4. IMPLEMENTASI SISTEM

Pada bab ini, akan dibahas implementasi sistem yang dilakukan berdasarkan desain sistem yang dirancang pada bab sebelumnya. Hubungan antara desain sistem dan segmen program dapat dilihat pada Tabel 4.1.

Tabel 4.1

Hubungan Segmen Program dan Desain Sistem

Segmen Program	**Referensi**	**Keterangan**
Segmen Program 4.1	-	Pembuatan Struktur Graph
Segmen Program 4.2	-	Convert Data ke RDF Statements
Segmen Program 4.3	3.4.3.1	Preprocessing Data
Segmen Program 4.4	3.4.3.2	Keyword Extraction
Segmen Program 4.5	3.4.3.3	Pengambilan Data di Wikidata
Segmen Program 4.6	3.4.3.4	Jaccard Similarity
Segmen Program 4.7	3.4.3.5	Euclidean Distance
Segmen Program 4.8	3.4.3.6	Cosine Similarity
Segmen Program 4.9	-	Metrik Evaluasi
Segmen Program 4.10	-	Koneksi GraphDB
Segmen Program 4.11	3.4.3.7	Pembuatan Database Vektor
Segmen Program 4.12	3.4.3.7	Inisialisasi Vektor
Segmen Program 4.13	3.4.3.7	Proses Search
Segmen Program 4.14	3.5.1	Pembuatan Halaman Home
Segmen Program 4.15	3.5.2	Pembuatan Halaman Hasil Search
Segmen Program 4.16	3.5.3	Pembuatan Halaman Detail Koleksi

4.1 Implementasi Database

Implementasi database melibatkan pembentukan struktur data yang dapat mengakomodasi kebutuhan sistem berdasarkan model *graph* yang telah dirancang.

4.1.1 Pembuatan Struktur Graph

Pembuatan struktur *graph* dilakukan dengan mendefinisikan *class*, data *properties*, dan *object properties* sesuai dengan skema yang telah dijelaskan di bagian 3.3.2. Struktur *graph* ini dibuat menggunakan *software* Protégé. Hasil dari Protégé berupa *file* dengan format *turtle* yang dapat dilihat pada Segmen Program 4.1.

Segmen Program 4.1 Pembuatan Struktur *Graph*

```
@prefix lib: <https://dewey.petra.ac.id/data/> .
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <https://dewey.petra.ac.id/data/> .

<https://dewey.petra.ac.id/data/> rdf:type owl:Ontology .

# Classes
lib:Collection rdf:type owl:Class .

lib:DigitalCollection rdf:type owl:Class ;
                      rdfs:subClassOf lib:Collection .

lib:PhysicalCollection rdf:type owl:Class ;
                       rdfs:subClassOf lib:Collection .

lib:SerialCollection rdf:type owl:Class ;
                     rdfs:subClassOf lib:Collection .

lib:Creator rdf:type owl:Class .

lib:Contributor rdf:type owl:Class .

lib:Category rdf:type owl:Class .

lib:Subcategory rdf:type owl:Class .

lib:Theme rdf:type owl:Class .

lib:Subject rdf:type owl:Class .

lib:Keyword rdf:type owl:Class .

lib:WikidataKeyword rdf:type owl:Class .

# Object Properties
lib:contributedBy rdf:type owl:ObjectProperty ;
                  rdfs:domain lib:Collection ;
```

```
                rdfs:range lib:Contributor .

lib:createdBy rdf:type owl:ObjectProperty ;
              rdfs:domain lib:Collection ;
              rdfs:range lib:Creator .

lib:hasCategory rdf:type owl:ObjectProperty ;
                rdfs:domain lib:Collection ;
                rdfs:range lib:Category .

lib:hasKeyword rdf:type owl:ObjectProperty ;
               rdfs:domain lib:Collection ;
               rdfs:range lib:Keyword .

lib:hasSubcategory rdf:type owl:ObjectProperty ;
                   rdfs:domain lib:Collection ;
                   rdfs:range lib:Subcategory .

lib:hasSubject rdf:type owl:ObjectProperty ;
               rdfs:domain lib:Collection ;
               rdfs:range lib:Subject .

lib:hasTheme rdf:type owl:ObjectProperty ;
             rdfs:domain lib:Collection ;
             rdfs:range lib:Theme .

lib:relatedTo rdf:type owl:ObjectProperty ;
              rdfs:domain lib:Keyword ;
              rdfs:range lib:WikidataKeyword .

# Data Properties
lib:id rdf:type owl:DatatypeProperty ;
       rdfs:domain lib:Collection ,
                   lib:Contributor ,
                   lib:Creator ,
                   lib:WikidataKeyword ;
       rdfs:range xsd:string .

lib:title rdf:type owl:DatatypeProperty ;
          rdfs:domain lib:Collection ;
          rdfs:range xsd:string .

lib:description rdf:type owl:DatatypeProperty ;
                rdfs:domain lib:Collection ;
                rdfs:range xsd:string .

lib:name rdf:type owl:DatatypeProperty ;
         rdfs:domain lib:Contributor ,
                     lib:Creator ;
         rdfs:range xsd:string .
```

4.1.2 *Convert* Data ke RDF *Statements*

Setelah struktur ontologi dibuat menggunakan Protégé, dilakukan pengisian data dengan cara melakukan convert data dari bentuk table ke RDF *statements*. Data yang akan dimasukkan mencakup entitas seperti koleksi, *creators*, *contributors*, *theme*, *category*, *subcategory*, *subject*, *keywords*, dan properti terkait. Proses ini dilakukan dengan membaca data dari dataset, memproses setiap baris, dan menambahkan data tersebut ke dalam *graph* RDF. Setiap entitas direpresentasikan sebagai URI yang unik, dan relasi antar entitas ditambahkan berdasarkan skema ontologi yang telah dirancang.

