunaan *function* dan *class* ini akan melakukan pembuatan *postings list* yaitu list dokumen-dokumen dimana *term* i muncul, *tf list* yaitu frekuensi kemunculan *term* tersebut muncul di setiap dokumen, serta mengurutkan *term* secara alfabetis dan terakhir akan memasukkannya ke index. Pada tahap ini td_pairs akan di *assign* value = None untuk menghapus memori perulangan dan dapat memulai perulangan selanjutnya dengan ruang kosong. Berikut adalah bagian pemanggilan *class* InvertedIndexWriter.

```
488             # jadi untuk setiap putaran block bakal dimasukin ke index_id yaitu indeks sementara
489             index_id = 'intermediate_index_'+block_dir_relative
490             # nah abis udh dimasukin ke index_id yang berisi block saat ini,
491             # baru dimasukin ke tempat penyimpanan seluruh index sementara yaitu di intermediate_indices
492             self.intermediate_indices.append(index_id)
493
494             with InvertedIndexWriter(index_id, self.postings_encoding, directory=self.output_dir) as index:
495                 self.write_to_index(td_pairs, index)
496                 td_pairs = None
497         self.save()
498
```

Gambar 3. 20 Class InvertedIndexWriter

Setelah semua dokumen dalam setiap blok telah diindeks dengan hasilnya juga disimpan dalam indeks sementara, selanjutnya akan dilakukan

menggabungkan indeks sementara ini dengan menggunakan *class* InvertedIndexReader. Berikut adalah kode dari InvertedIndexReader beserta penjelasannya.

```
100  class InvertedIndexReader(InvertedIndex):
105      def load_score(self):
106          if(len(self.term_upper_bound) == 0):
107              avdl = self.avdl
108              k1 = 1.2
109              b = 0.75
110              for token_id in self.terms:
111                  posting_list, tf_list = self.get_postings_list(token_id)
112
113                  maxBM25 = 0
114                  maxTFIDF = 0
115
116                  if(len(posting_list) == 0):
117                      continue
118
119                  idf = math.log10(len(self.doc_length) / len(posting_list))
120                  for i, doc in enumerate(posting_list):
121                      tf = tf_list[i]
122                      dl = self.doc_length[doc]
123                      bm25 = idf * ((k1 + 1) * tf) / (k1 * ((1 - b) + b * dl / avdl) + tf)
124                      maxBM25 = max(maxBM25, bm25)
125
126                      tfidf = idf * (1 + math.log10(tf_list[i]))
127
128                      maxTFIDF = max(maxTFIDF, tfidf)
129
130                  self.term_upper_bound[token_id] = (maxTFIDF, maxBM25)
```

Gambar 3. 21 Class InvertedIndexReader

Pada *class* InvertedIndexReader, terdapat beberapa *function* yang digunakan. *Function* pertama adalah load_score yang mana digunakan untuk memuat skor BM25 dan TF-IDF untuk setiap term dalam indeks. Diawali dengan IF yang mengecek apakah term_upper_bound ini kosong atau tidak. term_upper_bound ini sendiri adalah sebuah kamus yang menyimpan nilai-nilai skor, seperti BM25 dan TF-IDF yang akan kita gunakan untuk setiap *term* dalam indeks. Maka karena kita saat ini ingin mengisi term_upper_bound tersebut dengan nilai BM25 dan TF-IDF dari seluruh term, kita harus pastikan terlebih dahulu bahwasannya dia kosong.

Perlu diketahui bahwa tujuan dari pembuatan TF-IDF atau Term Frequency-Inverse Document Frequency ini ialah mengetahui skor atau bobot relevansi suatu term terhadap topik yang dibicarakan suatu dokumen dengan membandingkannya atau mengkalkulasikan frekuensi term tersebut didalam

satu dokumen dan dalam keseluruhan corpus. Berikut adalah rumus dari perhitungan TF-IDF.

$$\text{TF} = \{\ 1\ +\ log_{10}(f_t, d),\ 0\}$$

Dimana nilai ft,d adalah frekuensi term (t) pada document (d). Jadi jika suatu kata atau term terdapat dalam suatu dokumen sebanyak x kali maka diperoleh bobot = 1 + log(10) = 1.699. Tetatpi jika term tidak terdapat dalam dokumen tersebut, bobotnya adalah nol (0) [10].

Bila kita jabarkan secara teori, ada beberapa konsep yang mempengaruhi bobot ini sebagai berikut:

- Term tertentu bisa memiliki bobot yang tinggi karena memang jarang digunakan ataupun memang memiliki makna yang khusus. Jadi ketika term ini muncul, ada kemungkinan besar dokumen tersebut akan membahas term tersebut yang maka dari itu dapat meningkatkan bobot dari term tersebut.
- Lainhalnya bila suatu term muncul berkali-kali di banyak dokumen, bisa dianggap term ini tidak memiliki arti spesial. Maka dari itu kata-kata stopwords atau kata sambung kita hapus di bagian preprocessing karena tidak berpengaruh pada topik yang dibahas suatu dokumen.
- Kebalikannya, term yang hanya muncul beberapa kali di beberapa dokumen saja memiliki bobot yang lebih tinggi karena memiliki kemungkinan berpengaruh akan topik yang dibahas suatu dokumen.

Selanjutnya kita akan atur parameter yang akan digunakan diantaranya adalah avdl atau average doc length yang merupakan rata-rata dari panjang dokumen, k1 dan b yang merupakan parameter yang digunakan dalam perhitungan BM25 yaitu model perankingan. Selanjutnya akan dilakukan perulangan untuk setiap token_id di list terms. Perulangan ini berfungsi untuk mengambil posting_list dan tf_list terkait menggunakan metode get_postings_list dari token_id tersebut. Kemudian akan dilakukan

perulangan untuk membaca setiap token_id pada terms. Untuk setiap term, dilakukan pengambilan posting_list dan tf_list terkait dengan *term* tersebut menggunakan get_postings_list(token_id).

