BAB III
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

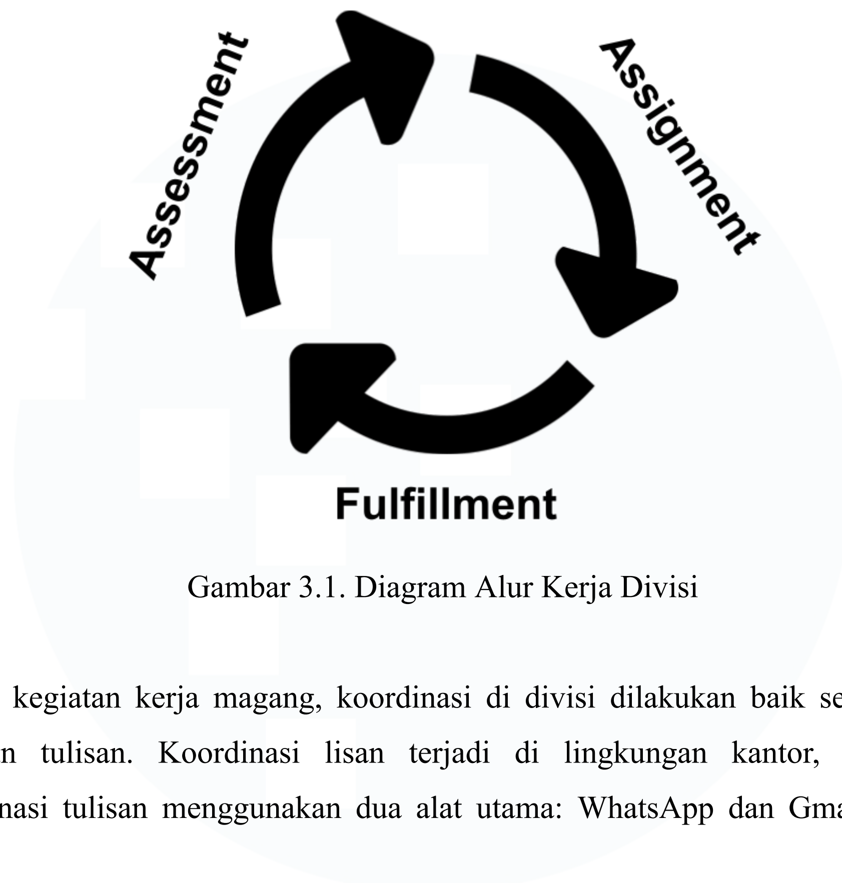
Divisi MDM di Kawan Lama Group terbagi menjadi dua subdivisi, yaitu Officer dan Data Analyst, yang masing-masing dipimpin oleh seorang Manager. Kedua subdivisi ini berada di bawah naungan seorang General Manager, yang melapor kepada Senior VP of Strategic Collaboration Project. Dalam pelaksanaan magang, posisi MDM Data Analyst Intern disupervisi secara langsung oleh General Manager, sesuai dengan informasi yang tertera pada portal Human Capital Kawan Lama Group. Alur kerja magang, seperti yang digambarkan pada Gambar 3.1 di bawah, bersifat iteratif dan terbagi menjadi tiga bagian utama, meliputi:

A. ***Assignment*****,** Pada tahapan ini, penugasan diberikan secara lisan atau tulisan oleh *Data Analyst Manager* atau *General Manager* sebagai *supervisor*. Tahapan ini mencakup penyampaian informasi mengenai tenggat waktu (*deadline*), ketentuan yang harus diikuti, serta penyediaan sumber daya yang dibutuhkan. Ini meliputi data terkait, instruksi, dan sumber daya lainnya yang mendukung keberhasilan proses.

B. ***Fulfillment,*** pemenuhan penugasan oleh setiap anggota tim MDM diberi kebebasan dalam hal teknik dan alat yang digunakan untuk menyelesaikan tugas, selama proses tersebut dilakukan dalam kurun waktu yang telah ditentukan.

C. ***Assessment,*** tahapan terakhir dalam alur penugasan adalah proses peninjauan hasil kerja. Proses ini dilakukan secara mingguan melalui pertemuan rutin (*weekly meeting*) yang dihadiri oleh *General Manager*, *Data Analyst Manager*, serta seluruh rekan divisi..

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.1. Diagram Alur Kerja Divisi

Dalam kegiatan kerja magang, koordinasi di divisi dilakukan baik secara lisan maupun tulisan. Koordinasi lisan terjadi di lingkungan kantor, sementara koordinasi tulisan menggunakan dua alat utama: WhatsApp dan Gmail/Google Chat.

3.2 Tugas dan Uraian Kerja Magang

Selama proses kerja magang MDM *Data Analyst Intern* bertanggung jawab dalam penyajian analisis serta pembenahan *master data* pada skala korporat. Besarnya jumlah data yang dikelola atau tingkat kompleksitas penyelesaian yang diperlukan tidak memungkinkannya penggunaan Excel atau Google Spreadsheet sebagai aplikasi pengelolaan data untuk digunakan. Memahami batasan tersebut, maka seluruh proses analisis serta lainnya mengimplementasi Python, Visual Studio Code, serta Google Colaboratory (Colab) sebagai *tools*. Dalam pengerjaan setiap *request* dilakukan dengan mempertimbangkan reusabilitas *code* untuk tugas yang serupa dimasa yang akan mendatang. Adapun penjabaran pekerjaan berdasarkan dengan periode pengerjaan dapat dilihat pada tabel 3.1. di bawah.

Tabel 3.1. Penjabaran Pekerja per Minggu

No	Pekerjaan	Minggu Ke-	Durasi (Minggu)	Tanggal Mulai	Tanggal Selesai
1	Pengenalan projek dan tim yang terlibat.	1	1	13 Mei 2024	13 Mei 2024
2	Pemenuhan Request: Transformasi Data Breadcrumb	2-3	2	20 Mei 2024	31 Mei 2024
3	Pemenuhan Request: Pemisahan Data Duplikat	4-5	2	3 Juni 2024	14 Juni 2024
4	Pemenuhan Request: Penggabungan Data Artikel	6-7	2	17 Juni 2024	28 Juni 2024
5	Pemenuhan Request: Rekonstruksi Template Bulk	8-9	2	1 Juli 2024	12 Juli 2024
6	Analisis Kelengkapan dan Kualitas Atribut Data	10	1	15 Juli 2024	19 Juli 2024
7	Pengembangan Antarmuka *Website* untuk Otomasi *request*	11-13	3	22 Juli 2024	9 Agustus 2024

8	Google AppScript untuk otomatisasi Google Form	14-17	4	12 Agustus 2024	6 September 2024

3.2.1 Pengenalaan Perusahaan dan Projek

Kegiatan pertama dalam praktik kerja magang adalah mengikuti *Personnel Employee Orientation Program* **(**PEOP**)** yang dibawakan oleh divisi human capital. Program ini bertujuan untuk memperkenalkan karyawan baru pada lingkungan kerja, aturan, serta tata cara lainnya sebagai panduan selama beraktivitas di Kawan Lama Group. Setelah itu, General Manager, yang juga bertindak sebagai supervisor, menjelaskan perihal proyek serta penugasan yang akan dikerjakan selama periode magang.

3.2.2 Pemenuhan Request: Transformasi Data *Breadcrumb*

Breadcrumb merupakan struktur hierarki yang menggambarkan bagaimana sebuah produk dikategorikan. hierarki ini tersusun atas nama perusahaan, departemen terkait, jenis komoditas, dan kelompok produk. Saat ini *master data* tersimpan dalam basis data dengan struktur seperti pada Gambar 3.2. di bawah. Struktur ini belum sesuai dengan ketentuan sistem PIM. Maka dari itu penyesuaian struktur diperlukan untuk kebutuhan migrasi *breadcrumb* dari basis data kedalam sistem PIM untuk memastikan keterbaharuan data secara berkala.

Hierarchy	Hierarchy Node	Validity Start Date	Validity End Date	Hierarchy Level	Parent Node	Description	Long Text	Name: Hierarchy Level
ORG	B200	01/01/2020	12/31/9999	2	ORG	Acme Corporation	Global operations	Company
ORG	B20001	01/01/2020	12/31/9999	3	B200	Sales Department	Handles all sales	Department
ORG	B20002	01/01/2020	12/31/9999	3	B200	HR Department	Manages HR activ	Department
ORG	B2000101	01/01/2020	12/31/9999	4	B20001	Domestic Sales	Sales within the co	Commodity
ORG	B2000102	01/01/2020	12/31/9999	4	B20001	International Sales	Sales outside the	Commodity
ORG	B2000103	01/01/2020	12/31/9999	4	B20001	Online Sales	E-commerce oper	Commodity
ORG	B200010101	01/01/2020	12/31/9999	5	B2000101	B2B Sales	Business-to-Busin	Class
ORG	B200010102	01/01/2020	12/31/9999	5	B2000101	B2C Sales	Business-to-Cons	Class
ORG	B20001010101	01/01/2020	12/31/9999	6	B200010101	B2B Enterprise Clients	Large business cli	Product Group
ORG	B20001010102	01/01/2020	12/31/9999	6	B200010101	B2B SME Clients	Small and mediun	Product Group
ORG	B2000201	01/01/2020	12/31/9999	4	B20002	Recruitment	Talent acquisition	Commodity
ORG	B2000202	01/01/2020	12/31/9999	4	B20002	Employee Relations	Managing employ	Commodity
ORG	B200020101	01/01/2020	12/31/9999	5	B2000201	Executive Recruitmen	Hiring for executiv	Class
ORG	B200020102	01/01/2020	12/31/9999	5	B2000201	Entry-level Recruitmer	Hiring for entry-lev	Class
ORG	B200020201	01/01/2020	12/31/9999	5	B2000202	Conflict Resolution	Handling workplac	Class
ORG	B200020202	01/01/2020	12/31/9999	5	B2000202	Employee Engagemer	Enhancing employ	Class

Gambar 3.2. Contoh *Master Data Breadcrumb*

Struktur data yang diterima oleh sistem PIM mengharuskan data untuk tersusun berdasarkan dengan tingkatan dari setiap hierarki sesuai dengan hierarki pada

level sebelumnya yang berkorespondensi seperti pada Gambar 3.3. di bawah. Gambar di bawah menunjukan satu baris data yang telah berhasil ditransformasi untuk kemudian membentuk *breadcrumb*.

Hierarchy Node	Description	Name: Hierarchy Level	Concatenated Description
B200	Acme Corporation	Company	B200 - Acme Corporation
Next Hierarchy Node	**Next Description**	**Next Name: Hierarchy Level**	**Next Concatenated Description**
B20001	Sales Department	Department	B20001 - Sales Department
Next Hierarchy Node	**Next Description**	**Next Name: Hierarchy Level**	**Next Concatenated Description**
B2000101	Domestic Sales	Commodity	B2000101 - Domestic Sales
Next Hierarchy Node	**Next Description**	**Next Name: Hierarchy Level**	**Next Concatenated Description**
B200010101	B2B Sales	Class	B200010101 - B2B Sales
Final Breadcrumb			
B200 - Acme Corporation/B20001 - Sales Department/B2000101 - Domestic Sales/B200010101 - B2B Sales			

Gambar 3.2. Contoh *Master Data Breadcrumb*

Proses pemetaan ini dilakukan dengan mengimplementasikan library Pandas. Proses dimulai dengan fungsi utama *get_df_breadcrumb*() yang mengelola seluruh alur pemrosesan seperti pada Gambar 3.4. di bawah. Fungsi ini memulai dengan menginisialisasi DataFrame breadcrumb untuk level hierarki pertama menggunakan fungsi *initialize_breadcrumb*().

```
def get_df_breadcrumb(df, levels, cols):
    df_breadcrumb = pd.DataFrame()

    for i, level in enumerate(levels):

        if i == 0:  # Initialize the breadcrumb DataFrame for the first level
            df_breadcrumb = initialize_breadcrumb(df, level, cols)
            df_breadcrumb = generate_additional_col(df_breadcrumb)
        else:  # Process subsequent levels
            df_breadcrumb = process_level(df_breadcrumb, df, cols)

        # Drop 'Parent Node' column for the last iteration
        if i == len(levels) - 1:
            df_breadcrumb.drop(columns=['Parent Node'], inplace=True, errors='ignore')

    df_breadcrumb = generate_breadcrumbs(df_breadcrumb)
    return df_breadcrumb
```

Gambar 3.4. Fungsi *get_df_breadcrumb*()