Segmen Program 4.2 Convert Data ke RDF Statements

```
def clean_words(words):
    words = re.sub(r'[^\w\s]', '', words)
    words = words.replace(' ', '_')
    return words

# Define namespaces
LIB = Namespace("https://dewey.petra.ac.id/data/")
WD = Namespace("http://www.wikidata.org/entity/")
WDT = Namespace("http://www.wikidata.org/prop/direct/")

# Initialize RDF graph
g = Graph()
g.bind("lib", LIB)
g.bind("wd", WD)
g.bind("wdt", WDT)

# Function to add row data to graph
def insert_row(row):
    id = row['id']
    title = row['title'] if pd.notna(row['title']) and
row['title'] != "-" else "Unknown Title"
    description = row['description'] if
pd.notna(row['description']) and row['description'] != "-" else
""
    creators = row['creators'] if pd.notna(row['creators']) and
row['creators'] != "-" else ""
```

```
    creator_ids = row['creator_ids'] if
pd.notna(row['creator_ids']) and row['creator_ids'] != "-" else
""
    contributors = row['contributors'] if
pd.notna(row['contributors']) and row['contributors'] != "-"
else ""
    contributor_ids = row['contributor_ids'] if
pd.notna(row['contributor_ids']) and row['contributor_ids'] !=
"-" else ""
    contributor_roles = row['contributor_roles'] if
pd.notna(row['contributor_roles']) and
row['contributor_roles'] != "-" else ""
    themes = row['themes'] if pd.notna(row['themes']) and
row['themes'] != "-" else ""
    category = row['category'] if pd.notna(row['category']) and
row['category'] != "-" else ""
    subcategory = row['subcategory'] if
pd.notna(row['subcategory']) and row['subcategory'] != "-" else
""
    subjects = row['subjects'] if pd.notna(row['subjects']) and
row['subjects'] != "-" else ""
    keywords = row['keywords'] if pd.notna(row['keywords']) and
row['keywords'] != "-" else ""
    properties = row['properties'] if
pd.notna(row['properties']) and row['properties'] != "-" else ""
    property_ids = row['property_ids'] if
pd.notna(row['property_ids']) and row['property_ids'] != "-"
else ""
    related_keywords = row['related_keywords'] if
pd.notna(row['related_keywords']) and row['related_keywords'] !=
"-" else ""
    related_keyword_ids = row['related_keyword_ids'] if
pd.notna(row['related_keyword_ids']) and
row['related_keyword_ids'] != "-" else ""
```

```
    creator_list = [name.strip() for name in creators.split('; ')] if isinstance(creators, str) and creators else []
    creatorID_list = [id.strip() for id in creator_ids.split('; ')] if isinstance(creator_ids, str) and creator_ids else []
    contributor_list = [name.strip() for name in contributors.split('; ')] if isinstance(contributors, str) and contributors else []
    contributorID_list = [id.strip() for id in contributor_ids.split('; ')] if isinstance(contributor_ids, str) and contributor_ids else []
    contributorRole_list = [role.strip() for role in contributor_roles.split('; ')] if isinstance(contributor_roles, str) and contributor_roles else []
    subject_list = [subject.strip() for subject in subjects.split('; ')] if isinstance(subjects, str) and subjects else []
    keywords_list = [keyword.strip() for keyword in keywords.split('; ')] if isinstance(keywords, str) and keywords else []
    all_properties_list = [properties.strip() for properties in properties.split('//')] if isinstance(properties, str) and properties else []
    all_propertyIDs_list = [property_ids.strip() for property_ids in property_ids.split('//')] if isinstance(property_ids, str) and property_ids else []
    all_related_keywords_list = [related_keywords.strip() for related_keywords in related_keywords.split('//')] if isinstance(related_keywords, str) and related_keywords else []
    all_related_keywordsID_list = [related_keyword_ids.strip() for related_keyword_ids in related_keyword_ids.split('//')] if isinstance(related_keyword_ids, str) and related_keyword_ids else []

    collection = URIRef(LIB[f"Collection/{id}"])

    if themes:
```

```
        theme_uri = URIRef(LIB[f"Theme/{clean_words(themes)}"])
        g.add((collection, LIB.hasTheme, theme_uri))
        g.add((theme_uri, RDF.type, LIB.Theme))
        g.add((theme_uri, RDFS.label, Literal(themes)))

    if category:
        category_uri =
URIRef(LIB[f"Category/{clean_words(category)}"])
        g.add((collection, LIB.hasCategory, category_uri))
        g.add((category_uri, RDF.type, LIB.Category))
        g.add((category_uri, RDFS.label, Literal(category)))

    if subcategory:
        subcategory_uri =
URIRef(LIB[f"Subcategory/{clean_words(subcategory)}"])
        g.add((collection, LIB.hasSubcategory, subcategory_uri))
        g.add((subcategory_uri, RDF.type, LIB.Subcategory))
        g.add((subcategory_uri, RDFS.label,
Literal(subcategory)))

    if any(keywords.strip() for keywords in keywords_list):
        for index, keyword in enumerate(keywords_list):
            keyword_uri =
URIRef(LIB[f"Keyword/{clean_words(keyword)}"])
            g.add((collection, LIB.hasKeyword, keyword_uri))
            g.add((keyword_uri, RDF.type, LIB.Keyword))
            g.add((keyword_uri, RDFS.label, Literal(keyword)))
            if index < len(all_properties_list):
                col_properties = all_properties_list[index]
                col_propertiesIDs = all_propertyIDs_list[index]
                col_related_keywords =
all_related_keywords_list[index]
                col_related_keyword_ids =
all_related_keywordsID_list[index]
```

```
                if col_properties and col_propertiesIDs and
col_related_keywords and col_related_keyword_ids:
                    property = [col_properties.strip() for
col_properties in col_properties.split(';')]
                    propertyID = [col_propertiesIDs.strip() for
col_propertiesIDs in col_propertiesIDs.split(';')]
                    related_keyword =
[col_related_keywords.strip() for col_related_keywords in
col_related_keywords.split(';')]
                    related_keywordID =
[col_related_keyword_ids.strip() for col_related_keyword_ids in
col_related_keyword_ids.split(';')]

                    for prop, prop_id, kw, kw_id in
zip(property, propertyID, related_keyword, related_keywordID):
                        if prop and prop_id and kw and kw_id:
                            g.add((WDT[prop_id], RDF.type,
OWL.ObjectProperty))
                            g.add((WDT[prop_id],
RDFS.subPropertyOf, LIB["relatedTo"]))
                            g.add((WDT[prop_id], RDFS.label,
Literal(prop)))

                            splitted_kw = kw.split('||')
                            splitted_kw_id = kw_id.split('||')

                            for related_kw, related_kw_id in
zip(splitted_kw, splitted_kw_id):
                                if related_kw and related_kw_id:
                                    is_not_numeric = not
related_kw.isnumeric()
                                    is_not_symbol = not
bool(re.fullmatch(r'[^\w\s]+', input_string, re.UNICODE))
                                    contains_mandarin =
bool(re.search(r'[\u4e00-\u9fff]', related_kw))
                                    if contains_mandarin:
```

```
                                        translated =
ts.translate_text(related_kw, translator='bing',
from_language='auto')
                                        related_kw = translated
                                    if is_not_numeric and
is_not_symbol:
                                        wikidata_keyword_uri =
URIRef(WD[related_kw_id])
                                        g.add((wikidata_keyword_
uri, RDF.type, LIB.WikidataKeyword))
                                        g.add((wikidata_keyword_
uri, RDFS.label, Literal(related_kw)))
                                        g.add((wikidata_keyword_
uri, LIB.id, Literal(related_kw_id)))
                                        g.add((keyword_uri,
WDT[prop_id], URIRef(WD[related_kw_id])))

    if any(creator.strip() for creator in creator_list):
        for creator_name, creator_id in zip(creator_list,
creatorID_list):
            creator_uri =
URIRef(LIB[f"Creator/{clean_words(creator_name)}_{clean_words(cr
eator_id)}"])
            g.add((collection, LIB.createdBy, creator_uri))
            g.add((creator_uri, RDF.type, LIB.Creator))
            g.add((creator_uri, LIB.name,
Literal(creator_name)))
            g.add((creator_uri, LIB.id, Literal(creator_id)))

    if any(contributor.strip() for contributor in
contributor_list):
        for contributor_name, contributor_id, contributor_role
in zip(contributor_list, contributorID_list,
contributorRole_list):
            contributor_uri =
URIRef(LIB[f"Contributor/{clean_words(contributor_name)}"])
```

```
            contributor_role_uri =
URIRef(LIB[f"{clean_words(contributor_role)}"])

            g.add((contributor_role_uri, RDF.type,
OWL.ObjectProperty))
            g.add((contributor_role_uri, RDFS.subPropertyOf,
LIB["contributedBy"]))
            g.add((contributor_role_uri, RDFS.label,
Literal(contributor_role)))

            g.add((collection,
LIB[clean_words(contributor_role)], contributor_uri))

            g.add((contributor_uri, RDF.type, LIB.Contributor))
            g.add((contributor_uri, LIB.name,
Literal(contributor_name)))
            g.add((contributor_uri, LIB.id,
Literal(contributor_id)))

    g.add((collection, RDF.type, LIB.DigitalCollection))
    g.add((collection, LIB.id, Literal(id)))
    g.add((collection, LIB.title, Literal(title)))
    g.add((collection, LIB.description, Literal(description)))

    if any(subject.strip() for subject in subject_list):
        for sub in subject_list:
            subject_uri =
URIRef(LIB[f"Subject/{clean_words(sub)}"])
            g.add((collection, LIB.hasSubject, subject_uri))
            g.add((subject_uri, RDF.type, LIB.Subject))
            g.add((subject_uri, RDFS.label, Literal(sub)))

for index, row in data.iterrows():
    insert_row(row)

# Serialize the graph to Turtle format
```

```
data_serialized = g.serialize(format='turtle')
data_serialized = '\n'.join(line for line in
data_serialized.splitlines() if not line.startswith('@prefix'))

with open("koleksi_digital_s1.ttl", "a", encoding="utf-8") as
file:
    file.write(data_serialized)
```

Setelah data berhasil di-*convert* dan ditambahkan ke dalam ontologi, *file* Turtle yang dihasilkan akan diimport ke dalam *database* GraphDB. Proses import dilakukan dengan membuka halaman admin GraphDB, membuat *repository*, dan mengunggah file Turtle melalui menu *Import* dengan memilih opsi *File Upload*.

