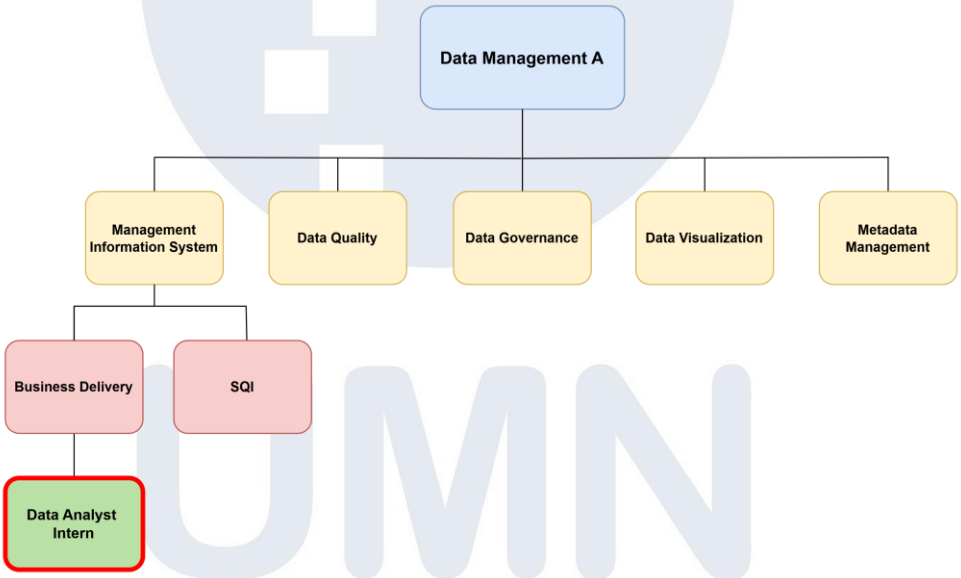
BAB III

PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

Pada program magang ini, posisi yang ditempati adalah sebagai *Data Analyst Intern* di dalam divisi *Management Information System* (MIS) yang berada di bawah biro *Data Management* A. Dalam biro *Data Management* A terdapat beberapa tim dan sub-tim yang dibawahinya.

3.1.1 Kedudukan



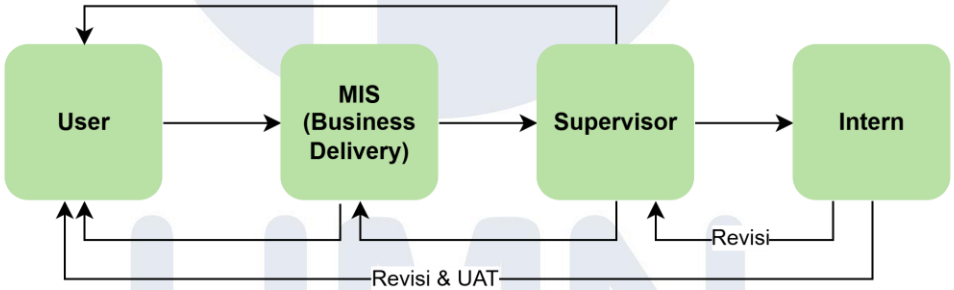
Gambar 2.3 Diagram Struktur Organisasi Biro *Data Management* A.

Posisi yang ditempati selama program magang adalah sebagai *Data Analyst Intern* di tim MIS yang berada di bawah Biro *Data Management* A (DTM-A). DTM-A terdiri dari beberapa tim, yaitu *Management Information System* (MIS), *Data Quality* (DQ), *Data Governance* (DG), *Data Visualization* (DV), dan *Metadata Management* (MM). Divisi *Management Information System* (MIS). MIS memiliki 2 subtim yaitu *Business Delivery* dan *SQI*, dan peserta magang ditempatkan di tim *Business Delivery*, yaitu bagian yang bertanggung jawab dalam penyediaan kebutuhan laporan,

automasi proses data, perancangan *website* untuk penyajian laporan, serta integrasi API dan pengembangan aplikasi internal.

3.1.2 Koordinasi

Selama periode magang, tugas dan tanggung jawab umumnya diberikan dan diarahkan oleh Ibu Evelyn Immanuel selaku *Senior IT Specialist Data Management A*, serta berada dalam koordinasi dengan Ibu Farissa Adelia selaku *Senior IT Analyst of Data Management* A. Proses komunikasi dan koordinasi pekerjaan dilakukan melalui *Microsoft Teams*, termasuk penyampaian permintaan, diskusi teknis, maupun pelaporan progres. Selain itu, terdapat meeting rutin mingguan bersama tim *Business Delivery* untuk membahas perkembangan tugas, serta meeting dua mingguan bersama seluruh tim MIS dan Kepala Biro guna menyampaikan progres pekerjaan serta mendapatkan arahan lebih lanjut.



Gambar 2.4 Alur Koordinasi Pekerjaan.

Alur koordinasi pekerjaan pada program magang di Divisi MIS dilakukan secara terstruktur untuk memastikan setiap permintaan dari unit kerja dapat ditangani dengan tepat. Proses dimulai ketika suatu unit atau divisi (*user*) mengajukan permintaan pembuatan laporan atau kebutuhan pengolahan data kepada Divisi MIS. Setelah permintaan diterima, pembimbing lapangan selaku *Senior IT Specialist* melakukan analisis awal untuk menentukan penanggung jawab tugas, baik dikerjakan langsung oleh pembimbing maupun dialokasikan kepada peserta magang.