Selanjutnya kita inisialisasikan variabel maxBM25 dan maxTFIDF dengan nilai awal 0. Hal ini dilakukan sebelum melakukan perulangan yang akan menggunakan *term* tersebut. Hal ini juga memastikan bahwa nilai maksimum untuk skor BM25 dan TFIDF akan deperbarui selama perulangan ketika nilai yang dihitung lebih besar dari nilai maksimum sebelumnya. Masuk ke tahap berikutnya dimana kita akan menghitung maxBM25 dan maxTFIDF dari setiap term. Dalam perhitungan BM25 dan TFIDF, kita juga memerlukan idf dari dokumen tersebut. Maka terlebih dahulu kita buat perhitungan idf yang menggunakan doc_length atau total dokumen keseluruhan dan posting list yaitu total dokumen dimana *term* tersebut muncul yang disimpan di parameter idf. Untuk perhitungan BM25 dan TFIDF akan dijabarkan sebagai berikut:

```
119                 idf = math.log10(len(self.doc_length) / len(posting_list))
120                 for i, doc in enumerate(posting_list):
121                     tf = tf_list[i]
122                     dl = self.doc_length[doc]
123                     bm25 = idf * ((k1 + 1) * tf) / (k1 * ((1 - b) + b * dl / avdl) + tf)
124                     maxBM25 = max(maxBM25, bm25)
125 
126                     tfidf = idf * (1 + math.log10(tf_list[i]))
127 
128                     maxTFIDF = max(maxTFIDF, tfidf)
129 
130                 self.term_upper_bound[token_id] = (maxTFIDF, maxBM25)
```

Gambar 3. 22 Perhitungan BM25 dan TF-IDF

- for i, doc in enumerate(posting_list): Dilakukan perulangan melalui setiap dokumen yang mengandung *term* saat ini.
- tf = tf_list[i]: Mendapatkan frekuensi *term* (tf) untuk dokumen saat ini dari tf_list.
- dl = self.doc_length[doc]: Mendapatkan panjang dokumen (dl) untuk dokumen saat ini dari self.doc_length.
- bm25 = idf * ((k1 + 1) * tf) / (k1 * ((1 - b) + b * dl / avdl) + tf): Menghitung skor BM25 untuk dokumen saat ini. Rumus BM25 digunakan untuk

menilai relevansi suatu *term* terhadap dokumen dalam sebuah koleksi. Variabel k1 dan b adalah parameter yang telah ditentukan sebelumnya, sedangkan avdl adalah panjang dokumen rata-rata dalam koleksi.

- maxBM25 = max(maxBM25, bm25): Memperbarui nilai maksimum BM25 jika skor BM25 yang baru dihitung lebih besar dari nilai maksimum sebelumnya.
- tfidf = idf * (1 + math.log10(tf_list[i])): Menghitung skor TF-IDF untuk dokumen saat ini. Skor TF-IDF adalah produk dari IDF dengan logaritma dari frekuensi *term* dalam dokumen.
- maxTFIDF = max(maxTFIDF, tfidf): Memperbarui nilai maksimum TF-IDF jika skor TF-IDF yang baru dihitung lebih besar dari nilai maksimum sebelumnya.

Setelah melakukan perulangan untuk semua dokumen yang mengandung *term* tersebut, nilai maksimum dari skor TF-IDF dan BM25 disimpan dalam term_upperbound untuk *term* tersebut menggunakan: self.term_upper_bound[token_id] = (maxTFIDF, maxBM25): Ini memungkinkan penggunaan nilai maksimum skor sebagai upper bound dalam proses pencarian dokumen relevan berdasarkan skor TF-IDF atau BM25.

Jadi, pada akhirnya kembali ke *function* do_indexing, *term* ataupun dokumen telah melewati banyak tahapan dan proses. Dimulai dari pembuatan objek InvertedIndexWriter yang digunakan untuk menulis index gabungan. Selanjutnya InvertedIndexReader akan membaca indeks sementara dari file-file yang ada dan setelah semua indeks sementara dibaca, mereka akan digabungkan menjadi satu indeks besar menggunakan metode merge_index yang mana hasilnya akan ditulis ke dalam objek. InvertedIndexWriter yang telah dibuat sebelumnya. Berikut adalah bagian dimana InvertedIndexWriter dibuat kembali yang diakhiri dengan penggabungan indeks oleh merge_index.

```
499         with InvertedIndexWriter(self.index_name, self.postings_encoding, directory=self.output_dir) as merged_index:
500             with contextlib.ExitStack() as stack:
501                 indices = [stack.enter_context(InvertedIndexReader(index_id, self.postings_encoding, directory=self.output_dir))
502                            for index_id in self.intermediate_indices]
503                 self.merge_index(indices, merged_index)
```