4.2 Implementasi *Preprocessing* Data

Pada subbab ini, dibahas tahapan-tahapan yang dilakukan untuk mempersiapkan data agar berada dalam kondisi bersih, terstruktur, dan siap digunakan dalam proses selanjutnya. Tahapan ini mencakup *preprocessing* data, *keyword extraction* untuk mencari frasa-frasa penting, serta penambahan *keyword* dari Wikidata.

4.2.1 *Preprocessing* Data

Sesuai dengan bagian 3.4.3.2, proses *preprocessing* data mencakup penghapusan data yang tidak memiliki ID, penyesuaian tipe data, penyesuaian format untuk masing-masing kolom, pemecahan kolom *creators* dan contributors, dan penghapusan data duplikat. Detail proses *preprocessing data* dapat dilihat pada Segmen Program 4.3.

Segmen Program 4.3 *Preprocessing* Data

```
# Membaca data dari file Excel
data = pd.read_excel('all_data_raw.xlsx')

# Menghapus baris yang tidak memiliki ID
data = data[data['Col ID'].notna()]

# Mengubah tipe data 'Col ID' menjadi integer dan 'Title'
menjadi string
data['Col ID'] = data['Col ID'].astype(int)
data['Title'] = data['Title'].astype(str)

# Mengganti tanda kutip tidak konsisten pada judul dengan tanda
kutip standar
```

```
data['Title'] = data['Title'].replace({'“': '"', '”': '"', '’’':
'"', "'": '"', '""': '"'}, regex=True)

# Menghapus koleksi yang judulnya mengandung kata-kata tertentu
data = data[~data['Title'].str.contains('diterima
sementara|mohon dihapus|akan diupdate|delete|blocked|block
until|embargo|confidential', case=False, na=False)]

# Memperbaiki penulisan tanda kutip pada judul
data['Title'] = data['Title'].apply(
    lambda text: re.sub(r"(\s')", r' "', text) if
text.count('"') == 1 else text  # Jika sebelum ' ada spasi,
ganti dengan "
).apply(
    lambda text: re.sub(r"('\s)", r'" ', text) if
text.count('"') == 1 else text  # Jika setelah ' ada spasi,
ganti dengan "
).apply(
    lambda text: text.replace('"', '') if text.count('"') == 1
and not re.search(r'[‘’]', text) and not re.search(r'\d"', text)
else text  # Jika ada satu tanda ", tanpa ‘ atau ’, dan bukan
angka sebelum "
).apply(
    lambda text: re.sub(r'\s"\s', ' ', text) if text.count('"')
== 1 else text  # Jika sebelum dan sesudah tanda " ada spasi,
hapus tanda petik
).apply(
    lambda text: text.replace('"', '') if re.search(r'[‘’]',
text) and text.count('"') == 1 else text  # Jika ada simbol
‘ atau ’, hapus tanda petik
)

# Menghapus tanda petik di awal dan akhir string
data['Title'] = data['Title'].apply(lambda text: (
    text[1:-1] if text.count('"') == 2 and text.startswith('"')
and text.endswith('"') else text
))

data['Title'] = data['Title'].apply(
    lambda text: (
        re.sub(r'"\s*([a-zA-Z])', lambda match:
f'"{match.group(1)}',
               (text.replace('"', '', 1) if text.count('"') == 3
and text.startswith('"') else text)
              )
        if text.count('"') == 3 and text.startswith('"')
        else text
    )
)

data['Title'] = data['Title'].apply(lambda text: (
    re.sub(r'\s+', ' ',
           re.sub(r'"(.*?)"',
```

```
                    lambda match: f'"{match.group(1).strip()}"',
text)
            ) if text.count('"') % 2 == 0 else text
))

# Menghapus tanda kurung di awal dan akhir string
data['Title'] = data['Title'].str.replace(r'^\((.*?)\)$', r'\1',
regex=True)

# Menghapus tanda tanya di awal dan di akhir string
data['Title'] = data['Title'].apply(lambda text: text.strip('?')
if text.startswith('?') and text.endswith('?') else text)

# Menghapus tanda ` di awal string
data['Title'] = data['Title'].apply(lambda text:
text.lstrip('`') if text.startswith('`') else text)

# Menghapus spasi berlebih
data['Title'] = data['Title'].str.strip()

# Menambahkan role "Unknown" untuk contributor yang tidak
memiliki role
exclude_values = [
    'PANTI ASUHAN DON BOSCO',
    'PUSAT KONSELING DAN PENGEMBANGAN PRIBADI',
    'PUSAT KARIR PETRA',
    'MANAJEMEN PERHOTELAN UK PETRA',
    'PUSAT KEROHANIAN UK PETRA',
    'PERPUSTAKAAN UK PETRA',
    'PETRA LITTLE THEATRE'
]
data['Contributors'] = data['Contributors'].apply(
    lambda x: '; '.join([contrib if '-' in contrib or contrib in
exclude_values else contrib + ' - Unknown'
                         for contrib in x.split(';')])
)
data['Contributors'] = data['Contributors'].str.replace(r' -
\s*$', ' - Unknown', regex=True)

# Mengurutkan creators berdasarkan nama (setelah ID)
data['Creators'] = data['Creators'].apply(
    lambda x: "; ".join(sorted([creator.strip() for creator in
x.split(';')], key=lambda c: c.split(')')[1].strip())) if ';' in
x else x
)

# Mengurutkan nama contributors
data['Contributors'] = data['Contributors'].apply(
    lambda x: "; ".join(sorted([contributor.strip() for
contributor in x.split(';')])) if ';' in x else x
)

# Mengurutkan subjects
```

```
data['Subjects'] = data['Subjects'].apply(
    lambda x: "; ".join(sorted([subject.strip() for subject in
x.split(';')])) if ';' in x else x
)
data['Subjects'] = data['Subjects'].str.title()

data['Cleaned_Title'] = data['Title'].apply(lambda x:
re.sub(r'\W+', '', str(x)).lower())
data['Cleaned_Description'] = data['Description'].apply(lambda
x: re.sub(r'\W+', '', str(x)).lower())
grouped = data.groupby(['Cleaned_Title', 'Cleaned_Description',
'Category'])

# Duplicate handling
final_results = []
for name, group in grouped:
    if len(group) > 1:
        to_drop = set()
        for i, creators_1 in enumerate(group['Creators']):
            for j, creators_2 in enumerate(group['Creators']):
                if i != j and creators_1 in creators_2:
                    to_drop.add(group.index[i])

        final_results.append(group.drop(index=to_drop))
    else:
        final_results.append(group)
final_data = pd.concat(final_results)
data = final_data.reset_index(drop=True)

# Memecah nama creator dan ID creator
pattern = r'\((.*?)\)\s*([^;]+)'
extracted = data['Creators'].apply(lambda x: re.findall(pattern,
x))
data['Creator_IDs'] = extracted.apply(lambda x: ";
".join([match[0] for match in x]))
data['Creators'] = extracted.apply(lambda x: ";
".join([match[1].strip().title() for match in x]))

# Memecah nama contributor, ID contributor, dan role contributor
filtered_data =
data[data['Contributors'].str.contains(r'Supervisor',
case=False, na=False)]
filtered_data['Contributor_IDs'] =
filtered_data['Contributors'].str.findall(r'\(([^)]+)\)').apply(
lambda x: '; '.join(x))
filtered_data['Contributor_Names'] =
filtered_data['Contributors'].str.replace(r'\([^)]*\)', '',
regex=True).str.strip().str.split(';').apply(
    lambda x: '; '.join(
        [item.split('-', 1)[0].strip().title() if '-' in item
else item.strip().title() for item in x]
    )
)
```

```
filtered_data['Contributor_Roles'] =
filtered_data['Contributors'].str.replace(r'\([^)]*\)', '',
regex=True).str.strip().str.split(';').apply(
    lambda x: '; '.join(
        [item.split('-', 1)[1].strip() if '-' in item else
'Unknown' for item in x]
    )
)
filtered_data2 =
data[~data['Contributors'].str.contains(r'Supervisor',
case=False, na=False)]
filtered_data2['Contributor_IDs'] =
filtered_data2['Contributors'].str.findall(r'\(([^)]+)\)').apply
(lambda x: '; '.join(x))

filtered_data2['Contributor_Names'] =
filtered_data2['Contributors'].str.replace(r'\([^)]*\)', '',
regex=True).str.strip().str.split(';').apply(
    lambda x: '; '.join(
        [item.rsplit('-', 1)[0].strip().title() if '-' in item
else item.strip().title() for item in x]
    )
)
filtered_data2['Contributor_Roles'] =
filtered_data2['Contributors'].str.replace(r'\([^)]*\)', '',
regex=True).str.strip().str.split(';').apply(
    lambda x: '; '.join(
        [item.rsplit('-', 1)[-1].strip() if '-' in item else
'Unknown' for item in x]
    )
)
data = pd.concat([filtered_data, filtered_data2],
ignore_index=True)
data.drop(columns=['Contributors'], inplace=True)
data.rename(columns={'Contributor_Names': 'Contributors'},
inplace=True)

# Menghapus duplikat
duplicates =
data[data.duplicated(subset=data.columns.difference(['Col ID']),
keep=False)]
idx_to_keep =
duplicates.groupby(duplicates.columns.difference(['Col
ID']).tolist())['Col ID'].idxmax()
duplicates_to_remove =
duplicates[~duplicates.index.isin(idx_to_keep)]
data = data[~data['Col ID'].isin(duplicates_to_remove['Col
ID'])]

data['Cleaned_Title'] = data['Title'].apply(lambda x:
re.sub(r'\W+', '', str(x)).lower())
data['Cleaned_Description'] = data['Description'].apply(lambda
x: re.sub(r'\W+', '', str(x)).lower())
```

```
# Menghapus duplikat berdasarkan kolom Title, Description,
Creators dan Category
duplicate_mask = data.duplicated(subset=['Cleaned_Title',
'Cleaned_Description', 'Creators', 'Category'], keep=False)
idx = data[duplicate_mask].groupby(['Cleaned_Title', 'Creators',
'Category'])['Col ID'].idxmax()
data = pd.concat([data.loc[idx], data[~duplicate_mask]])
```