Fungsi *initialize_breadcrumb*() seperti pada Gambar 3.5. di bawah berfungsi untuk mengambil data dari DataFrame yang sesuai dengan level hierarki yang ditentukan. Jika kolom *'Hierarchy Level'* belum ada, fungsi ini akan menetapkannya berdasarkan pemetaan level hierarki yang telah ditentukan sebelumnya. Setelah itu, fungsi ini menyaring data untuk hanya menyertakan baris yang sesuai dengan level hierarki yang dimaksud, mengatur ulang indeks, dan memilih kolom-kolom yang relevan untuk menghasilkan DataFrame awal.

```
def initialize_breadcrumb(df, level, cols):
    if 'Hierarchy Level' not in df.columns:
        level_pair = {
            'Company' : 2,
            'Department' : 3,
            'Comodity' : 4,
            'Class' : 5,
            'Product Group' : 6
        }

        df['Hierarchy Level'] = df['Name: Hierarchy Level'].map(level_pair)


    df_level = df[df['Hierarchy Level'] == level].copy()
    df_level.reset_index(drop=True, inplace=True)
    return df_level[cols]
```

Gambar 3.5. Fungsi *initialize_breadcrumb*()

Selanjutnya, fungsi *generate_additioanl_col*() pada Gambar 3.6. di bawah digunakan untuk menambahkan kolom tambahan pada DataFrame *breadcrumb*. Fungsi ini menggabungkan nilai dari kolom '*Hierarchy Node'* dan *'Description*' menjadi satu kolom baru yang memberikan informasi lebih detail tentang node hierarki tersebut. Fungsi ini memastikan bahwa setiap breadcrumb memiliki deskripsi yang jelas dan mudah dipahami.

```
def generate_breadcrumbs(df_breadcrumb):
    print("DF BREADCRUMBS: ", df_breadcrumb.head(2))
    fourth_cols = df_breadcrumb.iloc[:, 3::4]

    # Concatenate the values in these columns into a single string for each row
    df_breadcrumb['Breadcrumbs'] = fourth_cols.apply(lambda row: '/'.join(
        row.dropna().astype(str)), axis=1)

    return df_breadcrumb
```

Gambar 3.6. Fungsi *generate_breadcrumbs*()

Untuk level-level hierarki berikutnya, fungsi *process_level*() pada Gambar 3.7. di bawah digunakan untuk memproses dan menggabungkan data dengan DataFrame breadcrumb yang sudah ada, dengan memetakan *node parent* ke *node* anak yang sesuai.

```
def process_level(df_breadcrumb, df, cols):
    if 'Parent Node' not in df_breadcrumb.columns:
        raise KeyError("Column 'Parent Node' is missing in the breadcrumb DataFrame.")

    copy = df_breadcrumb.copy()
    df_temp = copy[['Parent Node']].merge(df[cols],
                                          left_on='Parent Node',
                                          right_on='Hierarchy Node',
                                          how='left').iloc[:, 1:]
    df_temp.reset_index(drop=True, inplace=True)
    df_breadcrumb.drop(columns=['Parent Node'], inplace=True)
    df_temp.rename(columns={'Parent Node_y': 'Parent Node'}, inplace=True)

    df_temp = generate_additional_col(df_temp)
    return pd.concat([df_temp, df_breadcrumb], axis=1)
```

Gambar 3.7. Fungsi *process_level*()

Terakhir, fungsi *generate_breadcrumbs*() pada Gambar 3.7. di bawah digunakan untuk menggabungkan nilai-nilai dari kolom-kolom yang relevan menjadi satu string breadcrumb yang lengkap, memudahkan navigasi dan penelusuran informasi dalam sistem manajemen data produk.

```
def generate_breadcrumbs(df_breadcrumb):
    print("DF BREADCRUMBS: ", df_breadcrumb.head(2))
    fourth_cols = df_breadcrumb.iloc[:, 3::4]

    # Concatenate the values in these columns into a single string for each row
    df_breadcrumb['Breadcrumbs'] = fourth_cols.apply(lambda row: '/'.join(
        row.dropna().astype(str)), axis=1)

    return df_breadcrumb
```

Gambar 3.8. Fungsi *generate_breadcrumbs*()

3.2.3 Pemenuhan Request: Pemisahan Data Duplikat

Dalam praktik bisnis pembuatan ulang artikel dengan deskripsi yang sama kerap kali terjadi. Hal ini berdampak pada duplikasi data dimana satu produk yang sama terdaftar atas dua entitas dengan kode unik yang berbeda. Hal ini dapat mengakibatkan kebingungan dalam proses manajemen inventaris, mempengaruhi akurasi laporan penjualan, dan menyebabkan ketidakefisienan dalam pengelolaan data. Oleh karena itu, upaya deduplikasi atau penghapusan data duplikat sangat diperlukan dalam Product Information Management (PIM) untuk memastikan integritas data perlu untuk dilakukan. Pencarian data duplikat dilakukan dengan menggunakan fungsi *find_duplicates*() pada Gambar 3.9. di bawah.

```python
def find_duplicates(df):
    """
    Returns rows where either 'Article Code / Article Number SAP' or 'Article description' columns have duplicates.

    Parameters:
    df (pd.DataFrame): DataFrame containing the columns to check for duplicates.

    Returns:
    pd.DataFrame: DataFrame containing rows with duplicates in either of the specified columns.
    """
    # Find duplicates in 'Article Code / Article Number SAP'
    duplicate_article_code = df[df['Article Code / Article Number SAP'].duplicated(keep=False)]

    # Find duplicates in 'Article description'
    duplicate_description = df[df['Article description'].duplicated(keep=False)]

    # Combine both duplicates
    duplicates = pd.concat([duplicate_article_code, duplicate_description]).drop_duplicates()

    return duplicates
```

Gambar 3.9. Fungsi *find_duplicates*()

Fungsi ini bekerja dengan cara mengidentifikasi dan menggabungkan semua baris yang memiliki nilai yang sama atau berulang di salah satu dari kedua kolom tersebut. Dengan menggunakan metode untuk mencari nilai duplikat dan menghilangkan salinan yang sama, fungsi ini menghasilkan DataFrame yang berisi hanya baris-baris yang memiliki nilai yang duplikat di kolom-kolom yang ditentukan.

3.2.4 Pemenuhan Request: Penggabungan Data Artikel

Penggabungan data artikel dari setiap *business units* juga perlu dilakukan pada tahap migrasi. Penggabungan dilakukan dengan mengimplementasi fungsi *generator* yang tersedia pada Python pada Gambar 3.10. di bawah

```
import os
directory = 'FILE PATH'

file_name = []
for root, dirs, files in os.walk(directory):
    for filename in files:
        file_name.append(os.path.join(root, filename))

# Define a generator to read excel files
def excel_generator(file_list):
  for file in file_list:
    print("Processing: ", file)
    yield pd.read_excel(file)

# Concat all excel files
df = pd.concat(excel_generator(file_name))

# Remove duplicates
df.drop_duplicates(inplace=True)

# Sort by SKU
df.sort_values(by='SKU', inplace=True) # Assuming 'SKU' is the column name

# Export to excel
df.to_excel(os.path.join(directory, 'Combined.xlsx'), index=False)
```

Gambar 3.10. Program untuk Penggabungan Artikel

Proses dimulai dengan mengumpulkan semua berkas Excel dari suatu direktori dan subdirektorinya, yang memungkinkan pengguna untuk menangani data dari berbagai sumber dalam satu waktu. Setelah berkas-berkas ini dikumpulkan, kode menggunakan generator untuk membaca setiap berkas secara bertahap, menghindari penggunaan memori yang berlebihan dengan hanya memproses satu berkas pada satu waktu. Data dari semua berkas kemudian digabungkan menjadi satu DataFrame besar, yang mempermudah analisis dan manipulasi data secara keseluruhan. Langkah selanjutnya adalah membersihkan data dengan menghapus entri duplikat dan mengurutkan data berdasarkan kolom tertentu, memastikan bahwa data yang dihasilkan adalah akurat dan terstruktur dengan baik. Akhirnya, hasil data yang telah diproses disimpan dalam berkas Excel baru, memungkinkan distribusi dan penggunaan data yang mudah. Pendekatan ini mengoptimalkan alur kerja pengolahan data, memudahkan integrasi dan pembersihan data dari berbagai sumber, serta menghasilkan output yang siap digunakan untuk analisis lebih lanjut atau pelaporan.

3.2.5 Pemenuhan Request: Rekonstruksi *Template Bulk*

Pembuatan artikel dalam jumlah besar pada PIM oleh pengguna dapat dilakukan dengan menggunakan *template bulk* dalam format Excel. Namun, *template* yang disediakan oleh sistem seperti pada Gambar 3.11. di bawah menerima komplain dari pengguna karena formatnya yang membingungkan membuat pengisian *template* menjadi tidak optimal. Beberapa masalah yang hadir dari *template* ini antara lain: kolom yang tidak tersortir berdasarkan kewajiban pengisian, *template* belum termasuk kolom atribut *product* yang dikhususkan atau berbeda untuk setiap *taxonomy*, serta penyajian informasi yang terlalu kompleks. Maka dari itu upaya rekonstruksi *template bulk* yang tersortir dengan baik, lengkap dengan atribut *product* per *taxonomy*, dengan tampilan yang lebih sederhana perlu untuk dilakukan.

Taxonomy Node	Prior Ven	Vendor Su	Regular V	Return Ag	Ctry of Or	Certificate	Certificate	Planned D	Confirmat	Purchasin	Purchasin	Info Recor
Automotive > Tire Accessories > Tire Repair Kits	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin	Purchasin
	Good to h	Good to h	Mandatoı	Mandatoı	Good to h	Good to h	Good to h	Mandatoı	Mandatoı	Mandatoı	Mandatoı	Mandatoı
	Text + Nu	Text + Nu	Text + Nu	Text + Nu	Text + Nu	Text + Nu	Text + Nu	Numeric	Text + Nu	Text + Nu	Text + Nu	Text + Nu
	Single	Single	Single	Single	Single	Single	Single	Single	Single	Single	Single	Single
			No	03 | Retu	QA | Qatar		1 | DKS+LB+SGS		Z003 | Im	LOCL		2 | Consig
			Yes	01 | No re	RE | Reunion		2 | SNI + SGS		Z001 | Import Confirmations			3 | Subco
				02 | Retu	RO | Romania		3 | SNI + BV		Z002 | Local Confirmations			0 | Standa
					RU | Russian Federa		4 | SKI+K.TUMB					
					RW | Rwanda		5 | NPIK					
					SA | Saudi Arabia		6 | IT					
					SB | Solomon Islanc		7 | NPIK + IT					
					SC | Seychelles		8 | SGS+NPK+IT					
					SD | Sudan		9 | BV+NPIK+IT					
					SE | Sweden		= | BATAN					

Gambar 3.11. Tampilan *Template Bulk* Sebelum Rekonstruksi

Dalam proses rekonstruksi, *template* di atas berperan sebagai *base template*. Atribut dari setiap produk bersumber pada sebuah basis data dalam format Excel dengan format seperti pada Gambar 3.12. di bawah.

ID	Category	Subcategory	Sub-Subcategory	Product Type	Item Name	Attr ID	Attribute Label	Requirement	Value Type	Value Unit
1001	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	1	Key Feature	Essential	String	No Unit
1002	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	2	Model Number	Essential	Alphanumeri c	No Unit
1003	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	3	Fabric Type	Essential	String	No Unit
1004	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	4	Color Variant	Essential	String	No Unit
1005	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	5	Product Style	Essential	String	No Unit
1006	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	6	Dimension s	Essential	Numeric	cm
1007	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	7	Shape	Optional	String	No Unit
1008	Health	Skin Care	Face Accessories	Face Cloths & Pads	Face Cloths & Pads	8	Target Audience	Optional	String	No Unit

Gambar 3.12. Tampilan *Base Template*

3.2.6 Analisis Kelengkapan dan Kualitas Atribut Data

Kelengkapan serta kesesuaian pengisian atribut data menjadi elemen penting dalam menyajikan informasi yang berkualitas atas data produk. Maka dari itu analisis serta identifikasi keduanya perlu untuk dilakukan. Proses ini melibatkan data dengan ukuran mencapai 500mb, hal ini tidak memungkinkannya upaya penyajian data yang relevan sesuai dengan tujuan perlu untuk dilakukan menggunakan *tools* konvensional seperti Google Spreadsheet ataupun Microsoft Excel. Tujuan daripada analisa ini untuk mengidentifikasi data yang telah aktif. Untuk dikategorikan sebagai aktif, artikel setidaknya harus memiliki satu kolom pada atribut produk yang tidak kosong. Proses ini dilakukan dengan menerapkan fungsi *filter_and_clean_product*() pada Gambar 3.13. di bawah.

```
def filter_and_clean_product_rows(df):
    # Step 1: Identify columns that contain '[Product]' in their names except 'Article Number[Product]'
    product_columns = [col for col in df.columns if '[Product]' in col and col != 'Article Number[Product]']

    # Step 2: Filter rows where at least one of the identified columns has a non-empty value
    filtered_df = df[df[product_columns].notna().any(axis=1)]

    # Step 3: Remove columns that are empty in all rows
    cleaned_df = filtered_df.dropna(axis=1, how='all')

    return cleaned_df
```