Gambar 3. 23 Function do_indexing

2) parsing_block

Function parsing_block ini akan dipanggil oleh *function* utama indexing yaitu dari *function* do_indexing. Fungsi utama dari *function* ini adalah untuk membaca dokumen-dokumen yang terdapat di dalam sebuah direktori, mengambil teks dari setiap dokumen, yang kemudian memproses teks tersebut untuk mendapatkan pasangan termID dan docID. Berikut adalah keseluruhan dari *function* parsing_block. Bagaimana *function* ini beroperasi adalah pertama akan dimulai dengan membuat sebuah list kosong yang akan menyimpan pasangan termID dan docID.

```
96      def parsing_block(self, block_path):
134         td_pairs = []
135         for dirpath, dirnames, filenames in os.walk(f"{self.data_dir}/{block_path}"):
136             for file_name in filenames:
137                 file_path = os.path.join(dirpath, file_name)
138                 if file_name.endswith('.pdf'):
139                     pdfFileObj = open(file_path, 'rb')
140                     with open(file_path, 'r', encoding="utf-8") as file:
141                         pdf_reader = PyPDF2.PdfReader(pdfFileObj)
142                         num_pages = len(pdf_reader.pages)
143                         content = ""
144                         for page_num in range(num_pages):
145                             page = pdf_reader.pages[page_num]
146                             content += page.extract_text()
147                     # content = file.read()
148                 elif file_name.endswith('.docx'):
149                     doc = docx.Document(file_path)
150                     content = ""
151                     for paragraph in doc.paragraphs:
152                         content += paragraph.text
153                 tokens = self.pre_processing_text(content)
154                 # tokenizer_pattern = r'\w+'
155                 # tokens = re.findall(tokenizer_pattern, tokens)
156                 for token in tokens:
157                     td_pairs.append((self.term_id_map[token], self.doc_id_map[file_name]))
158         return td_pairs
```

Gambar 3. 24 Function parsing_block

Selanjutnya, *function* akan melakukan perulangan melalui setiap dokumen dan direktori. Untuk setiap dokumen yang ditemukan, *function* akan memeriksa tipe dokumen apakah berupa dokumen PDF atau dokumen DOCX. Jika dokumen berekstensi PDF, *function* akan membuka dokumen tersebut dan dengan menggunakan PyPDF2 akan membaca isi dari dokumen

di setiap halamannya. Kemudian, teks dari setiap halaman tersebut akan ditambahkan ke dalam sebuah string yang menyimpan *term* dari seluruh dokumen yang sudah dibaca. Bila dokumen berbentuk DOCX, *function* akan membuka dokumen tersebut menggunakan modul python-docx dan mengambil teks dari setiap paragraf dalam dokumen. Teks dari setiap paragraf juga akan ditambahkan ke dalam string yang menyimpan konten dari seluruh dokumen yang telah dibaca tersebut.

Setelah dokumen-dokumen sudah berhasil dibaca dan diambil kontennya, *function* akan memanggil *function* lain yang bernama pre_processing_text untuk memproses teks tersebut. *Function* pre_processing_text ini akan melakukan preprocessing pada konten-konten yang sudah dimasukan ke string sebelumnya. *Function* akan melakukan tokenization atau tokenisasi yaitu memisahkan kata per kata sehingga menghasilkan daftar kata-kata. Selanjutnya, setiap kata yang telah di tokenisasi tersebut atau kata yang ada di daftar akan diubah menjadi huruf kecil menggunakan metode .lower(). Daftar kata yang sudah seragam selanjutnya akan mengalami proses stemming yaitu menerjemahkan seluruh kata ke bentuk dasarnya. Preprocessing lainnya ialah menghapus kata-kata stopwords. Kata stopwords adalah kata sambung seperti ‘di’, ‘ke’, ‘lalu’ dan lainnya. Hal ini dilakukan karena kata sambung tidak memiliki weight atau dampak ke topik yang dibahas suatu dokumen sehingga dianjurkan untuk dihapus. Akhirnya, daftar kata-kata yang telah diproses tersebut akan dikembalikan sebagai output dari *function* pre_processing_text. Berikut adalah kode dari *function* pre_processing_text.

```
70      def pre_processing_text(self, content):
73          """
74          # https://github.com/ariaghora/mpstemmer/tree/master/mpstemmer
75          stemmer = MPStemmer()
76
77          tokenizer_pattern = r'\w+'
78          words = re.findall(tokenizer_pattern, content)
79          filtered_sentence = [stemmer.stem(word.lower()) for word in words]
80          return filtered_sentence
```

Gambar 3. 25 Function pre_processing_text

Kembali ke *function* parsing_block yang mana telah memiliki daftar kata-kata yang sudah melewati preprocessing dari *function* pre_processing_text. Setiap kata di daftar kata-kata ini disebut sebagai token. Namun, list kata-kata ini belum tentu telah memiliki id karena ada kemungkinan kata tersebut baru dan belum ada didalam *dictionary* ataupun memang baru pertama kali melakukan indexing seluruh dokumen. Maka dari itu, akan digunakan *class* IdMap untuk melakukan fungsi ini yaitu memberikan id kepada masing-masing *term* dan masing-masing dokumen. Berikut adalah tampilan dari *class* IdMap.

```
class IdMap:

    def __init__(self):
        self.str_to_id = {}
        self.id_to_str = []

    def __len__(self):
        """Mengembalikan banyaknya term (atau dokumen) yang disimpan di IdMap."""
        return len(self.id_to_str)

    def __get_id(self, s):
        if self.str_to_id.get(s) != None:
            return self.str_to_id[s]
        self.str_to_id[s] = len(self.id_to_str)
        self.id_to_str.append(s)
        return len(self.id_to_str) - 1

    def __get_str(self, i):
        """Mengembalikan string yang terasosiasi dengan index i."""
        return self.id_to_str[i]

    def __getitem__(self, key):
        if type(key) == int:
            return self.__get_str(key)
        return self.__get_id(key)
```

Gambar 3. 26 Class IdMap

Pada inisialisasi *class* IdMap, dibuat sebuah struktur data yaitu str_to_id atau string ke id yang akan disimpan dalam python *dictionary* dan id_to_str atau id ke string yang disimpan dalam python list. Selanjutnya *function* __get_id beroperasi untuk mengembalikan id i yang berkorespondensi dengan sebuah string (*term* atau nama dokumen) s. Jika s atau tidak ada pada IdMap, maka *term* atau s tersebut akan di assign integer baru sebagai id nya. Maka dari itu, dari penggunaan *class* ini *function*

parsing_block akan menerima pasangan termId, docId. Contohnya adalah saat ada dokumen baru yang masuk dan diberikan id baru oleh IdMap misal saja 10, sedangkan untuk *term* yang sudah ada misal kata 'halo' sudah ada di *dictionary* dan ber id 1 ataupun *term* baru dari dokumen ini contoh *term* 'world' dengan id 19 maka td_pairs ini atau pasangan termId dan docId berupa [(1, 10), …, (19, 10), …].