4.2.2 Keyword Extraction

Keyword extraction dilakukan sesuai seperti bagian 3.4.3.3, dengan memanfaatkan model NLP berbasis *SpaCy* yang dilengkapi dengan *TextRank* untuk mengidentifikasi maksimal sepuluh frasa atau kata yang mewakili isi koleksi. Teks yang digunakan adalah judul dan deskripsi koleksi yang digabungkan dan diterjemahkan ke dalam Bahasa Inggris.

Segmen Program 4.4 *Keyword Extraction*

```
# Memuat model NLP SpaCy dengan pipeline bahasa inggris
nlp = spacy.load("en_core_web_sm")
# Menambahkan pipeline TextRank ke dalam model SpaCy untuk
ekstraksi keyword
nlp.add_pipe("textrank")

for index, row in data.iterrows():
    # Menggabungkan teks judul dan deskripsi
    text = str(row['Title']) + '. ' + str(row['Description'])
    # Translate teks ke bahasa inggris
        text = ts.translate_text(text, translator='google',
from_language='auto')
    # Memproses teks menggunakan model NLP SpaCy
    doc = nlp(text)

    unique_phrases = set()

    # Mengambil 10 frasa teratas berdasarkan peringkat TextRank
    for phrase in doc._.phrases[:10]:
          # Memfilter frasa untuk memastikan hanya frasa yang
mengandung karakter alfanumerik dan bukan angka murni yang
dimasukkan
```

```
        if any(char.isalnum() for char in phrase.text) and not
phrase.text.isdigit():
            unique_phrases.add(phrase.text)
    data.at[index, 'keywords'] = "; ".join(unique_phrases) if
unique_phrases else ""
```

4.2.3 Pengambilan Data di Wikidata

Proses pengambilan data di Wikidata dilakukan untuk mencari konsep yang terkait dengan setiap *keyword* yang didapat dari proses *keyword extraction*. Proses ini merupakan alasan mengapa teks yang dilakukan *keyword extraction* diterjemahkan ke Bahasa Inggris terlebih dahulu, karena Wikidata memiliki data dalam bahasa Inggris sebesar 11.04%, sedangkan data dalam Bahasa Indonesia kurang dari 1%.

Fungsi search_wikidata_entities ini bertujuan untuk mencari ID entitas di Wikidata berdasarkan kata kunci yang diinputkan. Jika tidak ditemukan, maka dilakukan *fallback* yaitu dengan memotong satu kata paling depan dari kata kunci secara bertahap dan mencoba pencarian ulang hingga entitas yang sesuai ditemukan atau hanya ada satu kata yang tersisa. Fungsi batch_get_labels ini digunakan untuk mengambil label item berdasarkan ID yang diinputkan. Label yang sudah pernah diambil sebelumnya disimpan dalam label_cache untuk menghindari pengambilan data yang sama berulang kali. Fungsi get_related_concepts digunakan untuk mengambil konsep terkait beserta *property* atau hubungannya berdasarkan *keyword* yang diinputkan. Setiap ID dan label konsep serta *property* akan disimpan. Untuk setiap baris data, dilakukan pemanggilan fungsi get_related_concepts dengan menginputkan kata kunci yang dihasilkan dari proses *keyword extraction*, yang akan mengembalikan properti, ID properti, kata kunci terkait, dan ID kata kunci terkait. Proses ini memastikan setiap koleksi memiliki metadata yang lebih lengkap untuk mendukung sistem pencarian.