Setelah penanggung jawab ditetapkan, sebuah grup komunikasi khusus dibuat melalui *platform Microsoft Teams* yang beranggotakan tim

MIS terkait (termasuk peserta magang) sebagai PIC projek, pembimbing lapangan, dan *user*. Grup ini digunakan sebagai media koordinasi utama untuk seluruh proses pekerjaan, seperti diskusi kebutuhan pengerjaan projek, penyampaian data pendukung, pembaruan progress, revisi, dan UAT (*User Acceptance Testing)*. Hal ini memungkinkan untuk peserta magang dapat berkoordinasi langsung dengan *user* maupun pembimbing terkait pengerjaan proyek tersebut. Alur koordinasi ini memungkinkan setiap pihak terlibat secara aktif dan memastikan pekerjaan berjalan sesuai kebutuhan *user* serta standar kualitas Divisi MIS.

3.2 Tugas yang Dilakukan

Selama periode program kerja magang, beberapa proyek telah dikerjakan berdasarkan kebutuhan dan permintaan dari *user* dan pembimbing lapangan. Uraian detail pekerjaan yang dilakukan dapat dilihat pada tabel 3.2.1.

Tabel 3.2.1 Detail Pekerjaan yang Dilakukan

No.	Minggu	Proyek	Keterangan
1	Minggu ke 1 - 3	Pemahaman Dasar Hal Teknis dan Perencanaan Proyek	Mengerjakan e-*learning* BCA, latihan SQL Oracle, SSIS/SSRS, dan diskusi rencana proyek.
2	Minggu ke 4	Merancang Control Flow Dan SQL Query Laporan Agunan Menggunakan SSIS	Menyusun alur ETL dan *query* untuk pembentukan tabel *staging* dan tabel final laporan agunan.
3	Minggu ke 4	Membuat Tampilan Laporan Agunan di SSRS	Merancang tampilan laporan agunan dan menyesuaikan struktur tabel agunan.
4	Minggu ke 5	Merancang Control Flow dan Query Laporan Pinjaman Menggunakan SSIS	Membuat alur ETL dan SQL *query* untuk kebutuhan pembuatan laporan pinjaman.
5	Minggu ke 5	Membuat Tampilan Laporan Pinjaman Menggunakan SSRS	Merancang tampilan laporan pinjaman dan memastikan data tampil sesuai format.
6	Minggu ke 5 - 6	Merancang Control Flow dan SQL Query Laporan KUR Menggunakan SSIS	Merancang alur ETL, tabel summary dan detail, serta proses validasi data KUR.
7	Minggu ke 5 - 6	Membuat Tampilan Laporan KUR Menggunakan SSRS	Membangun tampilan laporan KUR, termasuk tabel ALL, JENIS, dan SEKON.

8	Minggu ke 6	Pemahaman Alur Kerja Aplikasi MIRA dan Perencanaan Integrasi API	Menganalisis dan memahami struktur aplikasi MIRA dan kebutuhan untuk integrasi API.
9	Minggu ke 6 – 7, 11,15	Perancangan Flow Integrasi API pada Halaman User FAQ	Membangun alur pemanggilan API FAQ dan pengelompokan data FAQ ke tiga kategori.
10	Minggu ke 7 - 8	Perancangan Flow Integrasi API Pada Halaman Faqdetail	Mengembangkan alur untuk menampilkan detail FAQ berdasarkan request ID.
11	Minggu ke 8, 11, 12, 15	Perancangan Flow Integrasi API Pada Halaman *Report*s	Menyusun alur integrasi API untuk halaman *Report*s dan mengelola data laporan.
12	Minggu ke 9	Perancangan Flow Integrasi API Pada Form Update Detail BPRO	Perancangan alur integrasi API untuk proses persetujuan BPRO dan pembaruan status berdasarkan *input form*.
13	Minggu ke 9	Perancangan Flow Integrasi API Pada Form Finish BPRO	Perancangan alur integrasi API untuk penyelesaian BPRO termasuk pembaruan status dan pengiriman notifikasi.
14	Minggu ke 10	Menambahkan Fitur Notes dan Last modified	Menambahkan fitur pencatatan dan l*ast modified* pada aplikasi MIRA.
15	Minggu ke 12	Perancangan Endpoint GET Data Insidentil	Membuat *endpoint* API untuk menampilkan daftar insidentil sesuai kebutuhan aplikasi.
16	Minggu ke 12	Pembuatan Method Pengambilan Data Insidentil Dan Pengujian Endpoint API	Pembuatan *method* pada sisi model yang digunakan untuk mengambil data insidentil dan pengujian endpoint API.
17	Minggu ke 13	Merancang Control flow, SQL Query, dan Tampilan Laporan Likuiditas Intrahari	Merancang alur ETL, SQL *query*, dan Tampilan untuk membuat laporan likuiditas intrahari menggunakan SSIS dan SSRS.
18	Minggu ke 14, 16	Pembuatan Website Proyek Laporan Beehive	Membuat *website* beehive menggunakan ASP.NET.
19	Minggu ke 15	Menambahkan Header Pada Excel Hasil Download	Membuat alur untuk memasukkan *header* yang berisi data tertentu pada data excel.
20	Minggu ke 16	Membuat Laporan Beehive Menggunakan SSRS	Menampilkan data beehive dalam bentuk laporan menggunakan SSRS.