Gambar 3.13. Fungsi *filter_and_clean_product*()

Pertama, fungsi ini mengidentifikasi kolom-kolom yang mengandung substring '[*Product*]' dalam namanya, kecuali kolom '*Article Number*[*Product*]'. Langkah ini memastikan bahwa hanya kolom yang relevan untuk produk yang akan dipertimbangkan. Selanjutnya, fungsi memfilter baris-baris DataFrame sehingga hanya menyertakan baris di mana setidaknya satu dari kolom yang teridentifikasi memiliki nilai yang tidak kosong. Setelah itu, fungsi menghapus kolom-kolom yang kosong di semua baris yang tersisa, sehingga menghilangkan kolom yang tidak menyimpan data yang relevan. Hasil akhirnya adalah DataFrame yang bersih dan hanya berisi baris serta kolom dengan informasi produk yang signifikan, sehingga memudahkan analisis dan penggunaan data lebih lanjut.

```python
# Merge df1 and df2 on 'ID' and 'InputEntityPIMId', handling duplicates
df1_df2_merged = pd.merge(df1, df2, on=['ID', 'InputEntityPIMId'], how='outer', suffixes=('_df1', '_df2'))

# Fill missing values from df2 to df1 for columns with same name
for col in df1_df2_merged.columns:
    if col.endswith('_df1') and col[:-5] + '_df2' in df1_df2_merged.columns:
        df1_df2_merged[col] = df1_df2_merged[col].fillna(df1_df2_merged[col[:-5] + '_df2'])
        df1_df2_merged = df1_df2_merged.drop(col[:-5] + '_df2', axis=1)

# Rename columns to remove suffixes
df1_df2_merged = df1_df2_merged.rename(columns={col: col[:-5] for col in df1_df2_merged.columns if col.endswith('_df1')})

# Merge the result with df4_filtered on 'ID' and 'InputEntityPIMId'
merged_df = pd.merge(df1_df2_merged, df4_filtered, on=['ID', 'InputEntityPIMId'], how='outer', suffixes=('', '_df4'))

# Fill missing values from df4_filtered to the merged dataframe for columns with same name
for col in merged_df.columns:
    if col.endswith('_df4') and col[:-5] in merged_df.columns:
        merged_df[col[:-5]] = merged_df[col[:-5]].fillna(merged_df[col])
        merged_df = merged_df.drop(col, axis=1)

# Display the final merged dataframe
merged_df
```

Gambar 3.14. *Script* Menggabungkan Data Bersih

Fungsi tersebut kemudian diaplikasikan pada 3 *file* data, untuk kemudian disatukan dengan mengaplikasikan *script* seperti pada Gambar 3.14. diatas. Penggabungan data dilakukan berdasarkan kolom *identifier.* Sebelum analisa atribut data dilakukan, penambahan kolom '*Taxonomy*' serta perubahan nama pada beberapa kolom atribut perlu dilakukan dengan mengimplementasi fungsi seperti pada Gambar 3.14 diatas.

```python
attributes = pd.concat(all_sheets, ignore_index=True)

def get_taxonomy(row):
  levels = [row['Level 1'], row['Level 2'], row['Level 3'], row['Level 4']]
  return ' > '.join(str(level) for level in levels if level != 'nan')

# Apply the get_taxonomy function to create a 'Taxonomy' column
attributes['Taxonomy'] = attributes.apply(get_taxonomy, axis=1)

def process_attribute_names(names_series):
    def format_name(name):
        if name in ['Height', 'Width', 'Length']:
            return f'Product {name}[Product]'
        else:
            return f'{name}[Product]'
    return ', '.join(names_series.apply(format_name))

temp = attributes.groupby(['Taxonomy']).agg({
    'Attribute Name': process_attribute_names
}).reset_index()
```

Gambar 3.15. Fungsi *get_taxonomy*() dan *process_attribute_names*()

Fungsi tersebut kemudian diaplikasikan pada 3 *file* data, untuk kemudian disatukan dengan mengaplikasikan *script* seperti pada Gambar 3.15 diatas. fungsi *get_taxonomy*() digunakan untuk membuat kolom baru bernama *'Taxonomy'.* Fungsi ini mengambil nilai dari kolom 'Level 1', 'Level 2', 'Level 3', dan 'Level 4'

pada setiap baris DataFrame. fungsi *process_attribute_names*() bertugas memformat nama-nama atribut. Di dalamnya terdapat fungsi *format_name*() yang akan menambahkan kata 'Product' di depan nama atribut *'Height'*, '*Width'*, dan *'Length'*, serta menambahkan '[*Product*]' di akhir setiap nama atribut lainnya. Terakhir analisis kelengkapan atribut dilakukan dengan menerapkan fungsi *calculate_empty_and_poor_filled_percentage*() seperti pada Gambar 3.16 di bawah.

```
def calculate_empty_and_poor_filled_percentage(row, product_columns):
    # Check if product_columns is empty
    if not product_columns:
        return pd.Series({
            'Empty Percentage': 0,
            'Poor Filled Percentage': 0
        })

    # Convert row to a pandas Series if it's not already
    row_series = pd.Series(row)

    # Count empty or NaN values in product columns
    empty_count = row_series[product_columns].isna().sum() + (
        row_series[product_columns].apply(lambda x: isinstance(x, str) and len(x.strip()) == 0).sum()
    )

    empty_percentage = (empty_count / len(product_columns)) * 100 if product_columns else 0

    # Calculate poor-filled percentage for product columns
    poor_filled_values = {'.', '-', '0', 'no', 'none'}
    poor_filled_count = sum(
        1 for value in row_series[product_columns].values
        if isinstance(value, str) and value.strip().lower() in poor_filled_values
    )

    poor_filled_percentage = (poor_filled_count / len(product_columns)) * 100 if product_columns else 0

    return pd.Series({
        'Empty Percentage': empty_percentage,
        'Poor Filled Percentage': poor_filled_percentage
    })
```

Gambar 3.16 Fungsi *calcualte_empty_and_poor_filled_percentage*()

Fungsi *calculate_empty_and_poor_filled_percentage*() digunakan untuk menghitung persentase kolom kosong dan kolom yang kurang terisi dalam suatu baris DataFrame. Fungsi ini menerima dua parameter: *row*, yang merupakan data baris yang akan dianalisis, dan *product_columns*, yaitu daftar kolom produk yang ingin diperiksa. Pertama, fungsi mengecek apakah *product_columns* kosong; jika ya, maka akan mengembalikan persentase kosong dan kurang terisi sebagai 0. Selanjutnya, fungsi menghitung jumlah nilai kosong atau NaN dalam kolom produk serta menghitung persentase nilai-nilai tersebut. Kemudian, fungsi juga menghitung persentase kolom yang diisi dengan nilai kurang sesuai seperti '.', '-',

'0', '*no*', atau '*none*'. Hasil akhir berupa dua persentase: persentase kolom kosong dan persentase kolom yang kurang terisi.

3.2.7 Pengembangan Antarmuka *Website* untuk Otomasi *Request*

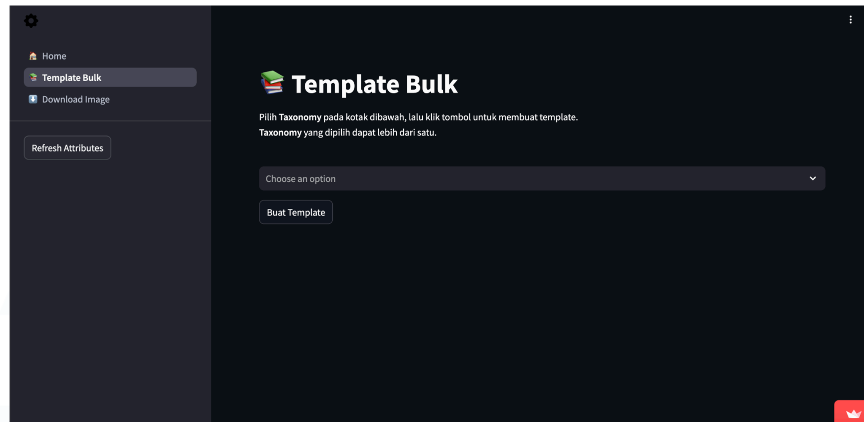
pekerjaan, seperti mengelola survei atau kuesioner, di mana setiap responden dapat diproses dan dianalisis lebih cepat. Selain itu, skrip dapat dikustomisasi untuk memenuhi kebutuhan spesifik, seperti validasi data dan pengaturan logika tertentu dalam form.

Dalam mendukung tim MDM *Officer* serta seluruh *user* untuk melakukan praktik bisnis secara efektif dan efisien, maka dari itu upaya otomasi melalui pengembangan antarmuka *website* dengan menggunakan Streamlit sebagai *tools* dilakukan. Adapun beberapa proses bisnis yang berhasil di otomasi dalam bentuk fitur pada *website* antara lain:

a. Fitur Pembuatan *Template Bulk*

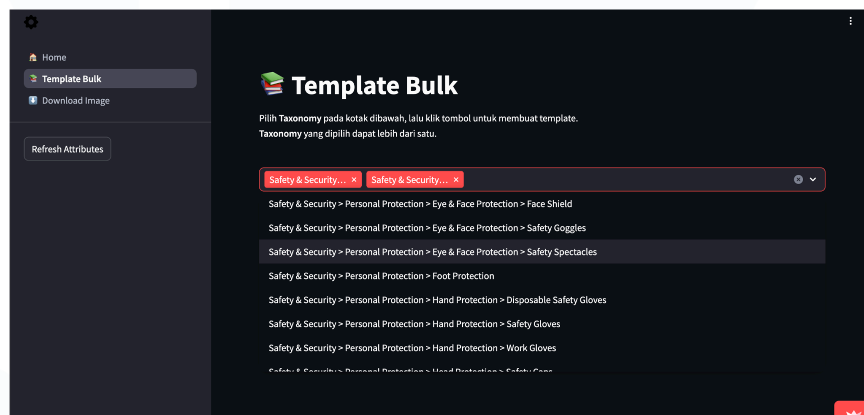
   Request terkait rekonstruksi *template bulk* yang dilakukan sebelumnya mampu menjawab permasalahan *user* dari sisi kemudahan melalui penyusunan data berdasarkan kewajiban pengisian serta *template* yang kini lengkap dengan atribut spesifik berdasarkan taksonomi. Namun, dalam praktik bisnisnya pembuatan artikel dalam jumlah besar seringkali melibatkan lebih dari satu jenis taksonomi produk. Maka dari itu, pengembangan fitur ini bertujuan untuk menyajikan *template bulk* yang bersifat dinamis bagi pengguna serta terbarukan menyesuaikan dengan taksonomi produk yang bertambah seiring waktu. Halaman *website* pada fitur pembuatan *template bulk* dapat dilihat pada Gambar 3.17 di bawah.

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.17 Halaman Utama Fitur *Template Bulk*

Pada halaman ini pengguna dapat memilih taksonomi yang diinginkan dengan menekan *dropdown*. Akan muncul seluruh tampilan dari taksonomi yang tersedia pada sistem seperti pada Gambar 3.18 di bawah.



Gambar 3.18. Tampilan *Dropdown* untuk Memilih Taksonomi

Setelah memilih taksonomi, pengguna kemudian dapat menekan tombol ‘buat *template*’. Sistem akan secara otomatis memproses pembuatan taksonomi dibelakang layar. Setelah proses selesai, maka akan muncul tombol untuk mengunduh seperti pada Gambar 3.19 di bawah.

Gambar 3.19. Tampilan Halaman Setelah *Template* Terbuat

Ketika pengguna menekan tombol untuk mengunduh, maka secara otomatis *template* akan langsung terunduh pada *browser*, adapun tampilan *template* berdasarkan taksonomi yang telah dipilih dapat dilihat pada Gambar 3.20 di bawah. *Template* berhasil terbuat menyesuaikan dengan taksonomi yang dipilih oleh pengguna, lengkap dengan atribut produk yang bersifat spesifik untuk setiap taksonomi yang direpresentasikan dengan tanda bintang. Penggunaan tanda bintang dilakukan untuk mencegah kebingungan pada sisi pengguna, serta sebagai tanda untuk atribut yang sama antar taksonomi.