3) write_to_index

Function write_to_index akan memproses value dari td_pairs yang telah didapatkan sebelumnya dan juga akan menggunakan *Class* InvertedIndexWriter. *Class* InvertedIndexWriter adalah *class* yang mengimplementasikan bagaimana caranya menulis secara efisien *inverted index* yang disimpan di sebuah file. *Function* write_to_index akan melakukan inversion pada td_pairs (list dari pasangan <termID, docID>) dan menyimpan mereka ke index dengan bantuan InvertedIndexWriter. Sebelum itu, pada *function* write_to_index ini diterapkan konsep BSBI dimana hanya di-maintain satu *dictionary* besar untuk keseluruhan block.

Tahapan pertama dari write_to_index adalah membuat sebuah *dictionary* yang disebut term_dict. *Dictionary* ini akan digunakan untuk melakukan inversion terhadap pasangan termID - docID atau pasangan td_pairs. Setiap entri dalam term_dict ini akan merepresentasikan sebuah termID. Ketika terdapat pasangan termID - docID dalam td_pairs, mereka akan dimasukkan ke dalam term_dict sesuai dengan termID-nya. Contohnya, bila kita memiliki pasangan td_pairs yaitu (term_id, doc_id) sebagai input, kita akan memeriksanya apakah term_id tersebut sudah ada dalam term_dict. Jika belum, kita akan membuat sebuah entri baru dengan term_id sebagai primary key dan sebuah *dictionary* kosong sebagai nilai. *Dictionary* kosong ini nantinya akan digunakan untuk menyimpan informasi kemunculan *term* dalam dokumen yang berbeda. Selanjutnya bila ternyata term_id sudah ada di term_dict, kita akan memeriksa apakah doc_id sudah ada dalam *dictionary* yang terkait dengan term_id tersebut. Jika belum, maka langkah selanjutnya

adalah membuat entri baru untuk doc_id dan menginisialisasinya dengan nilai 0. Maka pada titik ini termID dan docID sudah ada, maka akan memenuhi kondisi terakhir yang akan menambahkan nilai 1 frekuensi kemunculannya. Berikut adalah contoh tahapan penambahan *dictionary*:

```
182         term_dict = {}
183         for term_id, doc_id in td_pairs:
184             if term_id not in term_dict:
185                 term_dict[term_id] = dict()
186             if(term_dict[term_id].get(doc_id) == None):
187                 term_dict[term_id][doc_id] = 0
188             term_dict[term_id][doc_id] += 1
```

Gambar 3. 27 Pembuatan Dictionary pada Function write_to_index

1) Line 182 : Dibuat *dictionary* yang bernama term_dict.
2) Line 184 dan 185 : Mengecek apakah term_id sudah ada pada term_dictionary. Bila tidak ada maka kondisi akan TRUE maka akan ditambahkan entri baru untuk term_id dengan *dictionary* nya yang masih 0, atau seperti berikut:

   term_dict = {
   1: {}
   }

   Dimana pada contoh ini term_id baru adalah 1.
3) Line 186 dan 187 : Selanjutnya karena termnya sudah ada, sekarang kita akan cek apakah doc_id yang bersangkutan dengan term_id tersebut sudah ada atau tidak. Bila belum ada atau == None, maka doc_id akan ditambahkan namun berfrekuensi nilai 0 atau sebagai berikut:

   term_dict = {
   1: {10 : 0}
   }
4) Line 188 : Terakhir, karena telah memenuhi ketentuan sebelumnya, sekarang akan memasuki bahwa term_id sudah ada di *dictionary* dan doc_id yang berelasi dengan term_id juga sudah ada maka frekuensinya akan ditambahkan nilai 1 atau += 1 yang mana sebagai berikut:

term_dict = {
  1: {10 : 1}
}

Selanjutnya, term_id pada term_dict akan diurutkan dan dilakukan perulangan untuk setiap term_id pada term_dict. Pada setiap perulangan, *dictionary* dari term_id akan di urutkan juga berdasarkan doc_id sebagai primary key. Contohnya adalah pada perulangan term_id yang sebelumnya adalah 1 : {10 : 2, 20 : 2, 30 : 3} akan berubah menjadi 1 = [(10, 2), (20, 2), (30, 3)].

Selanjutnya kita akan pisahkan doc_id dan frekuensinya ke dalam dua list terpisah yaitu posting_List dan tf_list. posting_list akan berisi daftar doc_id sementara tf_list akan berisi frekuensi kemunculan *term* pada dokumen terkait. Terakhir, akan menggunakan index untuk append term_id, postings_list, dan tf_list. Berikut adalah bagaimana cara melakukan pengurutan term_id, pengurutan doc_id dan pengambilan item doc_id ke posting_list dan tf ke tf_list.

```
for term_id in sorted(term_dict.keys()):

    sorted_terms = sorted(term_dict[term_id].items(), key=lambda x: x[0])

    postings_list = [item[0] for item in sorted_terms]
    tf_list = [item[1] for item in sorted_terms]

    index.append(term_id, postings_list, tf_list)
```

Gambar 3. 28 Pengurutan doc_id pada Function write_to_index

Karena sorted *term* telah menjadi bentuk 1 = [(10, 2), (20, 2), (30, 3)], untuk mengambil postings_list dan tf_list kita hanya perlu mengambil index 0 untuk postings_list dan index 1 untuk tf_list. Sehingga masing-masing list akan berbentuk:

- postings_list akan menjadi [10, 20, 30]
- tf_list akan menjadi [2, 2, 3]

Terakhir adalah akan melakukan append ke index yang mana index adalah panggilan dari *class* InvertedIndexWriter. Pada saat index append terpanggil dalam function write_to_index, proses penambahan data ke index tersebut melewati beberapa tahapan yang mana dilakukan pada class InvertedIndexWriter. Berikut adalah bagian dari class InvertedIndexWriter.

```
class InvertedIndexWriter(InvertedIndex):
    def __enter__(self):
        self.index_file = open(self.index_file_path, 'wb+')
        return self
```

Gambar 3. 29 Class InvertedIndexWriter

Pertama, postings_list dan tf_list yang merupakan daftar dokumen di mana *term* tersebut muncul beserta frekuensinya, dienkripsi agar dapat disimpan secara efisien dalam file indeks. Selanjutnya, metadata tentang *term* tersebut diperbarui termasuk menambahkan *term* baru ke daftar terms dan perhitungan frekuensi *term* di setiap dokumen yang disimpan dalam doc_length. Metadata ini juga mencakup informasi tentang posisi awal *term* di file indeks dan panjang byte dari encoded_posting_list dan encoded_tf_list. Setelah itu, data yang telah dienkripsi ditulis ke dalam file indeks, dan posisi penulisan dipindahkan ke akhir file untuk menambahkan data baru. Setelah selesai menulis, rata-rata panjang dokumen (avdl) dihitung untuk keperluan analisis relevansi. Terakhir, proses akan keluar dari blok with statement, dan file indeks akan ditutup. Dengan demikian, InvertedIndexWriter berhasil menambahkan term, postings list, dan list TF ke indeks dengan memperbarui metadata dan menulis data terenkripsi ke dalam file indeks.

c) Kesimpulan Alur Indexing

Maka bisa kita rangkum hal yang dilakukan pada alur BSBI atau pengindeksan ini yang diawali dengan penggunaan *function* do_indexing yang merupakan inti dari proses indexing dokumen menggunakan skema BSBI (blocked-sort based indexing). Pertama, *function* ini melakukan pemindaian terhadap setiap sub-direktori dalam folder koleksi dokumen. Setiap sub-direktori

direpresentasikan sebagai satu blok dokumen. Selanjutnya, dokumen-dokumen dalam setiap blok diproses menggunakan *function* parsing_block untuk mengurai teks dan mendapatkan pasangan term-docID. Hasil *parsing* ini kemudian diindeks dengan menggunakan function write_to_index, yang melakukan proses inversion terhadap pasangan termID-docID dan menyimpannya ke dalam indeks sementara. Setelah seluruh blok dokumen diindeks, indeks-indeks sementara tersebut digabungkan menjadi satu indeks utama menggunakan metode merge_index.

Proses *parsing* dokumen dilakukan menggunakan *function* parsing_block, yang membaca konten dari dokumen dalam sebuah sub-direktori dan mengurai teks tersebut menjadi pasangan term-docID. Setiap kata dalam dokumen kemudian diproses menggunakan preprocessing, termasuk tokenisasi, perubahan ke huruf kecil, stemming, dan penghapusan stopwords. Selanjutnya, setiap *term* akan diberikan sebuah ID menggunakan IdMap, yang mengatur pemberian ID kepada *term* dan dokumen. Hasil akhir dari parsing_block adalah daftar pasangan term-docID yang akan digunakan dalam pembuatan indeks.

Function write_to_index bertanggung jawab atas pembuatan indeks dari pasangan term-docID yang telah didapatkan. Proses ini melibatkan pembuatan sebuah *dictionary* yang merepresentasikan term-docID, di mana setiap *term* memiliki entri yang berisi dokumen-dokumen di mana *term* tersebut muncul beserta frekuensinya. Setelah itu, *dictionary* tersebut diurutkan dan hasilnya akan diurutkan berdasarkan termID. Kemudian, data akan dipisahkan menjadi posting list dan *term* frequency list, dan akhirnya akan ditulis ke dalam indeks menggunakan InvertedIndexWriter. Setiap *term* akan dimasukkan ke dalam daftar terms, informasi mengenai frekuensi *term* di setiap dokumen akan diperbarui dalam doc_length, dan data akan disimpan dalam file indeks. Dengan demikian, *function* write_to_index berhasil membangun indeks dengan memperbarui metadata dan menyimpan data terenkripsi ke dalam file indeks.

Keseluruhan proses indexing ini menggabungkan konsep pembagian dan penggabungan indeks sementara untuk menghasilkan satu indeks utama yang

komprehensif dari seluruh koleksi dokumen. Prosesnya melibatkan pembacaan, *parsing*, dan pembalikan dokumen menjadi indeks yang terstruktur. Dengan menggunakan pendekatan BSBI, proses indexing ini mampu menangani koleksi dokumen yang besar secara efisien, sehingga memungkinkan pencarian yang cepat dan relevan terhadap informasi dalam dokumen-dokumen tersebut.