3.3 Uraian Pelaksanaan Kerja

Selama proses kerja magang, terdapat beberapa proyek yang dikerjakan, yaitu perancangan laporan agunan, pinjaman, KUR, likuditas harian, beehive, dan MIRA API. Setiap proyek mencakup proses perancangan hingga implementasi yang dilakukan secara rinci.

3.3.1 Proses Pelaksanaan

3.3.1.1 Pemahaman Dasar Hal Teknis dan Perencanaan Proyek

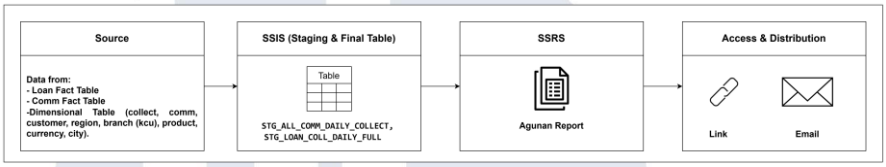
Tahap ini berfokus pada orientasi dan penguatan dasar sebelum memasuki pengerjaan proyek. Kegiatan diawali dengan mempelajari materi umum BCA melalui *website e-learning*, mencakup keamanan data, prosedur kerja, *employer branding*, etika di tempat kerja, serta kode etik yang berlaku. Setelah itu, peserta magang mengerjakan soal latihan terkait SQL dan relasi tabel untuk membiasakan diri menggunakan *Oracle* sebagai DBMS serta *SQL Developer* sebagai *tools* utama dalam pengolahan data. Selain itu, terdapat pembelajaran terkait struktur data, hubungan antar tabel, serta pengenalan penggunaan SSIS dan SSRS sebagai bagian dari proses integrasi dan pelaporan data. Selanjutnya, dilakukan diskusi dengan pembimbing dan tim terkait rencana proyek yang akan dikerjakan, termasuk pemahaman kebutuhan pengambilan data, alur kerja, serta output dan hasil yang diharapkan dari proyek.

3.3.1.2 Merancang Control Flow Dan SQL Query Laporan Agunan Menggunakan SSIS

Laporan Agunan merupakan laporan yang menampilkan data terkait detail agunan (jenis, nilai, dokumen, lokasi) yang terhubung dengan fasilitas kredit serta status kolektibilitas pada tanggal pelaporan untuk kebutuhan monitoring risiko dan validasi jaminan. Pembuatan laporan ini melalui perancangan *flow* ETL menggunakan SSIS dan penyajian *report* menggunakan SSRS, sehingga data agunan menjadi terstandar, konsisten, dan siap diakses unit kerja lain. Tabel 3.3.1 menunjukkan alur pengolahan data Laporan Agunan mulai dari sumber data, tahapan pemrosesan, hingga *output* yang dihasilkan.

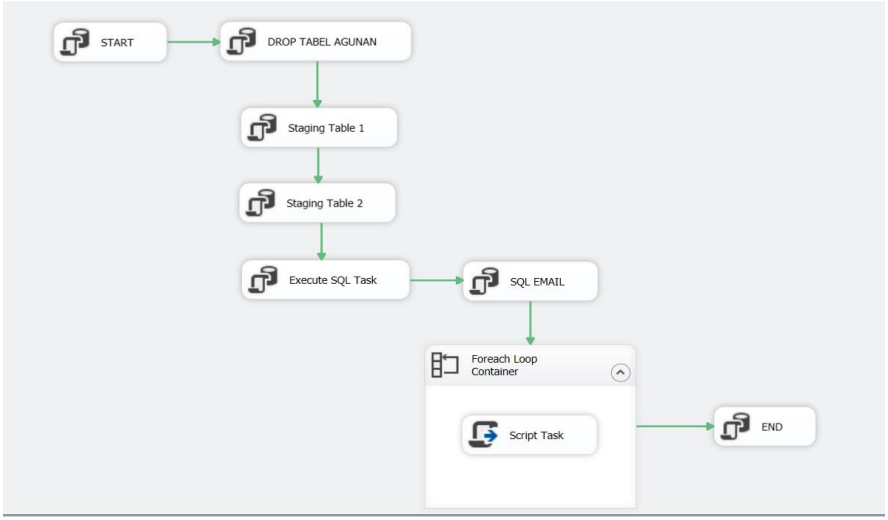
Tabel 3.3.1 Alur Data Laporan Agunan.

Input (Source)	Process	Output
Data komitmen harian & kolektibilitas	Pembuatan dan penggabungan staging table 1 dan 2	Tabel final agunan
Data Agunan (tabel final)	Bangun *report* tabular di SSRS	Laporan Agunan



Gambar 3.1 Alur Arsitektur Proyek Laporan Agunan.