Action	Product Na	SKU	Taxonomy I	Article desc	Gross Weig	Height	Length	Lower-lev. U	Unique Sell	Material	Color	Product For
			Fashion > Kids Accessories > Kids Jewelry						*	*	*	*
			Fashion > Kids Accessories > Kids Hair Accessories						*	*	*	
Default	Default	Default	Default	Basic Data	Basic Data	Basic Data	Basic Data	Basic Data	Product	Product	Product	Product
Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
Text + Num	Text + Num	Text + Num	Text + Num	Text + Num	Text + Num	Text + Num	Text + Num	Text + Num	Text	Text	Text	Text
	Panduan Pengisian											
	Pilihan data untuk pengisian dapat dilihat pada link dibawah:											
	Hapus baris panduan setelah pengisian.											

Gambar 3.20. Contoh *Template Bulk*

Untuk mengimplementasikan otomasi dalam pembuatan *template bulk*, penyusunan program dilakukan dengan menerapkan modularisasi kedalam *class* serta *method* untuk meningkatkan reprodusibilitas serta skalabilitas. Pertama inisiasi variabel konstan seperti pada Gambar 3.21 di bawah dilakukan. Variabel '*ACTIVITY*' merupakan enumerasi untuk menandakan tipe aktivitas yang digunakan yakni *template generator*. Variabel '*SEPARATED_BY*' diatur sebagai karakter *newline*, yang digunakan

sebagai pemisah antar baris teks. '*FILE_NAME*' diset sebagai 'attributes.csv', menunjukkan nama file CSV yang digunakan sebagai sumber atribut produk. '*SORTING_ORDER_FIELD*' berisi urutan numerik untuk berbagai kategori data seperti '*Basic Data*', '*Purchasing*', '*BOM*', dan seterusnya, yang masing-masing diberi nilai integer untuk menentukan prioritas urutannya. Sedangkan '*SORTING_ORDER_MANDATORY*' mengatur urutan berdasarkan pentingnya data.

```
ACTIVITY = Activity.TEMPLATE_GENERATOR
SEPARATED_BY = '\n'
FILE_NAME = 'attributes.csv'

SORTING_ORDER_FIELD = {
    'Default': 0,
    'Basic Data': 1,
    'Purchasing': 2,
    'BOM': 3,
    'Assortment Grade': 4,
    'Product': 5,
    'Buyer Info': 6,
    'Gideon Data': 7,
    'Images & Videos': 8,
    'PIMWorks System': 9
}

SORTING_ORDER_MANDATORY = {
    'Mandatory': 0,
    'Strongly Recommended': 1,
    'Good to Have': 2
}
```

Gambar 3.21. Inisiasi Variabel Konstan

Setelah inisiasi variabel konstan, pembuatan *class Attribute* seperti pada Gambar 3.22 di bawah. *Class Attribute* bertujuan untuk menghasilkan file CSV dari daftar atribut guna meningkatkan proses pengambilan informasi. *Class* ini menerima satu parameter *service*, yang mengacu kepada API atau layanan dari Google.

```
class Attribute:
    '''Generate Attribute CSV from Attribute List to enhance information retrieval'''
    def __init__(self, service) -> None:
        self.service=service
        self.file_name = FILE_NAME
        self.logger = Logger()
        pass
```

Gambar 3.22. *Class Attribute*

Class ini terdiri atas beberapa *method* atau fungsi. Fungsi pertama merupakan *_load_sheet*() yang ditampilkan pada gambar di bawah 3.21 di bawah. Fungsi ini merupakan metode statis yang bertujuan untuk memuat data dari Excel ke dalam DataFrame. Fungsi ini membaca file Excel berdasarkan nama sheet yang diberikan, dan mengubah semua kolom data menjadi tipe string. Selanjutnya, fungsi menambahkan kolom baru bernama '*Taxonomy*' yang nilainya dihasilkan dari fungsi *get_taxonomy* yang diterapkan pada setiap baris. Kolom '*Attribute Value Type*' dan '*Coverage*' diisi dengan nilai *default* '*Text*' jika kosong atau berisi nilai tidak valid seperti 'nan'. Baris yang tidak memiliki nilai di kolom '*Taxonomy*' akan dihapus dari DataFrame sebelum akhirnya mengembalikan DataFrame yang telah diproses.

```
@staticmethod
def _load_sheet(file, sheet_name):
    df = pd.read_excel(file, sheet_name)
    df = df.astype(str)

    #Assign new column Taxonomy
    df['Taxonomy'] = df.apply(lambda x: get_taxonomy(x), axis=1)

    df['Attribute Value Type'] = df['Attribute Value Type'].fillna('Text').replace({'nan': 'Text', '': 'Text'})
    df['Coverage'] = df['Coverage'].fillna('Text').replace({'nan': 'Text', '': 'Text'})

    df = df[df['Taxonomy'] != ''] #Drop row with empty Taxonomy
    return df
```

Gambar 3.23. Fungsi *_load_sheet*()

Berikutnya seperti pada Gambar 3.24 di bawah merupakan fungsi *_process_attribue_names*() yang juga merupakan metode statis. Fungsi ini digunakan untuk memproses nama atribut dengan menambahkan kata '*Product*', di depan nama atribut tertentu. Jika nama atribut adalah '*Height*', '*Width'*, atau '*Length*', dan hanya berisi nama tersebut, maka fungsi akan mengembalikan nama atribut dengan tambahan '*Product*' di depannya,

misalnya menjadi '*Product Height*'. Jika nama atribut berbeda dari ketiga nama tersebut, maka nama atribut akan dikembalikan tanpa perubahan.

```
@staticmethod
def _process_attribute_names(attribute_name):
    # Add 'Product' before 'Height', 'Width', 'Length' if they are the only content in the attribute name
    if attribute_name in ['Height', 'Width', 'Length']:
        return f'Product {attribute_name}'
    return attribute_name
```

Gambar 3.24. Fungsi *_process_attribute_names*()

Fungsi terakhir pada *class Attribute* merupakan *generate*() seperti pada Gambar 3.25 di bawah. Fungsi ini digunakan untuk menghasilkan dan mengunggah daftar atribut dari file Excel ke Google Drive. Metode ini memulai dengan mengambil path file Excel dari konfigurasi dan memuat seluruh sheet di dalamnya. Untuk setiap *sheet*, data dibaca menggunakan fungsi *_load_sheet*() dari kelas *Attribute*. Kemudian, nama atribut diproses melalui metode *_process_attribute_names*() untuk menyesuaikan nama atribut tertentu. Data dikelompokkan berdasarkan kolom '*Taxonomy*' dan nilai-nilai di kolom '*Attribute Name*', '*Coverage*', dan '*Attribute Value Type*' digabungkan menggunakan pemisah yang telah ditentukan pada variabel konstan *SEPARATED_BY*. Hasil pengolahan ini kemudian digabungkan ke dalam satu DataFrame. DataFrame yang telah digabungkan diekspor menjadi file CSV dalam bentuk buffer dan diunggah ke folder Google Drive yang ditentukan dengan menggunakan fungsi *upload_to_drive*(). Fungsi ini juga mengembalikan DataFrame akhir yang telah diproses.

```python
def generate(self):
    file_path = get_config(ACTIVITY).get('attribute_list_url')
    file = pd.ExcelFile(file_path)

    df_attributes = pd.DataFrame()
    for sheet in file.sheet_names:
        df = Attribute._load_sheet(file, sheet)

        #Rename certain value
        df['Attribute Name'] = df['Attribute Name'].apply(self._process_attribute_names)

        temp = df.groupby(['Taxonomy']).agg({
            'Attribute Name': SEPARATED_BY.join,
            'Coverage': SEPARATED_BY.join,
            'Attribute Value Type': SEPARATED_BY.join,
            }).reset_index()

        df_attributes = pd.concat([df_attributes, temp])

    buffer = io.BytesIO()
    df_attributes.to_csv(buffer, index=False)
    buffer.seek(0)

    folder_id = get_config(ACTIVITY).get('references_folder_id')
    upload_to_drive(self.logger, self.service,
                    folder_id, self.file_name,
                    buffer, MimeType.CSV.value,
                    replace=True)

    return df_attributes
```

Gambar 3.25. Fungsi *generate*()

Berikutnya merupakan *class Template*, *class* ini bertujuan untuk menyiapkan *template* sesuai dengan format yang telah ditentukan. Seperti pewarnaan baris menyesuaikan dengan nilai, penggabungan baris, hingga penyortiran kolom.

```python
class Template:
    def __init__(self, selected_taxonomies):
        self.selected_taxonomies = selected_taxonomies
        self.num_rows = len(self.selected_taxonomies)
        self.df = self._load_template()
        pass
```

Gambar 3.26. *Class Template*

Class ini terdiri atas beberapa metode atau fungsi. Fungsi pertama seperti pada Gambar 3.27 di bawah merupakan fungsi statis, *get_color_cell*(). Fungsi ini menggunakan pemetaan warna yang telah ditentukan, di mana nilai '*Mandatory*' akan mengembalikan gaya latar belakang berwarna

merah, '*Strongly Recommended*' akan berwarna oranye, dan '*Good to have*' akan berwarna kuning. Jika nilai yang diberikan tidak sesuai dengan salah satu dari tiga kategori tersebut, metode ini akan mengembalikan string kosong, sehingga tidak ada perubahan gaya pada sel tersebut.

```
@staticmethod
def get_color_cell(value):
    color_map = {
        'Mandatory': 'background-color: red',
        'Strongly Recommended': 'background-color: orange',
        'Good to have': 'background-color: yellow',
    }

    return color_map.get(value, '')
```

Gambar 3.27. Fungsi *get_color_cell*()

Gambar 3.28 di bawah merupakan fungsi *add_row*(). Fungsi ini merupakan metode statis yang digunakan untuk menambahkan baris kosong ke dalam DataFrame yang sudah ada. Fungsi ini membuat baris tambahan sebanyak '*num_rows*' dengan menggunakan struktur kolom yang sama seperti DataFrame asli. Baris-baris kosong ini kemudian digabungkan di bagian atas DataFrame asli. Setelah digabungkan, indeks DataFrame diperbarui dan diatur ulang agar berurutan tanpa memperhatikan indeks lama. Akhirnya, DataFrame yang telah diperbarui dengan baris tambahan dikembalikan sebagai hasil dari fungsi ini.

```
@staticmethod
def add_row(df, num_rows):
    additional_row = pd.DataFrame(index=range(num_rows), columns=df.columns)
    df_updated = pd.concat([additional_row, df], ignore_index=True)
    df_updated.reset_index(drop=True, inplace=True)

    return df_updated
```

Gambar 3.28. Fungsi *add_row*()

Gambar 3.29 di bawah merupakan fungsi statis terakhir pada *class template*, *sort_column*(). Fungsi ini digunakan untuk mengurutkan kolom-kolom dalam DataFrame berdasarkan urutan yang telah ditentukan.

Metode ini pertama-tama mentransposisikan DataFrame sehingga kolom menjadi baris. Kemudian, dua kolom baru ditambahkan: '*mandatory_order*' dan '*field_order*', yang berisi nilai urutan dari *mapping* '*SORTING_ORDER_MANDATORY*' dan '*SORTING_ORDER_FIELD*' berdasarkan nilai di baris tertentu. Jika nilai tidak ada dalam *mapping*, diisi dengan *float('inf')* untuk memposisikannya di akhir urutan. DataFrame kemudian diurutkan berdasarkan '*mandatory_order*' dan '*field_order*'. Setelah pengurutan, kolom '*mandatory_order*' dan '*field_order*' dihapus, dan DataFrame ditransposisikan kembali ke bentuk semula sebelum dikembalikan.

```
@staticmethod
def sort_column(df, num_rows):
    df_t = df.T.copy()
    df_t['mandatory_order'] = df_t[num_rows+1].map(
        SORTING_ORDER_MANDATORY).fillna(float('inf'))

    df_t['field_order'] = df_t[num_rows].map(
        SORTING_ORDER_FIELD).fillna(float('inf'))

    # Sort by the defined orders and index, then drop the columns
    df_t.sort_values(by=['mandatory_order', 'field_order'], inplace=True)
    df_t.drop(columns=['mandatory_order', 'field_order'], inplace=True)

    return df_t.T
```

Gambar 3.29. Fungsi *sort_column*()

Gambar 3.30 di bawah merupakan fungsi *_insert_taxonomy*(). Fungsi ini digunakan untuk menyisipkan nilai-nilai taksonomi yang dipilih ke dalam DataFrame pada kolom '*Taxonomy Node*'. Metode ini melakukan iterasi melalui indeks dan nilai-nilai dalam atribut '*selected_taxonomies*', lalu menempatkan setiap nilai taksonomi ke dalam baris yang sesuai pada kolom '*Taxonomy Node*'. Dengan demikian, setiap nilai taksonomi dimasukkan ke baris DataFrame berdasarkan indeksnya masing-masing. Setelah semua nilai taksonomi disisipkan, DataFrame yang telah diperbarui dikembalikan sebagai hasil dari fungsi ini.

```
def _insert_taxonomy(self, df):
    for index, taxonomy in enumerate(self.selected_taxonomies):
        df.at[index, 'Taxonomy Node'] = taxonomy

    return df
```