Segmen Program 4.5 Pengambilan Data di Wikidata

```
# Fungsi untuk mencari ID entitas di Wikidata berdasarkan kata
kunci
def search_wikidata_entities(keyword):
    url = "https://www.wikidata.org/w/api.php"
    params = {
        'action': 'wbsearchentities',
        'format': 'json',
```

```
        'language': 'en',
        'search': keyword
    }

    response = requests.get(url, params=params)
    if response.status_code == 200:
        results = response.json().get('search', [])
        if results:
            return results[0]['id']

    # Jika pencarian tidak ada, coba lagi dengan memotong kata
kunci
    words = keyword.split()
    while len(words) > 1:
        words.pop(0)
        current_search = " ".join(words)
        params['search'] = current_search
        response = requests.get(url, params=params)
        if response.status_code == 200:
            results = response.json().get('search', [])
            if results:
                return results[0]['id']
    return None

#  Cache  untuk  menyimpan  label  entitas  yang  sudah  diambil
sebelumnya
label_cache = {}

# Fungsi untuk mengambil label dari ID entitas secara batch
def batch_get_labels(entity_ids):
    # Memfilter ID yang belum ada di cache
     uncached_ids = [eid for eid in entity_ids if eid not in
label_cache]

    if not uncached_ids:
        return {eid: label_cache[eid] for eid in entity_ids}
```

```
    url = "https://www.wikidata.org/w/api.php"
    params = {
        'action': 'wbgetentities',
        'format': 'json',
        'ids': '|'.join(uncached_ids),
        'props': 'labels',
        'languages': 'en'
    }
    response = requests.get(url, params=params)
    if response.status_code == 200:
        entities = response.json().get('entities', {})
        for eid, entity in entities.items():
                    label  =  entity.get('labels',  {}).get('en',
{}).get('value', None)
            if label:
                            sanitized_label  =  label.replace(';',
'').replace('//', '').replace('||', '')
                label_cache[eid] = sanitized_label

    return {eid: label_cache.get(eid, None) for eid in entity_ids}

# Fungsi untuk mengambil konsep terkait berdasarkan keyword
input
def get_related_concepts(input):
    qid = search_wikidata_entities(input)
    if not qid:
        return None

    url =
f"https://www.wikidata.org/wiki/Special:EntityData/{qid}.json"
    response = requests.get(url)

    # Inisialisasi daftar untuk menyimpan properti dan konsep
terkait
    Properties = []
```

```
    Property_IDs = []
    Related_Keyword_IDs = []
    Related_Keywords = []

    if response.status_code == 200:
        entity_data = response.json().get('entities',
{}).get(qid, {})
        claims = entity_data.get('claims', {})

        for prop_id, prop_values in claims.items():
            prop_label =
batch_get_labels([prop_id]).get(prop_id)
            if not prop_label:
                continue

            entity_ids = []
            entity_count = 0
            valid_labels = []
            valid_entity_ids = []

            # Ambil entitas terkait untuk setiap properti
(maksimal 5 entitas)
            for value in prop_values:
                if entity_count >= 5:
                    break

                mainsnak = value.get('mainsnak', {})
                if mainsnak.get('datatype') == 'wikibase-item':
                    entity_id = mainsnak.get('datavalue',
{}).get('value', {}).get('id', None)
                    if entity_id:
                        entity_ids.append(entity_id)
                        entity_count += 1

            fetched_labels = batch_get_labels(entity_ids)
            for entity_id in entity_ids:
```

```
                label = fetched_labels.get(entity_id, None)
                if label is not None:
                    valid_labels.append(label)
                    valid_entity_ids.append(entity_id)

            if valid_labels:
                Properties.append(prop_label)
                Property_IDs.append(prop_id)
                Related_Keyword_IDs.append("||".join(valid_entit
y_ids))
                Related_Keywords.append("||".join(valid_labels))

    return {
        "Properties": ";".join(Properties),
        "Property_IDs": ";".join(Property_IDs),
        "Related_Keyword_IDs": ";".join(Related_Keyword_IDs),
        "Related_Keywords": ";".join(Related_Keywords)
    }

for index, row in data.iterrows():
    id = row['id']
    keywords = row['keywords']

    keywords_list = [keyword.strip() for keyword in
keywords.split('; ') if keyword]

    all_properties = []
    all_property_ids = []
    all_related_keywords = []
    all_related_keyword_ids = []

    for keyword in keywords_list:
        related_concepts = get_related_concepts(keyword)
        if related_concepts:
            properties = related_concepts['Properties']
            property_ids = related_concepts['Property_IDs']
```

```
            related_keywords =
related_concepts['Related_Keywords']
            related_keyword_ids =
related_concepts['Related_Keyword_IDs']

            # Replace None with empty strings
            all_properties.append(properties if properties is
not None else '')
            all_property_ids.append(property_ids if property_ids
is not None else '')
            all_related_keywords.append(related_keywords if
related_keywords is not None else '')
            all_related_keyword_ids.append(related_keyword_ids
if related_keyword_ids is not None else '')
        else:
            all_properties.append('')
            all_property_ids.append('')
            all_related_keywords.append('')
            all_related_keyword_ids.append('')

    # Join the results into single strings
    final_properties = "//".join(all_properties)
    final_property_ids = "//".join(all_property_ids)
    final_related_keywords = "//".join(all_related_keywords)
    final_related_keyword_ids =
"//".join(all_related_keyword_ids)

    data.loc[data['id'] == id, 'properties'] = final_properties
    data.loc[data['id'] == id, 'property_ids'] =
final_property_ids
    data.loc[data['id'] == id, 'related_keywords'] =
final_related_keywords
    data.loc[data['id'] == id, 'related_keyword_ids'] =
final_related_keyword_ids
```

4.3 Implementasi Perhitungan Similarity

Pada bagian ini dilakukan implementasi perhitungan similarity antara koleksi dan kata kunci dengan tiga algoritma yaitu *Jaccard Similarity*, *Euclidean Distance*, dan *Cosine Similarity*. Hasil dari ketiga algoritma ini dievaluasi menggunakan metrik *precision*@k, *precision*, *recall*, dan *F1-score*.

Segmen Program 4.6 *Jaccard Similarity*

```
# Jaccard Similarity
def jaccard_similarity(doc_words, keyword_set, keyword):
    doc_set = set(doc_words)
    base_intersection = doc_set & keyword_set

    explicit_candidates = {word for word in doc_words if keyword
in word}
    explicit_intersection = explicit_candidates -
base_intersection
    final_intersection =
base_intersection.union(explicit_intersection)

    union = len(doc_set | keyword_set)
    jaccard_similarity = len(final_intersection) / union if
union else 0

    return jaccard_similarity

jaccard_similarity_results = []

# Hitung jaccard similarity untuk setiap koleksi yang ada
for doc_id, words in collection_words.items():
    similarity = jaccard_similarity(words, search_keywords_set,
search_keyword)
    if similarity > 0:
        jaccard_similarity_results.append((doc_id, similarity))

# Urutkan hasil berdasarkan nilai similarity
sorted_jaccard_results = sorted(jaccard_similarity_results,
key=lambda x: x[1], reverse=True)
```

Segmen Program 4.7 *Euclidean Distance*

```
# Euclidean Distance
def euclidean_distance(vector1, vector2):
    vector1 = vector1.toarray().flatten()
    vector2 = vector2.toarray().flatten()
    return np.sqrt(np.sum((vector1 - vector2) ** 2))

target_index = ids.index(search_keyword)
target_vector = vectorized_documents_combined[target_index]

euclidean_distance_results = []

for idx, doc_id in enumerate(ids):
    if str(doc_id) != search_keyword and str(doc_id).isdigit():
        # Hitung Euclidean Distance
        euclidean_dist = euclidean_distance(target_vector,
vectorized_documents_combined[idx])
        euclidean_distance_results.append((str(doc_id),
euclidean_dist))

sorted_euclidean_results = sorted(euclidean_distance_results,
key=lambda x: x[1])
euclidean_data = [{"id": doc_id, "distance": distance} for
doc_id, distance in sorted_euclidean_results]
df_euclidean = pd.DataFrame(euclidean_data)
```

Segmen Program 4.8 *Cosine Similarity*

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.tokenize import word_tokenize

corpus = df['keywords'].fillna('')
```

```
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(corpus)

query_tokens = word_tokenize(query.lower())
query_text = " ".join(query_tokens)
query_vector = vectorizer.transform([query_text])

similarity_scores = cosine_similarity(query_vector, X)

df_cosine = pd.DataFrame({
    'id': df['id'],
    'cosine_similarity': similarity_scores[0]
})

df_cosine = df_cosine.sort_values(by='cosine_similarity',
ascending=False).reset_index(drop=True)
```

Segmen Program 4.9 Metrik Evaluasi

```
ground_truth = pd.read_excel('ground_truth.xlsx')

merged_df = pd.merge(df_jaccard, df_euclidean, on="id",
how='outer')
merged_df = pd.merge(merged_df, ground_truth, on='id',
how='inner')
merged_df = pd.merge(merged_df, df_cosine, on='id', how='inner')

merged_df['y_true'] = (merged_df['score'] > 0).astype(int)

merged_df['y_pred_jaccard'] = (merged_df['similarity'] >
0).astype(int)
threshold_distance = merged_df['distance'].quantile(0.25)
merged_df['y_pred_euclidean'] = (merged_df['distance'] <
threshold_distance).astype(int)
threshold_cosine = 0
```