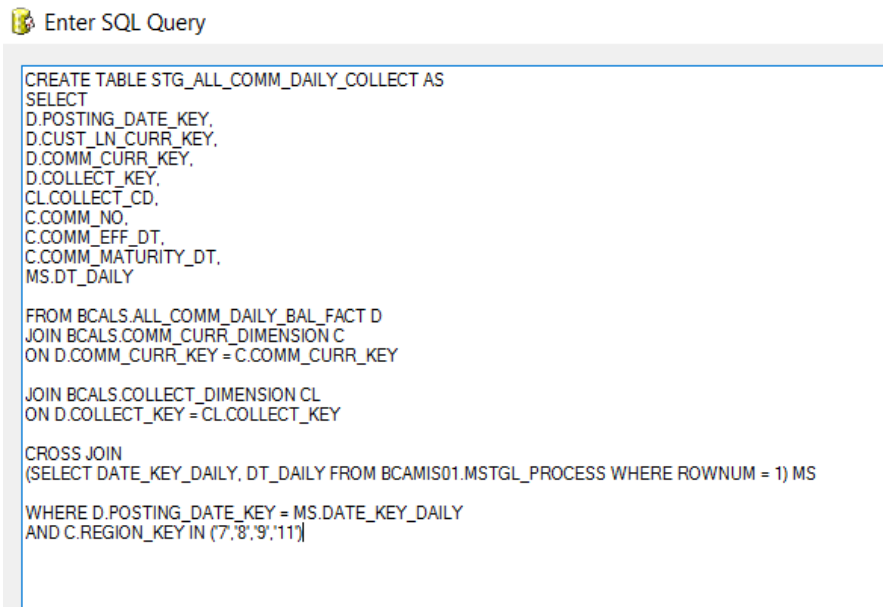
Gambar 3.1 menunjukkan alur arsitektur proyek Laporan Agunan dimulai dari sumber data berupa *fact table* dan *dimensional table*. Data tersebut diproses melalui SSIS untuk membentuk *staging table* lalu digabung menjadi tabel final agunan yang sudah siap pakai. Selanjutnya SSRS membaca tabel final tersebut untuk menghasilkan Agunan *Report*, kemudian laporan didistribusikan dan diakses oleh pengguna melalui *link* portal SSRS atau pengiriman email.



Gambar 3.2 *Control Flow* Paket SSIS Laporan Agunan.

Gambar 3.2 menunjukkan *control flow* SSIS yang menangani proses ETL dan *workflow* manajemen untuk pembuatan laporan

agunan, *flow* dimulai dari penghapusan tabel agunan sebelumnya melalui *task drop table* agunan untuk memastikan tidak ada data lama yang tertinggal. Proses selanjutnya adalah pembuatan *Staging Table* 1 dan *Staging Table* 2 yang berfungsi sebagai tempat sementara untuk menampung data hasil ekstraksi sebelum dibentuk menjadi struktur final. Setelah kedua *staging table* selesai dibuat, lanjut ke proses mengeksekusi *Execute SQL Task* untuk membentuk tabel akhir yaitu tabel laporan agunan. Selanjutnya, menjalankan *sql task* untuk SQL EMAIL sebagai pemicu proses pengiriman email. Tahapan ini diteruskan ke *Foreach Loop Container* yang berisi *Script Task*, di mana berisi logika detail seperti pembuatan *body email*, format pesan, hingga struktur *email* yang dikirimkan berdasarkan beberapa parameter yang telah ditentukan. Seluruh rangkaian proses ditutup dengan *task* END yang akan memperbarui data di tabel mis_*report*s yang menandakan bahwa paket SSIS selesai dieksekusi.



Gambar 3.3 *SQL Query* Pembuatan *Staging Table* 1.

Gambar 3.3 menunjukkan *SQL query* pada tahap pembuatan s*taging table* 1, dimana *query* tersebut digunakan untuk melakukan proses ekstraksi dan pembentukan tabel stg_all_comm_daily_collect,

yang berisi data komitmen harian yang diperoleh dari beberapa *fact table* dan *dimension table* melalui operasi *join* serta *cross join* dengan tabel proses tanggal harian. Tahap ini menghasilkan dataset awal mengenai transaksi kolektibilitas yang telah di filter berdasarkan tanggal dan *region* tertentu.