Gambar 3.30. Fungsi *_insert_taxonomy*()

Gambar 3.31 merupakan fungsi *_load_template*(), fungsi ini digunakan untuk memuat dan memproses template dari file Excel. Metode ini dimulai dengan mengambil path file template dari konfigurasi menggunakan *get_config*(), kemudian membaca file Excel tersebut ke dalam sebuah DataFrame. Setelah itu, metode *add_row*() digunakan untuk menambahkan sejumlah baris kosong ke dalam template sesuai dengan jumlah baris yang ditentukan oleh jumlah baris. Selanjutnya, metode *_insert_taxonomy*() dipanggil untuk menyisipkan nilai-nilai taksonomi yang telah dipilih ke dalam DataFrame yang telah ditambahkan baris. Akhirnya, DataFrame yang telah diperbarui dengan baris tambahan dan nilai-nilai taksonomi dikembalikan sebagai hasil dari fungsi ini.

```
def _load_template(self):
    file_path = get_config(ACTIVITY).get('template_url')
    template = pd.read_excel(file_path)

    template_added_row = Template.add_row(template, self.num_rows)
    template_inserted = self._insert_taxonomy(template_added_row)

    return template_inserted
```

Gambar 3.31. Fungsi *_load_template*()

Gambar 3.32 merupakan fungsi sederhana *load*(). Fungsi ini bertujuan untuk mengembalikan variabel DataFrame pada *class Template*.

```
def load(self):
    return self.df
```

Gambar 3.32. Fungsi *load*()

Fungsi terakhir pada *class Template* merupakan *update*() seperti pada Gambar 3.33 di bawah. Fungsi ini digunakan untuk memperbarui template dengan menggabungkan DataFrame template dengan DataFrame atribut produk. Metode ini menggabungkan kedua DataFrame secara horizontal (kolom) dan mereset indeks hasil penggabungan agar berurutan. Setelah itu, DataFrame yang digabungkan diurutkan menggunakan metode *sort_column*() pada *class Template* berdasarkan urutan yang telah ditentukan, dengan mempertimbangkan jumlah baris. Langkah terakhir adalah menerapkan pewarnaan sel pada DataFrame yang telah diurutkan menggunakan metode *get_color_cell*() pada *class Template* untuk menentukan warna latar belakang sel berdasarkan nilainya. Hasil akhirnya adalah DataFrame yang telah diperbarui, diurutkan, dan diberi warna, yang kemudian dikembalikan sebagai hasil dari fungsi ini.

```
def update(self, df_product):
    '''Update Template combined with Product Attributes Dataframe'''

    template_updated = pd.concat([self.df, df_product], axis=1)
    template_updated.reset_index(drop=True, inplace=True)
    template_sorted = Template.sort_column(template_updated, self.num_rows)
    template_colored = template_sorted.style.map(Template.get_color_cell)
    return template_colored
```

Gambar 3.33. Fungsi *update*()

Class terakhir sekaligus *class* yang nantinya digunakan pada pengembangan antarmuka untuk menghasilkan *template* menyesuaikan dengan *taxonomy* yang dipilih merupakan *class TemplateGenerator*.

```
class TemplateGenerator:
    def __init__(self) -> None:
        pass
```

Gambar 3.34. *Class TemplateGenerator*

Gambar 3.35 di bawah merupakan fungsi pertama pada *class* ini, fungsi *get_taxonomy_list*(). Fungsi ini merupakan metode statis yang digunakan untuk mendapatkan daftar taksonomi dari template. Fungsi ini mengembalikan daftar taksonomi dengan memeriksa apakah kolom 'Taxonomy' memiliki metode *tolist*(). Jika ya, ia mengonversi kolom tersebut menjadi sebuah daftar menggunakan *tolist*(), dan jika tidak, ia mengonversi kolom tersebut menjadi daftar dengan fungsi *list*(). Hasilnya adalah daftar taksonomi yang diambil dari daftar atribut.

```
@staticmethod
def get_taxonomy_list():
    # Load the attribute list from the Template
    attribute_list = TemplateGenerator.load_attribute_list()

    # Return the list of taxonomies
    return attribute_list['Taxonomy'].tolist() if hasattr(attribute_list['Taxonomy'], 'tolist') else list(attribute_list['Taxonomy'])
```

Gambar 3.35. Fungsi *get_taxonomy_list*()

Berikutnya pada Gambar 3.36 di bawah merupakan fungsi *rename_col_duplicate*(). Fungsi ini juga merupakan metode statis yang digunakan untuk mengubah nama kolom yang duplikat antara dua DataFrame dengan menambahkan sebuah sufiks pada kolom-kolom yang duplikat. Pertama, fungsi ini mengambil nama kolom dari '*df_source*' dan mencari kolom yang terdapat di kedua DataFrame tersebut. Kemudian, dibuat pemetaan nama kolom yang akan diubah dengan menambahkan '*suffix*' pada kolom duplikat. Nama kolom pada DataFrame kemudian diubah sesuai dengan pemetaan ini menggunakan fungsi *rename*(). Akhirnya, DataFrame dengan nama kolom yang telah diperbarui dikembalikan sebagai hasil dari fungsi ini.

```python
@staticmethod
def rename_col_duplicate(df, df_source, suffix=' Product'):
    template_cols = df_source.columns
    cols_to_replace = set(df.columns).intersection(set(template_cols))
    column_mapping = {col: col + suffix for col in cols_to_replace}

    df.rename(columns=column_mapping, inplace=True)
    return df
```

Gambar 3.36. Fungsi *rename_col_duplicate*()

Gambar 3.37 di bawah merupakan fungsi statis terakhir pada *class TemplateGenerator*. Fungsi ini digunakan untuk memuat daftar atribut dari sebuah file CSV. Fungsi ini pertama-tama mengambil *path* file CSV dari konfigurasi menggunakan *get_config*() dengan kunci '*attribute_csv_url*' di dalam aktivitas tertentu. Setelah mendapatkan *path* file, metode ini membaca file CSV tersebut ke dalam sebuah DataFrame. DataFrame yang berisi daftar atribut ini kemudian dikembalikan sebagai hasil dari fungsi ini.

```python
@staticmethod
def load_attribute_list():
    file_path = get_config(ACTIVITY).get('attribute_csv_url')
    attribute_list = pd.read_csv(file_path)
    return attribute_list
```

Gambar 3.37. Fungsi *load_attribute_list*()

Berikutnya merupakan fungsi *_get_df_explode*() seperti pada Gambar 3.38 di bawah. Fungsi ini digunakan untuk memproses dan meledakkan (*explode*) kolom-kolom tertentu dalam DataFrame berdasarkan pemisah yang telah ditentukan. Metode ini memeriksa apakah kolom-kolom yang dibutuhkan ('*Attribute Name*', '*Coverage*', '*Attribute Value Type*') ada dalam DataFrame, dan jika salah satu dari kolom tersebut hilang, maka akan mengembalikan peringatan *error*. Setelah memastikan kolom-kolom yang diperlukan ada, metode ini mengubah tipe data dari kolom-kolom tersebut menjadi string dan memisahkannya menjadi daftar berdasarkan pemisah yang ditentukan oleh varibael konstan '*SEPARATED_BY*'. Kemudian,

fungsi *explode*() digunakan untuk memperluas DataFrame dengan mengulang setiap elemen daftar di kolom-kolom tersebut sehingga setiap kombinasi atribut, cakupan, dan tipe nilai atribut muncul sebagai baris terpisah. Hasilnya adalah DataFrame yang telah diekspansi berdasarkan nilai-nilai dalam kolom yang diproses.

```
def _get_df_explode(self, df):
    required_columns = ['Attribute Name', 'Coverage', 'Attribute Value Type']
    for col in required_columns:
        if col not in df.columns:
            raise ValueError(f"Column {col} is missing from the DataFrame")

    df.loc[:, 'Attribute Name'] = df['Attribute Name'].astype(str).str.split(SEPARATED_BY)
    df.loc[:, 'Coverage'] = df['Coverage'].astype(str).str.split(SEPARATED_BY)
    df.loc[:, 'Attribute Value Type'] = df['Attribute Value Type'].astype(str).str.split(SEPARATED_BY)

    return df.explode(['Attribute Name', 'Coverage', 'Attribute Value Type'])
```

Gambar 3.38. Fungsi *_get_df_explode*()

Gambar 3.39 di bawah merupakan fungsi *_get_df_product*() yang digunakan untuk membentuk sebuah DataFrame baru berdasarkan atribut unik dan taksonomi yang terdapat dalam DataFrame awal. Pertama, metode ini mengambil daftar atribut unik dari kolom '*Attribute Name*'. Selanjutnya, dibentuk sebuah struktur DataFrame baru dengan kolom '*Taxonomy*' dan semua atribut unik tersebut sebagai kolom tambahan. Kemudian, untuk setiap taksonomi unik, metode ini membuat baris baru dengan mencocokkan atribut yang ada dan memberikan tanda '*' jika atribut tersebut ada pada taksonomi terkait, atau membiarkannya kosong jika tidak ada. Setelah semua baris dibentuk, DataFrame tersebut dikombinasikan dengan struktur DataFrame awal. Fungsi ini kemudian menambahkan tiga baris terakhir secara dinamis untuk menunjukkan informasi umum ('*Product*'), cakupan ('*Coverage*'), dan tipe nilai atribut ('*Attribute Value Type*'). Baris '*Product*' diisi dengan nilai '*Product*' untuk semua kolom, sementara baris '*Coverage*' dan '*Attribute Value Type*' diisi dengan nilai yang sesuai dari DataFrame awal berdasarkan nama atribut

yang bersesuaian. Kedua baris terakhir ini diisi dengan nilai pertama yang ditemukan untuk masing-masing atribut. Akhirnya, ketiga baris ini digabungkan ke dalam sebuah DataFrame, dan kolom pertama ('*Taxonomy*') dihapus sebelum mengembalikan DataFrame yang sudah diperbarui.

```
def _get_df_product(self, df):
    unique_attributes = df['Attribute Name'].unique().tolist()

    # Create a new DataFrame structure
    columns = ['Taxonomy'] + unique_attributes
    df_product = pd.DataFrame(columns=columns)

    # Populate the new DataFrame
    rows = []
    for taxonomy in df['Taxonomy'].unique():
        temp_df = df[df['Taxonomy'] == taxonomy]
        row = {'Taxonomy': taxonomy}
        for attr in unique_attributes:
            if attr in temp_df['Attribute Name'].values:
                row[attr] = '*'
            else:
                row[attr] = ''
        rows.append(row)

    df_product = pd.concat([df_product, pd.DataFrame(rows)], ignore_index=True)

    # Adding the last two rows for Coverage and Attribute Value Type dynamically
    product_row = {col: 'Product' for col in df_product.columns} #Assign product Row
    coverage_row = {col: '' for col in df_product.columns}
    attribute_value_type_row = {col: '' for col in df_product.columns}

    for attr in unique_attributes:
        coverage_value = df[df['Attribute Name'] == attr]['Coverage'].values
        attribute_value_type_value = df[df['Attribute Name'] == attr]['Attribute Value Type'].values
        if len(coverage_value) > 0:
            coverage_row[attr] = coverage_value[0]
        if len(attribute_value_type_value) > 0:
            attribute_value_type_row[attr] = attribute_value_type_value[0]

    lower_row_df = pd.DataFrame([product_row, coverage_row, attribute_value_type_row])
    df_product = pd.concat([df_product, lower_row_df], axis=0, ignore_index=True).iloc[:, 1:]
    return df_product
```

Gambar 3.39. Fungsi *_get_df_product*()