```
merged_df['y_pred_cosine'] = (merged_df['cosine_similarity'] >=
threshold_cosine).astype(int)

tn_jaccard, fp_jaccard, fn_jaccard, tp_jaccard =
confusion_matrix(merged_df['y_true'],
merged_df['y_pred_jaccard']).ravel()
tn_euclidean, fp_euclidean, fn_euclidean, tp_euclidean =
confusion_matrix(merged_df['y_true'],
merged_df['y_pred_euclidean']).ravel()
tn_cosine, fp_cosine, fn_cosine, tp_cosine =
confusion_matrix(merged_df['y_true'],
merged_df['y_pred_cosine']).ravel()

# Menghitung Recall@K, F1@K, MAP@K, dan Precision@K
def calculate_metrics_at_k(df, score_col, k):
    top_k = df.nlargest(k, score_col)
    precision_at_k = top_k['y_true'].sum() / k
    recall = top_k['y_true'].sum() / df['y_true'].sum()
    f1 = 2 * (precision_at_k * recall) / (precision_at_k +
recall) if (precision_at_k + recall) > 0 else 0
    relevant = top_k['y_true'].values
    precision_at_i = [
        relevant[:i+1].sum() / (i+1) for i in
range(len(relevant)) if relevant[i] == 1
    ]
    map_k = sum(precision_at_i) / df['y_true'].sum() if
df['y_true'].sum() > 0 else 0
    return precision_at_k, recall, f1, map_k

precision_jaccard_k, recall_jaccard_k, f1_jaccard_k,
map_jaccard_k = calculate_metrics_at_k(merged_df, 'similarity',
20)
precision_euclidean_k, recall_euclidean_k, f1_euclidean_k,
map_euclidean_k = calculate_metrics_at_k(merged_df, 'distance',
20)
```

```
precision_cosine_k, recall_cosine_k, f1_cosine_k, map_cosine_k =
calculate_metrics_at_k(merged_df, 'cosine_similarity', 20)

# Menghitung precision, recall, dan f1-score keseluruhan untuk
setiap metode
def calculate_overall_metrics(df, y_true_col, y_pred_col):
    precision = precision_score(df[y_true_col], df[y_pred_col])
    recall = recall_score(df[y_true_col], df[y_pred_col])
    f1 = f1_score(df[y_true_col], df[y_pred_col])
    return precision, recall, f1

precision_jaccard, recall_jaccard, f1_score_jaccard =
calculate_overall_metrics(merged_df, 'y_true', 'y_pred_jaccard')
precision_euclidean, recall_euclidean, f1_score_euclidean =
calculate_overall_metrics(merged_df, 'y_true',
'y_pred_euclidean')
precision_cosine, recall_cosine, f1_score_cosine =
calculate_overall_metrics(merged_df, 'y_true', 'y_pred_cosine')
```

4.4 Implementasi Sistem

Pada bagian ini dijelaskan implementasi sistem yang dilakukan mulai dari pembuatan koneksi ke GraphDB, inisialisasi vektor untuk perhitungan similarity, hingga proses search.

4.4.1 Koneksi pada GraphDB

Pertama, akan dibuat koneksi untuk menghubungkan sistem dengan *database* GraphDB. Koneksi dilakukan dengan memanfaatkan SPARQLWrapper. SPARQLWrapper merupakan *library* Python yang memungkinkan eksekusi query SPARQL pada endpoint GraphDB.

Segmen Program 4.10 Koneksi pada GraphDB

```
sparql_endpoint = "http://localhost:7200/repositories/koleksi"
sparql = SPARQLWrapper(sparql_endpoint)
```

4.4.2 Pembuatan Vektor

Pembuatan vektor dilakukan untuk mengubah data menjadi nol dan satu sehingga dapat dilakukan perhitungan *similarity* menggunakan *CountVectorizer*. Untuk mempersingkat waktu

pembuatan vektor, atribut dari setiap koleksi diambil dari GraphDB dan disimpan terlebih dahulu di *database* MySQL yang dapat dilihat pada Segmen Program 4.11.

Segmen Program 4.11 Pembuatan Vektor

```
# Query untuk fetch semua id koleksi
def fetch_all_ids():
    query = """
    PREFIX lib: <https://dewey.petra.ac.id/data/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX wdt: <http://www.wikidata.org/prop/direct/>
    PREFIX wd: <http://www.wikidata.org/entity/>

    SELECT DISTINCT ?id WHERE {
        ?collection rdf:type lib:Collection .
        ?collection lib:id ?id .
    }
    """
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    ids = [result["id"]["value"] for result in
results["results"]["bindings"]]
    return ids

# Fetch data keyword dan wikidata keyword koleksi
def fetch_keyword_label(id):
    query = f"""
    PREFIX lib: <https://dewey.petra.ac.id/data/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX wdt: <http://www.wikidata.org/prop/direct/>

    SELECT ?keywordLabel ?wdKeywordLabel WHERE {{
        ?collection lib:id {id} .
        ?collection lib:hasKeyword ?keyword .
```

```
        ?keyword rdfs:label ?keywordLabel .

        OPTIONAL {{
            ?keyword ?prop ?wdKeyword .
            ?wdKeyword rdfs:label ?wdKeywordLabel .
            FILTER(?prop NOT IN (
                    wdt:P123, wdt:P8875, wdt:P190, wdt:P282,
wdt:P7228, wdt:P1465, wdt:P6, wdt:P6224,
                    wdt:P7261, wdt:P1750, wdt:P2293, wdt:P1560,
wdt:P735, wdt:P1622, wdt:P5109,
                    wdt:P3103, wdt:P209, wdt:P1889, wdt:P47,
wdt:P793, wdt:P1343, wdt:1889, wdt:P31, wdt:P5008,
                    wdt:793
                ) &&
                ?prop != lib:relatedTo
            )
        }}
    }}
    """
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    return [
        (result.get("keywordLabel", {}).get("value"),
result.get("wdKeywordLabel", {}).get("value"))
        for result in results["results"]["bindings"]
    ]

# Fetch data subject koleksi
def fetch_subject_label(id):
    query = f"""
    PREFIX lib: <https://dewey.petra.ac.id/data/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?subjectLabel WHERE {{
        ?collection lib:id {id} .
```

```
        ?collection lib:hasSubject ?subject .
        ?subject rdfs:label ?subjectLabel .
    }}
    """
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    return [result.get("subjectLabel", {}).get("value") for
result in results["results"]["bindings"]]

# Fetch data creator koleksi
def fetch_creator_name(id):
    query = f"""
    PREFIX lib: <https://dewey.petra.ac.id/data/>

    SELECT ?creatorName WHERE {{
        ?collection lib:id {id} .
        ?collection lib:createdBy ?creator .
        ?creator lib:name ?creatorName .
    }}
    """
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    return [result.get("creatorName", {}).get("value") for
result in results["results"]["bindings"]]

# Fetch data contributor koleksi
def fetch_contributor_name(id):
    query = f"""
    PREFIX lib: <https://dewey.petra.ac.id/data/>

    SELECT ?contributorName WHERE {{
        ?collection lib:id {id} .
        ?collection lib:contributedBy ?contributor .
        ?contributor lib:name ?contributorName .
```