Enter SQL Query

```
CREATE TABLE STG_LOAN_COLL_DAILY_FULL AS
SELECT
E.REGION_SHRT_NM,
K.KCU_CD || '-' || K.KCU_NAME AS KCU,
F.BRANCH_NAME,
D.CUST_NO,
D.ACCT_LN_NO,
D.ACCT_NAME,
C.COMM_NO,
A.LN_NO,
G.PRODUCT_SUB_TYPE_NM,
H.CUST_LN_SEG_CD,
C.COMM_EFF_DT,
C.COMM_MATURITY_DT,
B.COLL_NO,
A.COLL_VALUE_OCUR,
CC.CURR_CD,
B.COLL_DESC_1,
B.COLL_DESC_2,
B.COLL_DESC_3,
T.COLL_TYPE_NM,
B.COLL_LAST_PRC_DT,
B.COLL_EXP_DT,
B.COLL_BUKTI_PEMILIKAN,
B.COLL_LOKASI_SIMPAN_DOCS,
B.COLL_SANDI_PENGIKATAN,
B.COLL_NM_KEPEMILIKAN,
B.COLL_PGTB_NO,
B.COLL_PGTB_DT,
A.POSTING_DATE_KEY,
MS.DT_DAILY,
C.COMM_CURR_KEY

FROM BCALS.LOAN_COLL_DAILY_FACT A

JOIN BCALS.COLL_CURR_DIMENSION B
ON A.COLL_CURR_KEY = B.COLL_CURR_KEY

JOIN BCALS.COLL_TYPE_DIMENSION T
ON A.COLL_TYPE_KEY = T.COLL_TYPE_KEY

JOIN BCALS.COMM_CURR_DIMENSION C
ON A.COMM_CURR_KEY = C.COMM_CURR_KEY

JOIN BCALS.ACCT_LOAN_CURR_DIMENSION D
ON D.ACCT_LN_CURR_KEY = C.ACCT_LN_CURR_KEY

JOIN BCADW.REGION_DIMENSION E
ON A.REGION_KEY = E.REGION_KEY

JOIN BCADW.BRANCH_DIMENSION F
ON A.BRANCH_KEY = F.BRANCH_KEY

JOIN BCADW.KCU_DIMENSION K
ON A.KCU_KEY = K.KCU_KEY

JOIN BCALS.LN_PRODUCT_SUB_TYPE_DIMENSION G
ON G.PRODUCT_SUB_TYPE_KEY = A.PRODUCT_SUB_TYPE_KEY

JOIN BCALS.CUST_LOAN_SEGMENT_DIMENSION H
ON H.CUST_LN_SEG_KEY = A.CUST_LN_SEG_KEY

JOIN BCADW.CURRENCY_DIMENSION CC
ON A.CURR_KEY = CC.CURR_KEY


CROSS JOIN (SELECT DATE_KEY_DAILY, DT_DAILY FROM BCAMIS01.MSTGL_PROCESS WHERE ROWNUM = 1) MS

WHERE A.POSTING_DATE_KEY = MS.DATE_KEY_DAILY
AND C.REGION_KEY IN ('7','8','9','11')
AND A.CUST_LN_SEG_KEY IN ('2','3','6')
AND G.PRODUCT_TYPE_NM NOT IN ('CREDIT CARD', 'KPPR','KKB')
```

Gambar 3.4 *SQL Query* Pembuatan *Staging Table* 2.

Gambar 3.4 menunjukkan *SQL query* pada tahap pembuatan *staging table 2*, *query* ini membentuk tabel stg_loan_coll_daily_full, yaitu tabel *staging* kedua yang berisi data pinjaman dengan cakupan lebih luas. *Query* ini mencakup penggabungan berbagai tabel dimensi seperti *customer, branch, region, loan segment, product type, currency,* hingga kolom spesifik seperti kolektibilitas, tanggal pengikat, serta informasi kepemilikan dokumen.

Enter SQL Query

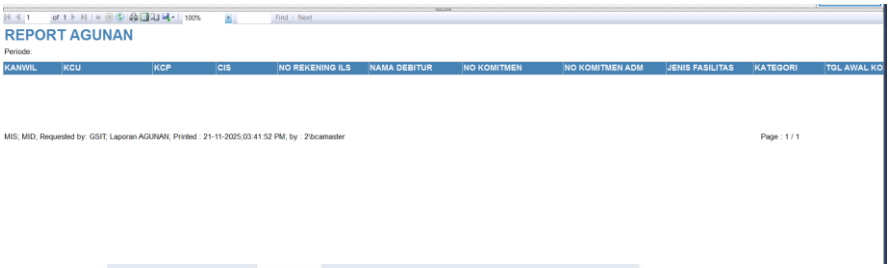
```
CREATE TABLE AGUNAN_TABLE AS

SELECT

STG2.REGION_SHRT_NM AS KANWIL,
STG2.KCU,
STG2.BRANCH_NAME AS KCP,
STG2.CUST_NO AS CIS,
STG2.ACCT_LN_NO AS NO_REKENING_ILS,
STG2.ACCT_NAME AS NAMA_DEBITUR,
STG2.COMM_NO AS NO_KOMITMEN,
STG2.LN_NO AS NO_KOMITMEN_ADM,
STG2.PRODUCT_SUB_TYPE_NM AS JENIS_FASILITAS,
STG2.CUST_LN_SEG_CD AS KATEGORI,
STG2.COMM_EFF_DT AS TGL_AWAL_KOMITMEN,
STG2.COMM_MATURITY_DT AS TGL_JATUH_TEMPO_KOMITMEN,
STG1.COLLECT_CD AS KOLEKTIBILITAS,
STG2.COLL_NO AS NO_AGUNAN,
STG2.COLL_VALUE_OCUR AS NILAI_AGUNAN,
STG2.CURR_CD AS MATA_UANG,

STG2.COLL_DESC_1 AS KOLOM_DESKRIPSI_1,
STG2.COLL_DESC_2 AS KOLOM_DESKRIPSI_2,
STG2.COLL_DESC_3 AS KOLOM_DESKRIPSI_3,
STG2.COLL_TYPE_NM AS JENIS_AGUNAN,
STG2.COLL_LAST_PRC_DT AS JATUH_TEMPO_AGUNAN,
STG2.COLL_EXP_DT AS TANGGAL_KADALUARSA,
STG2.COLL_BUKTI_PEMILIKAN AS BUKTI_PEMILIKAN,
STG2.COLL_LOKASI_SIMPAN_DOCS AS LOKASI_SIMPAN_DOKUMEN,
STG2.COLL_SANDI_PENGIKATAN AS SANDI_PENGIKATAN,
STG2.COLL_NM_KEPEMILIKAN AS NAMA_KEPEMILIKAN,
STG2.COLL_PGTB_NO AS NO_PENGIKATAN,S
STG2.COLL_PGTB_DT AS TGL_PENGIKATAN,
TO_DATE(STG2.POSTING_DATE_KEY,'J') AS POSTING_DATE_KEY,
STG2.DT_DAILY


FROM STG_LOAN_COLL_DAILY_FULL STG2

JOIN STG_ALL_COMM_DAILY_COLLECT STG1
ON STG2.POSTING_DATE_KEY = STG1.POSTING_DATE_KEY

AND STG2.COMM_CURR_KEY = STG1.COMM_CURR_KEY
```