Terakhir merupakan fungsi *generate*() yang mengagregat seluruh fungsi lainnya dalam *class TemplateGenerator* seperti pada Gambar 3.40 di bawah. Fungsi ini digunakan untuk menghasilkan sebuah *template* berdasarkan taksonomi yang dipilih. Pertama, fungsi ini memuat daftar atribut dengan memanggil fungsi *load_attribute_list*() dan menyaring atribut yang sesuai dengan taksonomi yang dipilih. Data yang terpilih kemudian diproses dengan mengimplementasi fungsi *_get_df_explode*() untuk meledakkan kolom tertentu sehingga setiap nilai menjadi baris

terpisah. Selanjutnya, fungsi *_get_df_product*() digunakan untuk membentuk DataFrame produk berdasarkan atribut yang telah diolah. Sebuah instance dari *class Template* dibuat dengan memberikan taksonomi yang dipilih, dan template awal dimuat menggunakan fungsi *load*(). Fungsi *rename_col_duplicate*() kemudian digunakan untuk mengganti nama kolom yang duplikat antara DataFrame produk dan *template*, dengan menambahkan sufiks pada kolom duplikat. Terakhir, *template* yang telah diperbarui dengan data produk kemudian di-update menggunakan fungsi *update*() pada *class Template*, yang menggabungkan DataFrame produk yang sudah diolah dengan template awal. Hasil akhir berupa template yang telah di-update sesuai dengan atribut dan taksonomi yang dipilih, dan hasil ini dikembalikan sebagai keluaran dari fungsi ini.

```
def generate(self, selected_taxonomies):
    attribute_list = TemplateGenerator.load_attribute_list()
    selected = attribute_list[attribute_list['Taxonomy'].isin(selected_taxonomies)]
    exploded = self._get_df_explode(selected)

    df_product = self._get_df_product(exploded)
    TemplateEngine = Template(selected_taxonomies)
    template = TemplateEngine.load()

    df_product_renamed = TemplateGenerator.rename_col_duplicate(df_product, template)

    template = TemplateEngine.update(df_product_renamed)
    return template
```

Gambar 3.40. Fungsi *generate*()

Berikutnya integrasi *class TemplateGenerator* kedalam pengembangan antarmuka menggunakan *streamlit* dilakukan. Integrasi diawali dengan melakukan *import* pada *class* ataupun *library* yang dibutuhkan serta pemuatan *cache* untuk menginisiasi *TemplateGenerator* serta *list* dari *taxonomy* yang tersedia seperti pada Gambar 3.41 di bawah.

```python
from utils.drive import authenticate
from utils.activity.template_generator import TemplateGenerator, Attribute
from pandas.io.formats import excel
import streamlit as st
import tempfile

excel.ExcelFormatter.header_style=None

@st.cache_resource
def load_data():
    generator = TemplateGenerator()
    taxonomy_list = generator.get_taxonomy_list()
    return generator, taxonomy_list

# Initialize Generator and taxonomy_list
Generator, taxonomy_list = load_data()
```

Gambar 3.41. Pemuatan *Library* dan Inisiasi *Cache*

Memastikan keterbaruan daftar *taxonomy*, maka penambahan *button* yang berfungsi untuk memuat ulang *list* dari *taxonomy* terbaru perlu dilakukan. Pembuatan *button refresh* ini dilakukan seperti pada Gambar 3.42 di bawah. Ketika *button* ditekan, aplikasi menampilkan *spinner* untuk menunjukkan bahwa proses pembaruan sedang berlangsung. Fungsi *authenticate*() digunakan untuk melakukan autentikasi dengan menggunakan kredensial dari akun layanan Google yang disimpan di *file secrets streamlit*. Setelah berhasil autentikasi, data atribut diperbarui dengan memanggil metode *generate*() dari kelas *Attribute*, yang mengambil data atribut terbaru. Daftar taksonomi unik dari data atribut tersebut kemudian disimpan ke dalam *session_state* bernama *taxonomy_list*. Setelah proses selesai, pesan sukses ditampilkan kepada pengguna untuk mengindikasikan bahwa pembaruan telah berhasil dilakukan.

```
if st.sidebar.button("Refresh Attributes"):
    with st.spinner("Refreshing attributes..."):
        # Refresh the data
        service = authenticate(st.secrets['google_service_account'])
        attribute_data = Attribute(service).generate()
        st.session_state.taxonomy_list = attribute_data['Taxonomy'].unique().tolist()
        st.success('Attributes refreshed successfully!')

# Set default session state if not present
if 'taxonomy_list' not in st.session_state:
    st.session_state.taxonomy_list = taxonomy_list
```

Gambar 3.42. Pembuatan *Button Refresh*

Selanjutnya daftar *taxonomy* yang telah disimpan pada *session_state* dengan nama variabel *taxonomy_list* kemudian dimasukan kedalam *dropdown* dengan mengimplementasi *multiselect* seperti pada Gambar 3.43. Pembuatan *dropdown* dirancang agar pengguna dapat memilih lebih dari satu *taxonomy* sesuai dengan kebutuhan.

```
st.markdown("""
Pilih **Taxonomy** pada kotak dibawah, lalu klik tombol untuk membuat template.\\
**Taxonomy** yang dipilih dapat lebih dari satu.
""")

# Dropdown to select taxonomies
selected_taxonomies = st.multiselect("Pilih Taksonomi", st.session_state.taxonomy_list, label_visibility='hidden')
```

Gambar 3.43. Pembuatan *Multiselect* untuk *Taxonomy*

Terakhir pada pengembangan antarmuka *template bulk* sekaligus komponen yang memegang peranan penting merupakan *button* untuk menjalankan pembuatan *template* seperti pada Gambar 3.44 di bawah. Ketika tombol pembuatan *template* ditekan, aplikasi pertama-tama memeriksa apakah ada taksonomi yang telah dipilih oleh pengguna. Jika ada, proses pembuatan template dimulai dan *spinner* dengan pesan "Sedang membuat template..." akan ditampilkan selama *template* sedang dibuat oleh sistem. Disini peranan *class TemplateGenerator* dengan fungsi *generate*() digunakan untuk membuat template berdasarkan taksonomi yang dipilih. Template ini kemudian disimpan sebagai file Excel sementara menggunakan *library tempfile* dengan ekstensi .xlsx. Setelah proses pembuatan template selesai, aplikasi menampilkan notifikasi dengan pesan "Template berhasil dibuat!" beserta ikon centang sebagai

konfirmasi kepada pengguna. Kemudian muncul *button* untuk melakukan pengunduhan *template* yang telah berhasil terbuat. Jika ditekan maka pengunduhan file Excel yang baru saja dibuat dengan nama "template.xlsx" akan terjadi. Jika tidak ada taksonomi yang dipilih, aplikasi akan menampilkan pesan kesalahan yang meminta pengguna untuk memilih setidaknya satu taksonomi agar dapat melanjutkan proses pembuatan template.

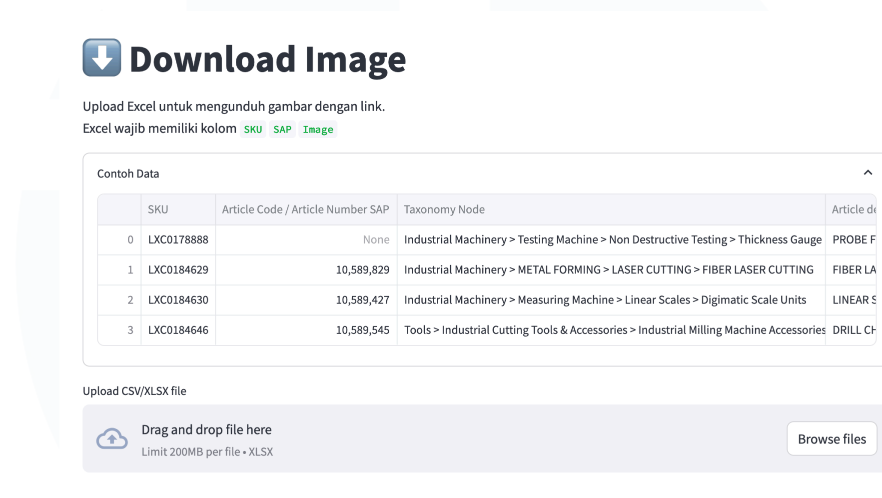
```
# Button to generate and download the template
if st.button("Buat Template"):
    if selected_taxonomies:
        with st.spinner("Sedang membuat template..."):
            # Generate the template
            template = Generator.generate(selected_taxonomies)

            with tempfile.NamedTemporaryFile(delete=False, suffix='.xlsx') as tmp:
                output = tmp.name
                template.to_excel(output, index=False)

            st.toast('Template berhasil dibuat!', icon='✅')

            st.divider()
            st.markdown("""Klik untuk download 👇""")
            with open(output, "rb") as file:
                st.download_button(
                    label="Unduh Template",
                    data=file,
                    file_name="template.xlsx",
                    mime="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
                )

    else:
        st.error("Pilih setidaknya satu taksonomi.")
```

Gambar 3.44. Pembuatan *Button* Pembuatan dan Pengunduhan

b. Fitur *Link* menjadi Gambar

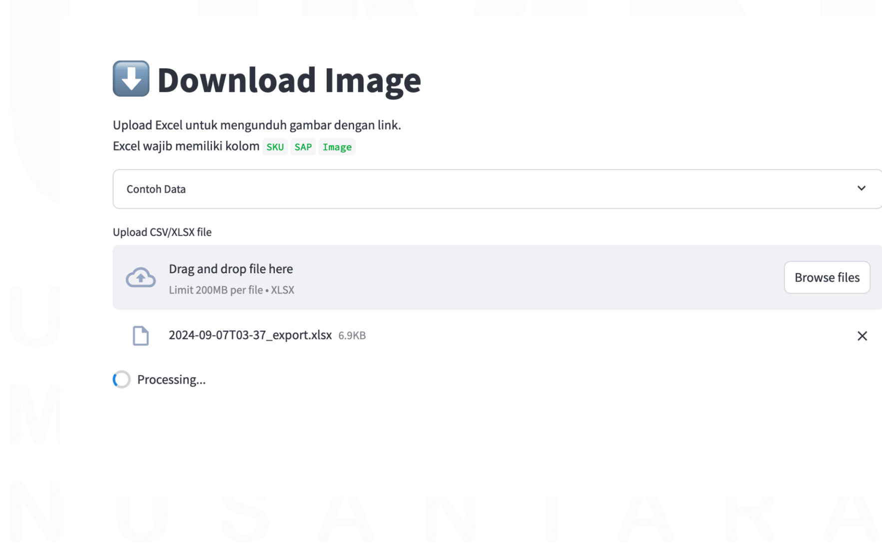
Selain daripada pembuatan *template bulk* menyesuaikan dengan taksonomi yang dipilih oleh pengguna, fitur lainnya merupakan pengunduhan gambar secara otomatis dengan penamaan yang disesuaikan. Fitur ini diperuntukan untuk menjawab permasalahan migrasi gambar yang dialami oleh tim MDM *officer*. Saat ini migrasi gambar dari *Product Information Management* (PIM) ke dalam sistem SAP masih dilakukan secara manual. Proses migrasi manual dilakukan dengan mengunduh gambar satu-per-satu, kemudian mengganti nama gambar menyesuaikan dengan kode artikel pada SAP. Maka dari itu pengembangan fitur ini ke dalam

antarmuka *website* dapat meningkatkan efisiensi serta produktivitas tim MDM *officer* untuk berfokus pada pekerjaan yang krusial serta mengeliminasi pekerjaan yang bersifat *non-value added*. Adapun tampilan antarmuka pada fitur ini dapat dilihat pada Gambar 3.45 di bawah.



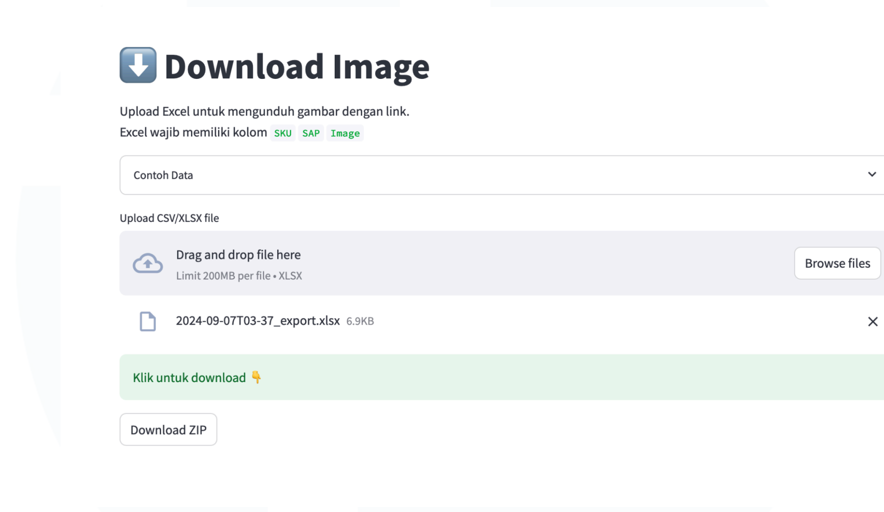
Gambar 3.45. Halaman Utama Fitur Unduh Gambar

Untuk melakukan pengunduhan gambar, pengguna dapat menarik atau memilih *file* dalam format Excel pada kotak yang telah disediakan. Adapun format data yang diterima harus menyesuaikan dengan contoh data pada tabel. Ketika pengguna memasukan gambar maka proses pengunduhan akan secara otomatis dilakukan dan tampilan halaman akan seperti pada Gambar 3.46 di bawah.