7. Pembuatan alur searching

a) Uraian Alur Searching

1) Home.html

Alur searching dimulai ketika tombol search pada halaman utama yaitu di halaman home.html dipencet oleh user. Selain query yang diberikan oleh user, pencarian juga bisa dilengkapi dengan filter yaitu filter tanggal dan filter satuan kerja. Berikut adalah kode dari search bar input query dan filter user.

```
<!-- SEARCH BAR -->
<!-- 2 -->
<div class="s003">
    <form action="#" onsubmit="redirectToSearch(); return false;" method="get">
        <div class="inner-form">
            <!-- date filter -->
            <div class="input-field fourth-wrap">
                <input class="date_pick" type="date" name="date" placeholder="MM/DD/YY">
            </div>
            <!-- satker filter -->
            <div class="input-field first-wrap">···
            </div>
            <!-- query form -->
            <div class="input-field second-wrap">
                <input type="text" name="query" id="search_bar" placeholder="Enter Keywords?" value="{{query}}"/>
            </div>
            <div class="input-field third-wrap">
                <button class="btn-search" type="submit">···
                </button>
            </div>
        </div>
    </form>
</div>
```

Gambar 3. 30 Input pada File home.html

Ketiga tempat input ini kita satukan menjadi satu form karena ketiganya akan selalu dikirimkan atau akan menjalani proses walaupun ada kemungkinan salah satunya tidak diisi oleh user karena pencarian akan tetap berjalan walaupun input dari user kosong. Form ini memiliki method GET karena sesuai dengan tujuannya yaitu mengirimkan data untuk melakukan

pencarian dan menampilkan hasilnya. Maksudnya adalah kita akan melakukan proses pengambilan dokumen berdasarkan input dari form ini, maka dari itu jenis methodnya adalah GET. Selain itu, atribut onsubmit digunakan untuk mengarahkan proses ke JavaScript yang mana mengatur proses selanjutnya secara lebih detail. JavaScript yang dituju adalah function redirectToSearch.

Function JavaScript yang akan mengirimkan query dan input-input lainnya diberi nama redirectToSearch dan dapat dilihat kodenya pada gambar berikut ini.

```
<script>
    function redirectToSearch() {
        var url = '{% url "search" %}?';
        var dateInput = document.querySelector('input[name="date"]');
        var satkerInput = document.querySelector('select[name="satker"]');
        var queryInput = document.querySelector('input[name="query"]');

        if (dateInput.value !== '') {
            url += 'date=' + encodeURIComponent(dateInput.value) + '&';
        }
        if (satkerInput.value !== '') {
            url += 'satker=' + encodeURIComponent(satkerInput.value) + '&';
        }
        if (queryInput.value !== '') {
            url += 'query=' + encodeURIComponent(queryInput.value);
        }
        window.location.href = url;
    }
</script>
```

Gambar 3. 31 Function redirectToSearch

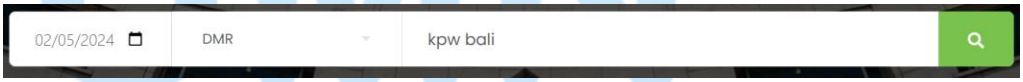
Pertama-tama dibuat atau didefinisikan variabel url yang berisi URL yang dihasilkan oleh template tag Django yaitu {% url "search" %}. URL ini adalah template yang menggabungkan antara urls.py dengan views.py yang seperti kita bahas sebelumnya saling berhubungan dan bekerja sama untuk memberikan pemetaan alur dari sebuah request atau proses URL aplikasi. Jadi tag tersebut akan mengarahkan ke file urls.py tepatnya ke path search yang mana seperti dibawah ini.

```
urlpatterns = [
    path('', home, name="home"),
    path('search', ask, name='search'),
```

Gambar 3. 32 urlPatterns pada File urls.py

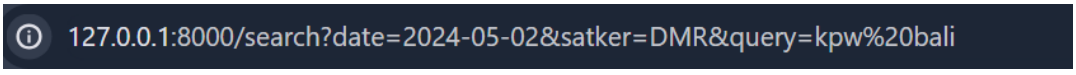
Line diatas bisa kita baca yaitu pada url ‘search’ maka akan diteruskan ke function ‘ask’ pada file views.py. Selanjutnya kita akan membuat variabel untuk menampung input user. Variabel tersebut ialah dateInput yang mereferensikan elemen input tanggal, satkerInput yang mereferensikan elemen select satuan kerja atau satker, dan queryInput yang mereferensikan elemen input teks untuk kata kunci pencarian yang diberikan user.

Selanjutnya dibuat ketentuan ‘if’ yang berfungsi untuk memeriksa apakah ketiga input tidak kosong. Jika tidak kosong, maka akan disatukan url default. Contohnya bila ketiga input diisi oleh user, maka yang akan terjadi adalah url dari variabel url yang kita telah definisikan sebelumnya akan ditambah dengan ‘date’ lalu ‘nilai input date’ lalu huruf sambung ‘&’ karena akan disatukan dengan input selanjutnya yaitu ‘satker’ lalu ‘nilai input satker’. Maka dari itu telah terbuat url yang telah disatukan dengan input yang diberikan user. Berikut adalah contoh hasil url yang diperagakan kedalam halaman search engine.



Gambar 3. 33 Search Bar Input

Pada gambar … kita coba masukkan input tanggal ‘02/05/2024’, input satker ‘DMR’, dan query ‘kpw bali’. Ketika kita klik tombol search, maka akan diarahkan ke Url yang mana dijabarkan sebelumnya dengan hasil seperti dibawah ini.



Gambar 3. 34 Hasil URL Berdasarkan Input

URL ini diawali url default yaitu /search/ dan diteruskan dengan detail input dari user.