Gambar 3.5 *SQL Query* Pembuatan *Final Table* (Tabel Agunan).

Gambar 3.5 menunjukkan *SQL query* pada tahap pembuatan *final table* yaitu tabel agunan. Pada tahap ini, data dari kedua *staging table* digabungkan menggunakan operasi *join* untuk menghasilkan informasi agunan secara lengkap, termasuk informasi wilayah (kanwil), cabang (KCP), identitas nasabah, jenis fasilitas, kategori kredit, nilai agunan, tanggal jatuh tempo, serta berbagai kolom deskripsi dokumen agunan. Proses transformasi ini sekaligus membersihkan dan menormalkan data sehingga siap ditampilkan dalam bentuk laporan. Berdasarkan berbagai proses yang dilakukan menggunakan SSIS ini, terdapat perubahan pada data yang digunakan, dimana sebelum diproses, data agunan masih tersebar di beberapa sumber (komitmen harian, kolektibilitas, data pinjaman, dan beberapa atribut agunan) dan belum terintegrasi. Setelah diproses, data

digabung melalui *staging* lalu menjadi tabel final agunan yang siap dipakai SSRS sebagai *report* dan dapat diakses melalui link.

3.3.1.3 Membuat Tampilan Laporan Agunan Menggunakan SSRS dan UAT.



Gambar 3.6 Tampilan Laporan Agunan.

Gambar 3.6 menunjukkan tampilan *report* yang dibangun menggunakan SSRS (*SQL Server Reporting Services*). SSRS digunakan untuk membangun tampilan laporan agunan dengan format tabular, dimana pada proyek ini menampilkan berbagai informasi seperti lokasi cabang, jenis agunan, nilai agunan, tanggal pengikatan, hingga detail dokumen pendukung. Laporan ini kemudian di-*deploy*, sehingga dapat diakses oleh unit kerja terkait sesuai dengan tujuan penerima *email* secara rutin sesuai dengan periode yang ditentukan.

Tabel 3.3.2 Tabel UAT Laporan Agunan.

Scenario	Input	Expected Output	Result
Normal	*Run package* dan *report*.	Tabel agunan terbentuk dan *report* agunan tampil.	Berhasil membuat tabel dan menampil *report* agunan.
Data periode kosong	*Run report.*	Tidak terdapat data pada periode ini.	Muncul pesan "Tidak terdapat data pada periode ini"

Tabel 3.3.2 menunjukkan hasil UAT untuk Laporan Agunan dengan dua skenario utama, yaitu skenario normal dan skenario periode data kosong. Pada skenario normal, paket SSIS dan *report*

dijalankan dan diharapkan tabel agunan terbentuk serta *report* agunan dapat ditampilkan, dengan hasil uji berhasil membuat tabel dan menampilkan *report*. Pada skenario periode kosong, *report* dijalankan pada rentang waktu tanpa data dan sistem diharapkan menampilkan informasi bahwa tidak terdapat data pada periode tersebut, dengan hasil uji berupa munculnya pesan "Tidak terdapat data pada periode ini".

Tabel 3.3.3 Estimasi Waktu Proses dan Frekuensi Eksekusi Laporan Agunan.

Laporan	Estimasi Waktu Run	Frekuensi
Laporan Agunan	9 Menit (Run SSIS dan render SSRS)	Mingguan

Tabel 3.3.3 menunjukkan estimasi waktu yang dibutuhkan untuk menjalankan proses laporan agunan (SSIS hingga render SSRS) serta frekuensi pelaksanaannya, sehingga memudahkan pemantauan kebutuhan operasional dan jadwal distribusi laporan.