Gambar 3.46. Tampilan Saat Pemrosesan *File*

Setelah proses pengunduhan selesai, maka akan muncul notifikasi serta *button* untuk melakukan pengunduhan gambar dalam ekstensi ZIP seperti pada Gambar 3.47 di bawah.



Gambar 3.47. Tampilan Setelah *File* Berhasil Diproses

File dengan ekstensi ZIP yang berhasil terunduh akan terdiri atas gambar dengan ukuran yang telah diseragamkan menjadi persegi 700x700 dengan penamaan merupakan kode artikel SAP atau kode artikel pada PIM seperti pada Gambar 3.48 di bawah.

Images

10589427_3.jpg 10589545_4.jpg

10589829_2.jpg LXC0178888_1.jpg

Gambar 3.48. Contoh Hasil Pengunduhan

Serupa dengan fitur pembuatan *template bulk,* pengembangan fitur pengunduhan gambar ini turut menerapkan modularisasi berupa pengembangan *class* dan fungsi untuk mempertahankan reprodusibilitas program. *Class* penunjang pertama merupakan *ImageFormatter* seperti pada Gambar 3.49 di bawah. *Class* ini bertujuan untuk memproses data gambar baik dari sisi ukuran, format, dan lainnya.

```
class ImageFormatter:
    def __init__(self, service=None, data:pd.DataFrame=None, folder_link:str=None):
        self.service = service
        self.data = data
        self.folder_id = get_folder_id(folder_link) if folder_link else None
```

Gambar 3.49. *Class ImageFormatter*

Class ImageFormatter hanya terdiri atas satu fungsi statis yakni *process_image*() seperti pada Gambar 3.50 di bawah. Fungsi ini digunakan untuk memproses gambar, mengonversi nya ke ukuran target 700x700 piksel, dan memastikan aspek rasio tetap terjaga. Gambar dimuat dan

diubah menjadi mode "RGBA" untuk memungkinkan transparansi. Jika gambar tidak berbentuk persegi, metode ini membuat latar belakang putih dengan ukuran target, mengubah ukuran gambar sambil menjaga rasio aspek, dan menempatkannya di tengah latar belakang. Jika gambar sudah persegi, gambar langsung diubah ukurannya ke ukuran target. Setelah itu, gambar diubah ke mode "RGB" untuk menghilangkan transparansi dan disimpan dalam format JPEG. Gambar yang telah diproses dikembalikan sebagai aliran data (*stream*) yang dapat digunakan lebih lanjut, misalnya untuk diunggah atau disimpan.

```
@staticmethod
def process_image(image_file, target_size=(700, 700)):
    with Image.open(image_file) as img:
        img = img.convert("RGBA")

        if img.size[0] != img.size[1]:
            # Create a white background
            background = Image.new("RGBA", target_size, (255, 255, 255, 255))

            # Resize the image while maintaining the aspect ratio
            base_width = target_size[0]
            wpercent = (base_width / float(img.size[0]))
            hsize = int((float(img.size[1]) * float(wpercent)))
            img = img.resize((base_width, hsize), Image.Resampling.LANCZOS)

            # Calculate position to paste the image onto the background
            img_w, img_h = img.size
            bg_w, bg_h = background.size
            offset = ((bg_w - img_w) // 2, (bg_h - img_h) // 2)
            background.paste(img, offset)
            img = background
        else:
            # If already square, resize directly
            img = img.resize(target_size, Image.LANCZOS)

        # Convert image to "RGB"
        img = img.convert("RGB")

        # Save image to a BytesIO object
        output = io.BytesIO()
        img.save(output, format='JPEG')
        output.seek(0)

        return output
```

Gambar 3.50. Fungsi *process_image*()

Class berikutnya merupakan *ImageDownloader* seperti pada Gambar 3.51 di bawah. *Class* ini bertujuan untuk mengelola *link* pada DataFrame untuk kemudian mengunduh masing-masing dari setiap gambar kedalam *file*

dengan ekstensi ZIP. *Class* ini juga memungkinkan penyimpanan gambar pada Google Drive.

```
class ImageDownloader:
    def __init__(self, service, data: pd.DataFrame, save_to_drive=True):
        self.data = ImageDownloader.validate(data)
        self.service = service
        self.temp_dir = tempfile.mkdtemp()
        self.save_to_drive= save_to_drive
        self.logger = Logger()
        self.image_formatter = ImageFormatter()

        if self.save_to_drive:
            self.folder_id = create_folder(self.service, ACTIVITY.value,
                                           get_config(ACTIVITY).get('parent_folder_id'))
            self.folder_result_id = create_folder(self.service, ACTIVITY.value, self.folder_id,
                                                  "Result")
```

Gambar 3.51. *Class ImageDownloader*

Fungsi *validate*() merupakan metode statik pada *class ImageDownloader* seperti pada Gambar 3.52. Fungsi ini memeriksa apakah DataFrame memiliki kolom yang mengandung kata kunci tertentu, dalam hal ini 'SAP' atau '*IMAGE*'. Metode ini mencari kolom yang namanya mengandung salah satu dari kata kunci tersebut (dengan mengabaikan huruf besar/kecil). Jika tidak ada kolom yang cocok, metode ini akan mengembalikan *error* yang memberi tahu pengguna bahwa tidak ada kolom yang mengandung salah satu dari kata kunci yang diinginkan. Jika ditemukan kolom yang cocok, metode ini mengambil kolom pertama yang sesuai dan menggabungkan nilainya dengan kolom 'SKU' dengan mengisi nilai kosong di kolom pertama menggunakan data dari 'SKU'. Nilai-nilai ini diubah menjadi string, dan '.0' dihilangkan dari akhir teks. Setelah validasi selesai, metode ini mencetak pesan konfirmasi dan mengembalikan DataFrame yang hanya mencakup kolom-kolom yang cocok.

```python
@staticmethod
def validate(data):
    """Check if certain columns exist."""
    keywords = ['SAP', 'IMAGE']
    matching_columns = [
        col for col in data.columns
        if any(keyword in col.upper() for keyword in keywords)
    ]

    if not matching_columns:
        raise ValueError(f'No columns containing any of the keywords: {keywords} in DataFrame.')

    # Combine SKU columns to SAP
    sap_col = matching_columns[0]
    data[sap_col] = data[sap_col].fillna(data['SKU']).astype('str').str.replace(
        '.0', '', regex=False
    )

    print("Data Validation Successful")
    return data[matching_columns]
```

Gambar 3.52. Fungsi *validate*()

Fungsi statik lainnya pada *class* ini merupakan *filename_generator*(). Seperti pada Gambar 3.53 di bawah, fungsi ini digunakan untuk menghasilkan nama file gambar berdasarkan identifier yang diberikan. Metode ini menerima dua parameter: '*identifier*' dan '*index*'. Pertama, '*identifier*' diubah menjadi string untuk memastikan konsistensi. Jika '*index*' disediakan, nama file akan dihasilkan dalam format '{*identifier*}_{*index*}.jpg', di mana '*identifier*' dan '*index*' digabungkan dengan garis bawah. Jika index tidak disediakan, nama file hanya berupa '{*identifier*}.jpg'. Metode ini mempermudah pembuatan nama file yang konsisten dan terstruktur untuk penyimpanan atau pengelolaan file gambar.

```python
@staticmethod
def filename_generator(identifier, index=None):
    identifier = str(identifier)
    if index:
        return f'{identifier}_{index}.jpg'
    return f'{identifier}.jpg'
```

Gambar 3.53. Fungsi *filename_generator*()

Selanjut pada Gambar 3.54 di bawah merupakan fungsi *_download_image_to_cache*(). Fungsi ini mengunduh gambar dari URL, menyimpannya di direktori sementara, lalu memproses dan menimpa gambar dengan versi yang sudah diubah ukurannya. Proses dimulai dengan permintaan *GET* ke URL dengan batas waktu 10 detik. Jika berhasil, gambar disimpan ke lokasi yang ditentukan. File gambar kemudian dibuka, diproses menggunakan fungsi *process_image* pada *class ImageFormatter* untuk penyesuaian ukuran atau format, dan disimpan kembali. Jika terjadi pengecualian, pesan kesalahan dicatat. Metode ini memastikan gambar terunduh, terproses, dan tersimpan dengan benar, sambil menangani kesalahan yang mungkin terjadi selama proses.

```
def _download_image_to_cache(self, url, filename):
    try:
        response = requests.get(url, timeout=10)
        if response.status_code == 200:
            file_path = os.path.join(self.temp_dir, filename)
            with open(file_path, 'wb') as f:
                f.write(response.content)
            self.logger.info(f"Downloaded {url} to {file_path}")

            with open(file_path, 'rb') as img_file:
                processed_image = self.image_formatter.process_image(img_file)

            with open(file_path, 'wb') as f:
                f.write(processed_image.read())
        else:
            self.logger.error(f"Failed to download {url}: Status code {response.status_code}")
    except Exception as e:
        self.logger.error(f"Failed to download {url}: {e}")
```

Gambar 3.54. Fungsi *_download_image_to_cache*()

Gambar 3.55 di bawah merupakan fungsi *_upload_image_from_cache*(), fungsi ini bertujuan untuk mengunggah file gambar dari *cache* lokal ke Google Drive. Metode ini dimulai dengan membuka file gambar yang disimpan pada '*file_path*' dalam mode baca biner. File tersebut kemudian diunggah ke Google Drive menggunakan fungsi *upload_to_drive*(). Jika unggahan berhasil, metode ini mencatat pesan keberhasilan di log; jika gagal, pesan kesalahan dicatat dengan rincian tentang masalah yang terjadi, memastikan proses unggahan terdokumentasi dengan baik, dan kesalahan dapat diidentifikasi dengan jelas.

```
def _upload_image_from_cache(self, file_path, filename):
    try:
        with open(file_path, 'rb') as media:
            upload_to_drive(self.logger, self.service,
                            self.folder_result_id,
                            filename, media,
                            MimeType.IMAGE.value)
        self.logger.info(f"Uploaded {filename} from {file_path} to Google Drive")
    except Exception as e:
        self.logger.error(f"Failed to upload {filename} from {file_path}: {e}")
```

Gambar 3.55. Fungsi *_upload_image_from_cache*()

Selanjutnya merupakan fungsi untuk mengunggah file log ke Google Drive dan memastikan manajemen log yang rapi dengan nama *_upload_log_file*() seperti pada Gambar 3.56 di bawah. file log tersebut diunggah ke Google Drive dengan memanfaatkan fungsi *upload_to_drive*(). Jika terjadi kesalahan selama proses unggahan, metode ini menangkap dan mencatat pesan kesalahan yang relevan. Terakhir, metode ini memastikan semua *log* telah dibuang dan semua *handler log* ditutup dengan memanggil fungsi *flush*(). Terakhir, jika file *log* masih ada di sistem lokal, metode ini menghapusnya dan mencatat informasi bahwa file *log* telah dihapus, memastikan file log tidak menumpuk di penyimpanan lokal.

```
def _upload_log_file(self):
    log_file = self.logger.get_log_file()
    try:
        # Upload the log file to Google Drive
        upload_to_drive(self.logger, self.service,
                        self.folder_id,
                        log_file, log_file,
                        MimeType.TXT.value)
    except Exception as e:
        self.logger.error(f"Failed to upload log file: {e}")
    finally:
        # Ensure all logs are flushed and handlers are closed
        self.logger.flush()
        # Remove the log file
        if os.path.exists(log_file):
            os.remove(log_file)
            self.logger.info(f"Removed local log file: {log_file}")
```

Gambar 3.56. Fungsi *_upload_log_file*()

Berikutnya pada Gambar 3.57 di bawah merupakan fungsi *_cleanup_temp_file*() yang bertujuan untuk membersihkan file sementara yang berada di direktori sementara. Pertama, fungsi ini melakukan iterasi pada setiap nama file yang terdapat di dalam direktori sementara tersebut. Jika *file* tersebut adalah *file* biasa, maka file tersebut dihapus. Setelah semua file dihapus, metode ini mencoba menghapus direktori sementara itu sendiri. Jika berhasil, metode ini mencatat *log* bahwa direktori sementara telah dihapus. Namun, jika terjadi kesalahan selama proses pembersihan, seperti kegagalan dalam menghapus *file* atau direktori, metode ini menangkap pengecualian dan mencatat pesan kesalahan yang relevan, memastikan bahwa setiap kegagalan tercatat dengan baik untuk penelusuran masalah lebih lanjut.

```
def _cleanup_temp_files(self):
    try:
        for filename in os.listdir(self.temp_dir):
            file_path = os.path.join(self.temp_dir, filename)
            if os.path.isfile(file_path):
                os.remove(file_path)
        os.rmdir(self.temp_dir)
        self.logger.info(f"Removed temporary directory: {self.temp_dir}")
    except Exception as e:
        self.logger.error(f"Failed to clean up temporary files: {e}")
```