2) views.py (ask)

Setelah function redirectToSearch dijalankan dan diarahkan oleh urls.py ke file views.py, function yang dijalankan pada file tersebut adalah function ask(request). Function tersebut dapat dilihat pada gambar berikut ini.

```
def ask(request):
    if request.method == 'GET':
        param  = request.GET.get('query', '')
        param_satker = request.GET.get('satker','')
        param_date = request.GET.get('date','')
        try:
            list_satker = SatuanKerja.objects.filter()
            result = search_service.ask(param, param_satker, param_date)
            # result = Document.objects.all()
            serializer = DocumentSerializer(result, many=True)
            # print(serializer.data)
            context = {"results" : result, "query" : param, "list_satker":list_satker}
            # context = {}
            return render(request, 'search.html', context)
        except Exception as e:
            print(e)
```

Gambar 3. 35 Function ask

Function ini pertama akan mengecek bila request yang diterima adalah dalam bentuk GET. Bila request sudah sesuai, selanjutnya kita buat 3 variabel yang akan menampung input yang akan diambil dari URL sebelumnya. Variabel tersebut ialah param untuk menampung query, param_satker untuk menampung satuan kerja, dan param_date untuk menampung tanggal. Ketiga nilai tersebut di ekstrak dari URL yang telah dibangun sebelumnya.

Selanjutnya dibangun blok Try sebagai penanganan kesalahan. Pertama mengambil semua objek di tabel SatuanKerja dari database, lalu memanggil function ask dari search_service dengan parameter nilai-nilai yang disimpan di variabel sebelumnya yaitu param, param_satker, dan param_date. Selanjutnya dibuat variabel context yang berisi hasil pencarian atau ‘results’, nilai query atau di ‘query’, dan daftar satker atau di

‘list_satker’. Pada context ini sudah menyimpan list dokumen-dokumen yang terpilih dan di ranking dari search_service. Terakhir, nilai-nilai tersebut akan di render ke halaman result page atau search.html.

3) search.html

Halaman search.html adalah halaman yang menampilkan dokumen-dokumen result dari proses pencarian ini. Berikut adalah kode dari halaman search.html yaitu bagian hasil dokumen akan ditampilkan.

```
<div class="output_outer_container">
    {% if results %}
    {% for result in results %}
    <div class="output_holder">
        <div class="output">
            <div class="row">
                <!-- 1 -->
                <div class="e1"> <img class="pdf_icon" src="{% static 'assets/pdf.png' %}" alt="pdf"> </div>
                <!-- 2 -->
                <div class="e2"> <a class="file_name" href="/search_engine/{{ result.file }}" download>{{ result.judul }}</a
            </div>
            <div class="row">
                <!-- 3 -->
                <div class="e3"> <a class="date_name" style="font-size: 2.2vh"> {{result.waktu_rapat}} </a> </div>
                <!-- 4 -->
                <div class="e4"> <a class="satker_name" style="font-size: 2.2vh">| {{result.satker_pengundang.nama}}</a> </d
                <!-- 5 -->
                <div class="e5"> <button class="open_file_btn" onclick="location.href='/search_engine/{{ result.file }}';">Op
                <!-- 6 -->
                <div class="e6">···
                </div>
            </div>
        </div>
    </div>
    {% endfor %}
    {% else %}
```

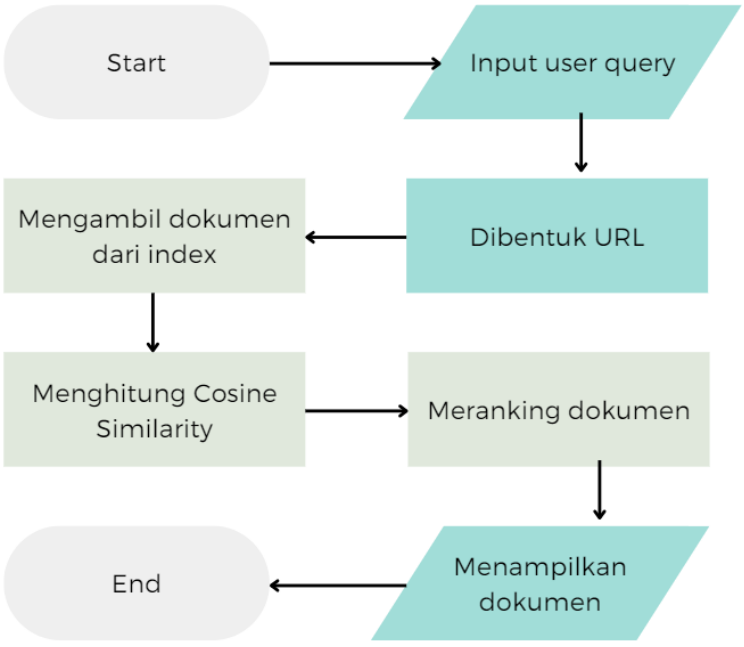
Gambar 3. 36 Menampilkan Result pada Halaman search.html

Diawali dengan penggunaan tag django {% if results %} yang mana memeriksa apakah variabel results memiliki isi atau tidak. Variabel results ini diisi oleh function ask yang didapat pada file views.py sebelumnya. Selanjutnya dilanjutkan dengan pembuatan looping melalui setiap item yang ada di dalam results ini. Setiap results mewakili satu dokumen hasil pencarian. Item-item tersebut akan di breakdown atau ditampilkan di placeholdernya masing-masing. Seperti nama file atau result.judul akan ditampilkan di class file_name, result.waktu_rapat ditampilkan di class date_name, dan result.satker_pengundang.nama ditampilkan di class satker_name.

b) Kesimpulan Alur Searching

Jadi, alur *searching* atau proses pencarian dimulai dari halaman home.html ketika user menekan tombol search setelah mengisi query, date, dan satuan kerja. Disisi back-end, ketiga input ini akan dijalankan dengan menggunakan function redirectToSearch yang mana akan membangun URL berdasarkan input tersebut. URL ini kemudian diarahkan ke path search di urls.py, yang menghubungkan ke fungsi ask di views.py

Fungsi ask di views.py menangani request GET dengan mengekstrak input query, satuan kerja, dan tanggal dari URL. Fungsi ini memanggil layanan pencarian (search_service) untuk mendapatkan hasil pencarian dan menyimpan hasil tersebut dalam variabel context, yang kemudian dirender ke halaman search.html. Di halaman search.html, hasil pencarian ditampilkan menggunakan tag Django, yang menampilkan informasi setiap dokumen yang ditemukan (seperti judul, tanggal, dan satuan kerja) dalam placeholder yang telah ditentukan. Berikut adalah gambaran workflow dari alur searching.