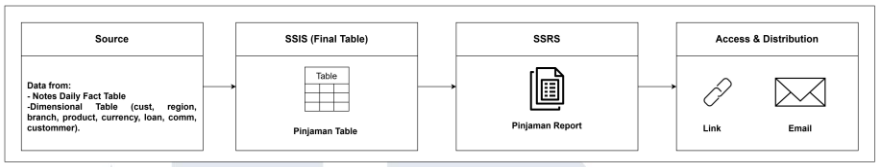
3.3.1.4 Merancang Control Flow dan Query Laporan Pinjaman Menggunakan SSIS

Laporan Pinjaman merupakan laporan yang menampilkan informasi pinjaman sebagai bahan pemantauan portofolio kredit, dimana pembuatan laporan ini berfungsi untuk membentuk laporan terstruktur untuk SSRS, serta SSIS untuk pembuatan proses ETL dan pengiriman link laporan melalui email terjadwal, sehingga memudahkan akses user untuk laporan pinjaman. Tabel 3.3.4 menunjukkan alur pengolahan data Laporan Pinjaman mulai dari sumber data, tahapan pemrosesan, hingga *output* yang dihasilkan.

Tabel 3.3.4 Alur Data Laporan Pinjaman.

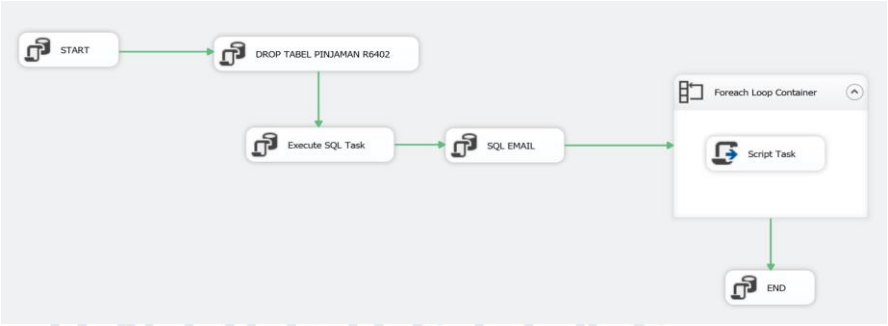
Input (Source)	Process	Output
Data *region, branch, product, currency, collector, loan*.	Pembuatan tabel pinjaman dan fungsi	Tabel Pinjaman

	pengiriman link melalui email.	
Data Pinjaman (tabel final)	Bangun *report* tabular di SSRS	Laporan Pinjaman



Gambar 3.7 Alur Arsitektur Proyek Laporan Pinjaman.

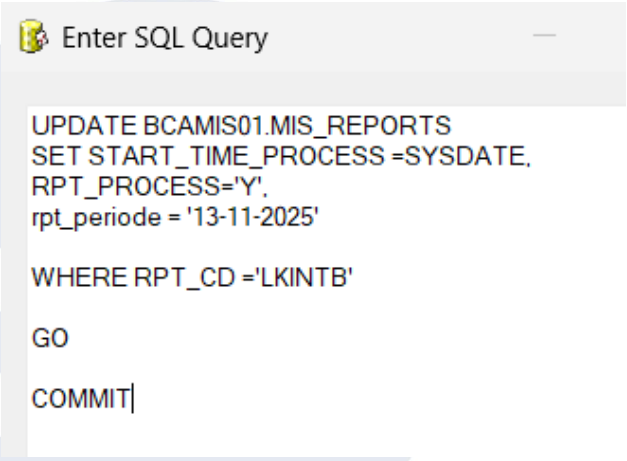
Gambar 3.7 menunjukkan arsitektur proyek Laporan Pinjaman dimulai dari sumber data berupa *fact table* pinjaman harian dan *dimensional table* (nasabah, *region*, cabang, produk, *currency*, *loan*, dan komitmen). Data kemudian diproses melalui SSIS untuk membentuk tabel pinjaman final sebagai dataset terstruktur. Selanjutnya SSRS membaca tabel tersebut untuk menghasilkan Pinjaman *Report*, dan laporan didistribusikan serta diakses pengguna melalui *link* portal SSRS atau pengiriman email terjadwal.



Gambar 3.8 *Control Flow* Paket SSIS Laporan Pinjaman.

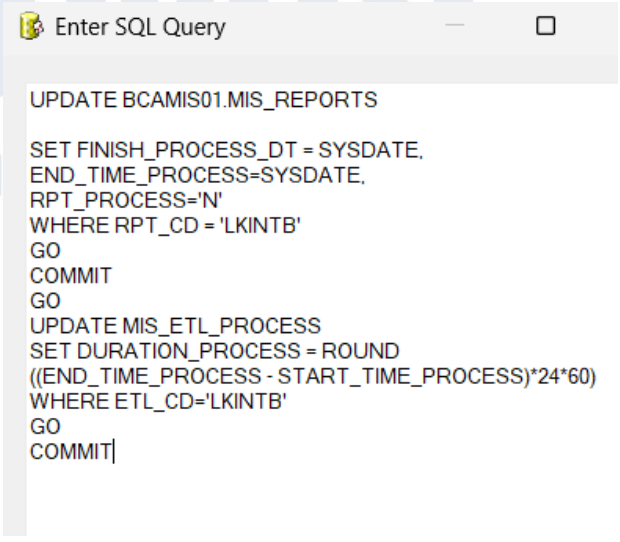
Gambar 3.8 menunjukkan *control flow* pada SSIS yang menangani proses pembuatan laporan pinjaman, dimana dimulai dari *sql task start* sebagai penanda awal proses dan melakukan *update* ke *database* pada tabel mis_*report*s, bahwa ssis telah dijalankan. Lalu, dilanjutkan dengan menjalankan *task drop* tabel pinjaman. Setelah itu, proses dilanjutkan ke *execute SQL task*, yang berfungsi mengeksekusi *query* untuk membentuk tabel atau dataset baru yang diperlukan

dalam laporan pinjaman. Setelah data berhasil diproses, paket masuk ke tahap SQL EMAIL dan *foreach loop container* untuk menjalankan *script task* secara berulang untuk melakukan pengiriman *link* laporan dan pesan email berdasarkan format yang telah dibuat. Setelah seluruh proses selesai dijalankan dalam *loop*, *flow* berakhir pada *task* END.