```
    }}
    """
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    return [result.get("contributorName", {}).get("value") for
result in results["results"]["bindings"]]

def fetch_details_for_id(id):
    future_keyword = executor.submit(fetch_keyword_label, id)
    future_subject = executor.submit(fetch_subject_label, id)
    future_creator = executor.submit(fetch_creator_name, id)
    future_contributor = executor.submit(fetch_contributor_name,
id)

    keyword_results = future_keyword.result()
    subject_results = future_subject.result()
    creator_results = future_creator.result()
    contributor_results = future_contributor.result()

    if id not in collection_words:
        collection_words[id] = {"keywords": [], "subjects": [],
"creators": [], "contributors": []}

    for keyword, wd_keyword in keyword_results:
        if keyword and keyword.lower() not in
collection_words[id]["keywords"]:
            collection_words[id]["keywords"].append(keyword.lowe
r())
        if wd_keyword and wd_keyword.lower() not in
collection_words[id]["keywords"]:
            collection_words[id]["keywords"].append(wd_keyword.l
ower())

    for subject in subject_results:
```

```
        if subject and subject.lower() not in
collection_words[id]["subjects"]:
            collection_words[id]["subjects"].append(subject.lowe
r())

    for creator in creator_results:
        if creator and creator.lower() not in
collection_words[id]["creators"]:
            collection_words[id]["creators"].append(creator.lowe
r())

    for contributor in contributor_results:
        if contributor and contributor.lower() not in
collection_words[id]["contributors"]:
            collection_words[id]["contributors"].append(contribu
tor.lower())

collection_words = {}
ids = fetch_all_ids()
with concurrent.futures.ThreadPoolExecutor(max_workers=5) as
executor:
    futures = {executor.submit(fetch_details_for_id, id): id for
id in ids}
    for future in concurrent.futures.as_completed(futures):
        id = futures[future]
        try:
            future.result()
        except Exception as e:
            print(f"Error fetching details for ID {id}: {e}")

# Menyiapkan data untuk dimasukkan ke MySQL
data_rows = []
for collection_id, types in collection_words.items():
    for type_name, words in types.items():
        for word in words:
```

```
            data_rows.append({"id": int(collection_id), "word":
word, "type": type_name})

# Koneksi ke MySQL
db = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="Password123",
    database="perpustakaan"
)

cursor = db.cursor()

# Membuat tabel
sql_create_table = """
CREATE TABLE collection_words_1000 (
    id INT,
    word TEXT,
    type TEXT
)
"""
cursor.execute(sql_create_table)

# Insert data ke tabel
sql_insert = "INSERT INTO collection_words_1000 (id, word, type)
VALUES (%s, %s, %s)"
try:
    for row in data_rows:
        cursor.execute(sql_insert, (row["id"], row["word"],
row["type"]))
    db.commit()
except mysql.connector.Error as error:
    print(f"Failed to insert data into MySQL table: {error}")
    db.rollback()

# Menutup koneksi
```

```
cursor.close()
db.close()
```

Segmen Program 4.12 *Function* Inisialisasi Vektor

```
is_initialized = False
collection_words = {}

def initialize_collection_words():
    global is_initialized
    if is_initialized:
        return

    db = mysql.connector.connect(
        host="localhost",
        user="root",
        passwd="Password123",
        database="perpustakaan"
    )
    cursor = db.cursor()
    query = "SELECT id, word FROM collection_words_1000"
    cursor.execute(query)

    for (id, word) in cursor.fetchall():
        if id not in collection_words:
            collection_words[id] = []
        if word not in collection_words[id]:
            collection_words[id].append(word)

    cursor.close()
    db.close()
    is_initialized = True
```

4.4.3 Proses Search

Pada bagian ini, dijelaskan proses search koleksi dari kata kunci yang diinputkan. Pertama, akan dilakukan inisialisasi vektor. Setelah itu kata kunci yang diinputkan akan melalui

proses keyword extraction dan pengambilan data tambahan dari Wikidata untuk memperkaya informasi. Setelah itu, akan dibuat vektor untuk query kata kunci dan dilakukan perhitungan similarity. Koleksi yang memiliki similarity lebih dari nol akan dianggap sebagai koleksi-koleksi yang relevan.

Segmen Program 4.13 Proses *Search*

```
@app.route("/search", methods=["GET"])
def search():
    global collection_words
    current_page = int(request.args.get("page", 1))
    query = request.args.get("query")

    if query in cache[collection_choice]:
        results = cache[collection_choice][query]
    else:
        initialize_collection_words(collection_words)
        documents = ['|||'.join(words).lower() for words in
collection_words.values()]
        ids = list(collection_words.keys())

        search_keyword = query.lower()
        search_keywords_list = set()

        if search_keyword not in ids:
            translated_text = ts.translate_text(search_keyword,
translator='bing', from_language='auto')
            search_keywords_list.add(translated_text.lower())
            doc = nlp(translated_text)

            for phrase in doc._.phrases[:10]:
                search_keywords_list.add(phrase.text.lower())
            for keyword in list(search_keywords_list):
                related_concepts =
get_related_concepts_cached(keyword)
                if related_concepts:
                    split_related_concepts = re.split(r'[;||]+',
related_concepts['Related_Keywords'])
                    for concepts in split_related_concepts:
                        search_keywords_list.add(concepts.lower(
))

            if not (query.startswith('"') and
query.endswith('"')) or (query.startswith("'") and
query.endswith("'")):
                search_keywords_split =
set(translated_text.lower().split())
                search_keywords_list.update(search_keywords_spli
t)

            search_keywords_list = [
```

```
                re.sub(r"['\"]", '', keyword).strip()  #
Menghapus tanda petik (' dan ")
                for keyword in search_keywords_list
            ]

            new_document =
'|||'.join(search_keywords_list).lower()
            documents.append(new_document)
            ids.append(search_keyword)

        def custom_tokenizer(doc):
            tokens = doc.split('|||')
            if search_keyword in doc:
                tokens.append(search_keyword)
            return tokens

        vectorizer_combined =
CountVectorizer(tokenizer=custom_tokenizer, lowercase=False)
        vectorized_documents_combined =
vectorizer_combined.fit_transform(documents)
        vectorized_documents_combined =
csr_matrix(vectorized_documents_combined)

        phrases_containing_keyword = [
            (phrase, vectorizer_combined.vocabulary_[phrase])
            for phrase in vectorizer_combined.vocabulary_
            if search_keyword in phrase
        ]

        vectorized_documents_combined =
vectorized_documents_combined.todok()
        for i, doc in enumerate(documents):
            if search_keyword in doc:
                for phrase, phrase_index in
phrases_containing_keyword:
                    vectorized_documents_combined[i,
phrase_index] = 1

        vectorized_documents_combined =
vectorized_documents_combined.tocsr()

        # Perhitungan Jaccard Similarity
        target_index = ids.index(search_keyword)
        target_vector =
vectorized_documents_combined[target_index]

        target_id = str(ids[target_index])
        batch_size = 1000
        batches = [
            (vectorized_documents_combined[i : i + batch_size],
ids[i : i + batch_size])
            for i in range(0,
vectorized_documents_combined.shape[0], batch_size)
```

```
        ]

        results = []
        with ThreadPoolExecutor(max_workers=10) as executor:
            all_results = executor.map(
                lambda batch: process_batch_sparse(batch[0],
target_vector, batch[1], target_id),
                batches,
            )
            results = [item for batch in all_results for item in
batch]

        sorted_results = sorted(results, key=lambda x: x[1],
reverse=True)
        cache[collection_choice][query] = sorted_results
        results = sorted_results
```

4.5 Implementasi Website *Search Engine*

Implementasi *search engine* ditampilkan berbasis website sesuai dengan desain interface yang ada pada bagian 3.5.

4.5.1 Pembuatan Halaman Home

Pada halaman home, selain terdapat *search box* untuk melakukan pencarian, halaman ini juga menampilkan delapan koleksi terbaru. *Query* yang digunakan untuk mengambil judul dan *theme* koleksi dapat dilihat pada Segmen Program 4.14.

Segmen Program 4.14 Pembuatan Halaman *Home*