Gambar 3. 37 Workflow alur searching

Kontribusi mahasiswa dalam alur ini ditandai dengan cell berwarna biru diantaranya adalah:

- *Input user query*, mahasiswa membantu dalam menghubungkan input user di home.html
- Membentuk URL, mahasiswa membuat function yang berfungsi untuk mengubah input menjadi bentuk url.
- Menampilkan dokumen dimana mahasiswa mengambil hasil perangkingan dokumen dan menampilkannya kembali di result page atau search.html dengan menggunakan pengulangan.

3.3. Kendala yang Ditemukan

Selama pelaksanaan program kerja magang yang dilakukan sebagai *website developer intern* di Bank Indonesia sudah pastinya terdapat beberapa kendala yang menjadi penghambat suatu pekerjaan ini berjalan. Beberapa kendala yang dihadapi mahasiswa selama pelaksanaan kerja magang adalah sebagai berikut:

1. Kurangnya pengetahuan dan keahlian mahasiswa pada framework yang digunakan ataupun bidang pengembangan yaitu NLP atau Natural Language Processing. Pada pengembangan kali ini yang menggunakan Django menjadi hambatan yang cukup besar pada mahasiswa dikarenakan belum pernah dipaparkan atau dipelajari mahasiswa selama masa perkuliahan. Selain itu, pengaplikasian NLP juga merupakan hal yang asing bagi mahasiswa karena belum pernah mencobanya ataupun mempelajarinya sebelumnya. Terutama untuk pembuatan *search engine* itu sendiri memiliki sumber referensi online yang sangat sedikit sehingga menghambat mahasiswa untuk mempelajari dan mencoba mengaplikasikan pembangunan *search engine* itu terlebih dahulu sebelum proyek dimulai. Kendala yang dihadapi mahasiswa dalam pembuatan beberapa alur utama adalah sebagai berikut:
   - Mahasiswa mengalami kendala dalam pemetaan templates dan pengaturan home.html sehingga terbaca oleh mesin saat server django dijalankan.
   - Pada alur upload, mahasiswa menemukan hambatan di penyesuaian antara tipe data yang ditentukan di home.html dan tipe data yang diteruskan pada upload_service serta tipe data yang sebenarnya diterima di tabel database Django.
   - Pada alur indexing, untuk membuat alur indexing dengan metode Block Sort Indexing mahasiswa belum memahami konsepnya secara menyeluruh. Mahasiswa juga

mendapatkan hambatan ketika ingin memanggil function preprocessing untuk penggunaannya di function parsing_block.

2. Kurangnya kesesuaian timeline dan kemajuan progress proyek. proyek *search engine* yang mahasiswa aktif berjalan di bulan Maret 2024 disaat seluruh tim pengembang sudah siap untuk mengerjakan proyek. Hal ini menyisakan waktu pengerjaan kurang dari 3 bulan untuk menyelesaikan proyek ke tahap prototype.

3.4. Solusi atas Kendala yang Ditemukan

Bila dibiarkan secara terus menerus, maka akan dapat mendapatkan dampak negatif bagi perusahaan maupun mahasiswa yang menjalankan praktek kerja nyata ini. Maka dari itu, diperlukannya sebuah solusi agar dapat mengatasi kendala tersebut sehingga mahasiswa dapat menjalani praktek kerja nyata dengan efektif dan dapat memberikan kinerja yang memuaskan bagi proyek ataupun perusahaan. Berikut adalah solusi yang dilakukan dari kendala yang dihadapi mahasiswa selama melakukan kerja magang di perusahaan:

1. Pembuatan *search engine* terutama dengan menggunakan vector space model yang mana memiliki minim referensi dan pembahasan secara online menjadi tantangan besar bagi mahasiswa dalam menjalani program kerja nyata nya. Namun, untuk mengurangi hambatan tersebut, mahasiswa tetap mencari tahu dan mencoba memahami konsep dari *search engine* ataupun information retrieval secara umum. Mahasiswa juga mencoba mencari tahu dari berbagai sumber seperti youtube, coursera, hingga github. Hal ini membantu mahasiswa memahami konsep dasar dari information retrieval hinggal konsep dasar dari pengaplikasian vector space model. Dalam mengatasi kendala pada pengembangan *search engine* di beberapa alur, mahasiswa mendapatkan bantuan dan solusi sebagai berikut:
   - Mahasiswa mendapat bantuan dari supervisor dan rekan magang dalam pengaturan pemetaan proyek ini yang menggunakan framework Django.
   - Pada kendala ketidaksesuaian tipe data saat melakukan *parsing* dari file ke database Django, mahasiswa mendapatkan bantuan dari video tutorial youtube dan menyusun ulang file python secara perlahan dan bertahap sehingga menemukan titik kesalahannya.

- Pada alur indexing, karena mahasiswa belum memahami metode Block Sort Indexing, mahasiswa mendapatkan bantuan dari rekan magang yang pernah melakukan BSBI dan dapat mengajarkan konsep tersebut kepada mahasiswa.

2. Waktu pengerjaan yang cukup singkat mengharuskan mahasiswa untuk beradaptasi dengan cepat dalam mengerjakan proyek secara nyata ini. Mahasiswa dapat mengatasi hambatan ini dengan memperbanyak komunikasi dalam tim, baik itu dengan supervisor atau antara anggota. Mahasiswa mengutamakan komunikasi agar dapat menyelesaikan proyek dengan baik di waktu yang sempit ini.