Gambar 3.57. Fungsi *_cleanup_temp_files*()

Fungsi pada Gambar 3.58 di bawah merupakan *get_folder_link*() yang mengembalikan URL lengkap untuk mengakses folder di Google Drive. Fungsi ini menyusun link dengan menggabungkan bagian dasar URL Google Drive dengan ID folder. Hasilnya adalah tautan yang dapat digunakan untuk mengakses folder spesifik di Google Drive, dan metode ini mengembalikan tautan tersebut sebagai *string*.

```
def get_folder_link(self):
    link = 'https://drive.google.com/drive/u/0/folders/' + self.folder_result_id
    return link
```

Gambar 3.58. Fungsi *get_folder_link*()

Fungsi terakhir pada *class* ini merupakan fungsi *run*() untuk menjalankan proses pengunduhan gambar secara paralel dan mengelola file gambar. Mengimplementasi ThreadPoolExecutor dengan maksimal tiga pekerja, fungsi ini memproses setiap baris dalam DataFrame, mengunduh gambar dari URL yang ditemukan, dan menyimpan gambar ke dalam cache sementara. Setelah semua gambar diunduh, jika opsi '*save_to_drive*' diaktifkan, gambar akan diunggah ke Google Drive, *log* file akan diunggah, dan direktori sementara akan dihapus. Jika tidak, gambar-gambar yang diunduh akan dikompresi menjadi file ZIP dan dikembalikan sebagai *buffer* ZIP. Metode ini juga menangani pengecualian selama eksekusi tugas paralel dan memastikan semua file sementara dihapus setelah proses selesai.

```
def run(self):
    futures = []
    with ThreadPoolExecutor(max_workers=3) as executor:
        for index, row in self.data.iterrows():
            identifier = row.iloc[0]
            image_cols = [col for col in self.data.columns if
                          col.startswith('Image') and len(col) < 10]

            urls = [row[col] for col in image_cols
                    if isinstance(row[col], str)]

            image_index = index + 1 if len(urls) > 1 else None
            for url in urls:
                filename = ImageDownloader.filename_generator(identifier, image_index)
                futures.append(executor.submit(self._download_image_to_cache, url, filename))
                time.sleep(0.1)  # Introduce a short delay between task submissions

    for future in as_completed(futures):
        try:
            future.result()  # This will propagate exceptions if any occurred during execution
        except Exception as e:
            self.logger.error(f"Exception occurred during future execution: {e}")

    if self.save_to_drive:
        for filename in os.listdir(self.temp_dir):
            file_path = os.path.join(self.temp_dir, filename)
            self._upload_image_from_cache(file_path, filename)
        self._upload_log_file()
        self._cleanup_temp_files()
        return get_folder_url(self.folder_result_id)

    file_list = [(filename, os.path.join(self.temp_dir, filename)) for filename in os.listdir(self.temp_dir)]
    zip_buffer = zip_files(file_list)
    return zip_buffer
```

Gambar 3.59. Fungsi *run*()

Berikutnya integrasi *class ImageDownloader* dalam antarmuka *website* dilakukan. Tahapan pertama merupakan *import library* yang dibutuhkan

serta inisiasi *file* yang akan digunakan sebagai data sampel seperti pada Gambar 3.60 di bawah.

```
from utils.drive import authenticate
from utils.activity.image_downloader import ImageDownloader
from config import Activity, get_config
import streamlit as st
import pandas as pd

ACTIVITY = Activity.IMAGE_DOWNLOAD
sample_import_data = pd.read_excel(get_config(ACTIVITY).get('import_sample_url'))
```

Gambar 3.60. *Import Library* serta Inisiasi Data Sampel

Data sampel yang telah diinisiasi kemudian dimasukan kedalam elemen *expander,* hal ini bertujuan agar pengguna dapat menutup ataupun membuka contoh sampel data sesuai dengan kebutuhan seperti pada Gambar 3.61 di bawah.

```
st.markdown("""
    Upload Excel untuk mengunduh gambar dengan link. \\
    Excel wajib memiliki kolom `SKU` `SAP` `Image`
""")

with st.expander("Contoh Data"):
    st.dataframe(sample_import_data)

if "uploader_key" not in st.session_state:
    st.session_state.uploader_key = 0
```

Gambar 3.61. Pemuatan Data Sampel

Pembuatan fungsi *run_image_downloader*() seperti pada Gambar 3.62 di bawah juga dilakukan dalam mengautentikasi akun layanan Google menggunakan kredensial yang disimpan pada *file secrets* Streamlit. kemudian menginisiasi *class ImageDownloader* dengan data dan parameter '*save_to_drive*'. Fungsi ini memanggil fungsi *run*() *class ImageDownloader* untuk memulai proses pengunduhan gambar dan, jika diperlukan, unggah gambar ke Google Drive. Hasil dari fungsi *run*() kemudian dikembalikan sebagai keluaran fungsi.

```python
def run_image_downloader(data, save_to_drive=False):
    service = authenticate(st.secrets['google_service_account'])
    downloader = ImageDownloader(service, data, save_to_drive)
    result = downloader.run()
    return result
```

Gambar 3.62. Fungsi *run_image_downloader*()

Selanjutnya, pada Gambar 3.63 di bawah pengembangan antarmuka untuk mengembangkan antarmuka untuk mengunggah file CSV atau XLSX menggunakan Streamlit. Setelah file diunggah, fungsi *run_image_downloader*() digunakan untuk memproses data yang diunggah. Selama pemrosesan, pengguna akan melihat *spinner* yang menunjukkan bahwa proses sedang berlangsung. Setelah proses selesai, pengguna diberi tahu bahwa mereka dapat mengunduh file ZIP yang dihasilkan. Tombol unduh ZIP dilengkapi dengan label dan tipe *file* yang sesuai. Setelah tombol di-klik, fungsi *update_key*() dipanggil untuk meningkatkan nilai '*uploader_key*' dalam *session_state*, yang mungkin digunakan untuk mengubah status atau tampilan elemen uploader di antarmuka.

```python
uploaded_file = st.file_uploader("Upload CSV/XLSX file", type=['xlsx'], key=f"uploader_{st.session_state.uploader_key}")

def update_key():
    st.session_state.uploader_key += 1

if uploaded_file is not None:
    data = pd.read_excel(uploaded_file)

    with st.spinner('Processing...'):
        result = run_image_downloader(data)
        st.success("Klik untuk download 👇")

        st.download_button(
            label="Download ZIP",
            data=result,
            file_name="images.zip",
            mime="application/zip",
            on_click=update_key,
        )
```

Gambar 3.63. Pembuatan *Upload Placeholder* dan *Button* Unduh

3.2.8 Google Apps Script untuk Otomatisasi Google Form

Google Apps Script adalah platform scripting berbasis JavaScript yang dirancang untuk mengotomatisasi berbagai layanan Google, termasuk Google Form [10]. Dengan Google Apps Script, pengguna dapat menciptakan otomatisasi untuk memproses respons secara real-time, mengirim notifikasi email, atau

memindahkan data dari Google Form ke Google Spreadsheet tanpa perlu intervensi manual. Fitur ini sangat berguna dalam meningkatkan efisiensi.

3.3 Kendala yang Ditemukan

Adapun kendala yang ditemukan selama periode kerja magang sebagai MDM *Data Analyst Intern* di Kawan Lama Group antara lain:

a. Kesenjangan Keterampilan (*Skill Gaps*)

Pendekatan atau solusi dalam menyelesaikan penugasan atau masalah sering kali bersifat task-oriented, di mana setiap masalah memerlukan keterampilan khusus untuk mencapai solusi yang optimal. Kesenjangan antara keterampilan yang dimiliki dan ekspektasi ideal dari penugasan yang diberikan dapat menyebabkan inefisiensi waktu, memperlambat kemajuan, serta menurunkan kualitas kerja. Dampak dari kesenjangan ini tidak hanya mempengaruhi produktivitas dan kualitas hasil kerja, tetapi juga dapat berdampak pada motivasi tim, mengurangi semangat, dan menghambat pencapaian tujuan yang diinginkan.

b. Kurangnya Komunikasi dan Koordinasi antar Anggota Tim

Efektivitas komunikasi dan koordinasi yang kuat merupakan landasan fundamental bagi kesuksesan tim. Ketika kedua aspek ini mengalami kendala, konsekuensi negatif dapat muncul, seperti penurunan produktivitas, efisiensi, dan semangat tim. Kesalahpahaman mengenai tujuan dan peran masing-masing anggota dapat menyebabkan kebingungan dan arah kerja yang tidak terfokus. Komunikasi yang tidak efektif berpotensi memicu kesalahan, keterlambatan, dan kerugian finansial. Anggota tim yang merasa terputus dan kurang dihargai mungkin mengalami penurunan motivasi, menghasilkan pekerjaan dengan kualitas lebih rendah. Selain itu, kurangnya komunikasi dapat membuka celah bagi kesalahpahaman dan konflik, yang pada akhirnya menimbulkan ketegangan dalam tim.

c. Tidak adanya Pembagian Penugasan yang Jelas

Efektivitas tim yang baik sangat bergantung pada pembagian tugas yang jelas. Tanpa adanya kejelasan, tim akan menghadapi berbagai tantangan terkait alokasi waktu, tenaga, dan fokus para anggotanya. Keterlibatan seorang individu dalam banyak proyek secara bersamaan dapat mengurangi produktivitas, efisiensi, dan fokus mereka. Hal ini dikarenakan individu tersebut kesulitan untuk memprioritaskan dan membagi waktunya secara efektif. Akibatnya, pengerjaan tugas menjadi tersendat-sendat, rentan kesalahan, dan berpotensi menimbulkan kelelahan (burnout) pada individu. Selain itu, kurangnya kejelasan pembagian tugas juga berdampak pada fokus dan konsentrasi. Ketika individu mengerjakan banyak tugas dalam waktu bersamaan, mereka akan kesulitan untuk memusatkan perhatiannya pada satu hal. Akibatnya, kualitas hasil kerja menurun dan risiko terjadinya kesalahan meningkat.

3.4 Solusi atas Kendala yang Ditemukan

Memahami kendala yang dihadapi, solusi yang dilakukan untuk mengatasi masalah tersebut antara lain:

a. Peningkatan Keterampilan

Peningkatan keterampilan merupakan solusi utama untuk mengatasi kesenjangan keterampilan yang dapat mempengaruhi efisiensi dan kualitas kerja. Dengan mengidentifikasi keterampilan yang kurang atau tidak sesuai dengan ekspektasi penugasan, organisasi dapat merancang program pelatihan yang spesifik untuk mengatasi kebutuhan tersebut. Pelatihan ini harus berfokus pada pengembangan keterampilan yang relevan, disertai dengan penyediaan sumber daya yang memadai serta dukungan dari mentor atau rekan kerja yang lebih berpengalaman. Selain itu, pengembangan keterampilan secara berkelanjutan melalui kursus, workshop, dan pembelajaran mandiri akan memperkuat kemampuan tim, mengurangi inefisiensi waktu, serta meningkatkan kualitas hasil kerja. Dengan cara ini, organisasi dapat memastikan bahwa timnya tetap

termotivasi dan produktif, serta mampu mencapai tujuan yang diinginkan dengan lebih efektif.

b. Meningkatkan Komunikasi antar anggota tim

Peningkatan komunikasi antar anggota tim dapat dicapai melalui berbagai langkah. Salah satunya adalah dengan membangun saluran komunikasi yang terbuka dan rutin, seperti rapat mingguan, di mana semua anggota tim dapat terhubung dan berbagi informasi. Selain itu, penting untuk mempromosikan budaya mendengarkan yang inklusif, di mana setiap anggota tim didorong untuk berpartisipasi aktif dalam diskusi dan memberikan masukan. Mediasi yang dilakukan oleh mentor dan supervisor juga berperan penting dalam meningkatkan kualitas komunikasi antar anggota tim.

c. Mengajukan kejelasan dalam pembagian penugasan yang merata

Pembagian penugasan yang jelas dan merata merupakan kunci untuk mengoptimalkan produktivitas tim. Hal ini dapat dicapai dengan mengidentifikasi keahlian dan minat masing-masing anggota tim untuk menyesuaikan tugas dengan kemampuan mereka. Memastikan bahwa setiap anggota tim memahami tujuan umum proyek dan peran mereka dalam mencapainya akan membantu menghindari kebingungan dan overlapping tugas. Selain itu, manajemen dan pemantauan progres setiap anggota tim dapat dilakukan menggunakan Google Spreadsheet untuk memantau kemajuan dan memastikan semua tugas terlaksana dengan baik.

UNIVERSITAS MULTIMEDIA NUSANTARA