```
@app.route("/", methods=["GET", "POST"])
def index():
    query = """
    PREFIX lib: <https://dewey.petra.ac.id/data/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?id ?title ?themeLabel WHERE {
        ?collection rdf:type lib:Collection .
        ?collection lib:id ?id .
        OPTIONAL {
            ?collection lib:title ?title .
        }
        OPTIONAL {
            ?collection lib:hasTheme ?theme .
```

```
            ?theme rdfs:label ?themeLabel .
        }
    }
    ORDER BY DESC(?id)
    LIMIT 8
    """
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    collections = {}

    for result in results["results"]["bindings"]:
        id = result["id"]["value"]

        title = result["title"]["value"]
        themeLabel = result["themeLabel"]["value"]

        if id not in collections:
            collections[id] = {}

        if title:
            collections[id]['title'] = title
        if themeLabel:
            collections[id]['themeLabel'] = themeLabel

    collections_json = json.dumps(collections)
    return render_template("home.html",
collections=collections_json)
```

4.5.2 Pembuatan Halaman Hasil Search

Halaman ini ditampilkan setelah pencarian berhasil dilakukan. Pada halaman ini, koleksi-koleksi ditampilkan secara berurutan berdasarkan tingkat relevansinya menurut sistem, sesuai dengan kata kunci yang diinputkan. Setiap halaman menampilkan sepuluh koleksi sekaligus.

Segmen Program 4.15 Pembuatan Halaman Hasil Search

```
@app.route("/search", methods=["GET"])
```

```
def search():
    global is_initialized
    if not is_initialized:
        initialize_collection_words()

    query = request.args.get("query")
    results = search_process(query)
    id_values = " ".join(doc_id for doc_id, similarity in
results)
    similarity_results = {doc_id: similarity for doc_id,
similarity in results}
    batch_size = 100
    id_values_list = id_values.split()
    batch_results = []
    collection_data = {}

    for id_batch in chunk_list(id_values_list, batch_size):
        # Membuat query untuk setiap batch
        batch_id_values = " ".join(id_batch)
        sparql_query = f"""
        PREFIX lib: <https://dewey.petra.ac.id/data/>
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

        SELECT ?id ?title ?themeLabel ?creatorName ?contributorN
ame WHERE {{
            ?collection rdf:type lib:Collection .
            ?collection lib:id ?id .
            VALUES ?id {{ {batch_id_values} }}
            OPTIONAL {{
                ?collection lib:title ?title .
            }}
            OPTIONAL {{
                ?collection lib:hasTheme ?theme .
                ?theme rdfs:label ?themeLabel .
```

```
            }}
            OPTIONAL {{
                ?collection lib:createdBy ?creator .
                ?creator lib:name ?creatorName .
            }}
            OPTIONAL {{
                ?collection lib:contributedBy ?contributor .
                ?contributor lib:name ?contributorName .
            }}
        }}
        ORDER BY DESC(?id)
        """
        sparql.setQuery(sparql_query)
        sparql.setReturnFormat(JSON)
        sparql_results = sparql.query().convert()

        # Proses hasil query untuk setiap batch
        for result in sparql_results["results"]["bindings"]:
            id_value = result["id"]["value"]

            if id_value not in collection_data:
                collection_data[id_value] = {
                    "id": id_value,
                    "title": result.get("title",
{}).get("value", "No Title"),
                    "themeLabel": result.get("themeLabel",
{}).get("value", "No Theme"),
                    "creatorName": [],
                    "contributorName": [],
                    "similarity":
similarity_results.get(id_value, 0)
                }

            creator = result.get("creatorName", {}).get("value")
            contributor = result.get("contributorName",
{}).get("value")
```

```
            if creator and creator not in
collection_data[id_value]["creatorName"]:
                collection_data[id_value]["creatorName"].append(
creator)

            if contributor and contributor not in
collection_data[id_value]["contributorName"]:
                collection_data[id_value]["contributorName"].app
end(contributor)

    collection_data_list = list(collection_data.values())
    collection_json = json.dumps(collection_data_list)

    return render_template("page2.html", query=query,
results=collection_json)
```

4.5.3 Pembuatan Halaman Detail Koleksi

Halaman detail koleksi ditampilkan ketika salah satu *box* koleksi di-klik. Halaman ini memuat informasi detail koleksi seperti judul, deskripsi, nama *creator*, nama *contributor*, *theme*, *category*, *subcategory*, dan *subject*. *Query* untuk mengambil detail koleksi dapat dilihat pada Segmen Program 4.16.

Segmen Program 4.16 Pembuatan Halaman Detail Koleksi

```
@app.route("/view", methods=["GET"])
def view():
    data_id = request.args.get("id")
    sparql_query = f"""
    PREFIX lib: <https://dewey.petra.ac.id/data/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?id ?title ?description ?themeLabel ?categoryLabel ?s
ubcategoryLabel ?subjectLabel ?creatorName ?contributorName ?con
tributorRoleLabel WHERE {{
```

```
        ?collection rdf:type lib:Collection .
        ?collection lib:id ?id .
        VALUES ?id {{ {data_id} }}
        OPTIONAL {{
            ?collection lib:title ?title .
        }}
        OPTIONAL {{
            ?collection lib:description ?description .
        }}
        OPTIONAL {{
            ?collection lib:hasTheme ?theme .
            ?theme rdfs:label ?themeLabel .
        }}
        OPTIONAL {{
            ?collection lib:hasCategory ?category .
            ?category rdfs:label ?categoryLabel .
        }}
        OPTIONAL {{
            ?collection lib:hasSubcategory ?subcategory .
            ?subcategory rdfs:label ?subcategoryLabel .
        }}
        OPTIONAL {{
            ?collection lib:hasSubject ?subject .
            ?subject rdfs:label ?subjectLabel .
        }}
        OPTIONAL {{
            ?collection lib:createdBy ?creator .
            ?creator lib:name ?creatorName .
        }}
        OPTIONAL {{
            ?collection ?predicate ?contributor .
            ?predicate rdfs:subPropertyOf* lib:contributedBy .
            ?predicate rdfs:label ?contributorRoleLabel .
            ?contributor lib:name ?contributorName .
        }}
    }}
```

```
    """

    sparql.setQuery(sparql_query)
    sparql.setReturnFormat(JSON)
    sparql_results = sparql.query().convert()

    collection_data = {}
    for result in sparql_results["results"]["bindings"]:
        id_value = result["id"]["value"]

        if id_value not in collection_data:
            collection_data[id_value] = {
                "id": id_value,
                "title": result.get("title", {}).get("value",
"No Title"),
                "description": result.get("description",
{}).get("value", "No Description"),
                "themeLabel": result.get("themeLabel",
{}).get("value", "No Theme"),
                "categoryLabel": result.get("categoryLabel",
{}).get("value", "No Category"),
                "subcategoryLabel":
result.get("subcategoryLabel", {}).get("value", "No
Subcategory"),
                "subjects": [],
                "creatorName": [],
                "contributorName": [],
            }

        subject = result.get("subjectLabel", {}).get("value",
"No Subject")
        creator = result.get("creatorName", {}).get("value",
"Unknown")
        contributor = result.get("contributorName",
{}).get("value", "Unknown")
```

```
        contributor_role = result.get("contributorRoleLabel",
{}).get("value", "")

        if subject and subject not in
collection_data[id_value]["subjects"]:
            collection_data[id_value]["subjects"].append(subject
)

        if creator and creator not in
collection_data[id_value]["creatorName"]:
            collection_data[id_value]["creatorName"].append(crea
tor)

        if contributor:
            contributor_data = {"name": contributor, "role":
contributor_role if contributor_role else None}
            if contributor_data not in
collection_data[id_value]["contributorName"]:
                collection_data[id_value]["contributorName"].app
end(contributor_data)

    collection_data_list = list(collection_data.values())
    collection_json = json.dumps(collection_data_list)

    return render_template("page3.html",
results=collection_json)
```