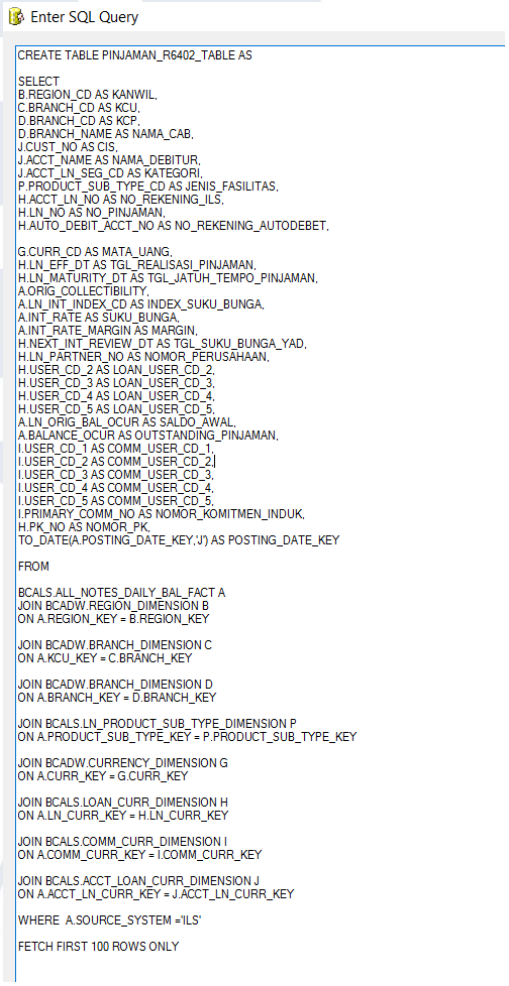
Gambar 3.9 SQL *Query* Pada SQL *Task Start*.

Gambar 3.9 menunjukkan SQL *query* pada SQL *task start*, dimana *query* ini akan melakukan *update* tabel mis_*report*s, dengan mengisi data mulainya proses dengan waktu saat *query* dijalankan dan mengubah status proses dengan "Y". *Query* ini memungkinkan untuk mencatat ke *database*, bahwa *job* SSIS telah dijalankan.



Gambar 3.10 *SQL Query Pada SQL Task End*.

Gambar 3.10 menunjukkan SQL *query* pada SQL *task end*, dimana *query* ini akan melakukan *update* tabel mis_*report*s, dengan mengisi data selesainya proses dengan waktu saat *query* dijalankan dan mengubah status proses dengan "N". Selain itu, juga menghitung durasi proses dengan menghitung selisih antara waktu mulai dan selesainya proses yang dikonversi ke dalam bentuk menit dan dibulatkan.


Enter SQL Query

```
CREATE TABLE PINJAMAN_R6402_TABLE AS

SELECT
B.REGION_CD AS KANWIL,
C.BRANCH_CD AS KCU,
D.BRANCH_CD AS KCP,
D.BRANCH_NAME AS NAMA_CAB,
J.CUST_NO AS CIS,
J.ACCT_NAME AS NAMA_DEBITUR,
J.ACCT_LN_SEG_CD AS KATEGORI,
P.PRODUCT_SUB_TYPE_CD AS JENIS_FASILITAS,
H.ACCT_LN_NO AS NO_REKENING_ILS,
H.LN_NO AS NO_PINJAMAN,
H.AUTO_DEBIT_ACCT_NO AS NO_REKENING_AUTODEBET,

G.CURR_CD AS MATA_UANG,
H.LN_EFF_DT AS TGL_REALISASI_PINJAMAN,
H.LN_MATURITY_DT AS TGL_JATUH_TEMPO_PINJAMAN,
A.ORIG_COLLECTIBILITY,
A.LN_INT_INDEX_CD AS INDEX_SUKU_BUNGA,
A.INT_RATE AS SUKU_BUNGA,
A.INT_RATE_MARGIN AS MARGIN,
H.NEXT_INT_REVIEW_DT AS TGL_SUKU_BUNGA_YAD,
H.LN_PARTNER_NO AS NOMOR_PERUSAHAAN,
H.USER_CD_2 AS LOAN_USER_CD_2,
H.USER_CD_3 AS LOAN_USER_CD_3,
H.USER_CD_4 AS LOAN_USER_CD_4,
H.USER_CD_5 AS LOAN_USER_CD_5,
A.LN_ORIG_BAL_OCUR AS SALDO_AWAL,
A.BALANCE_OCUR AS OUTSTANDING_PINJAMAN,
I.USER_CD_1 AS COMM_USER_CD_1,
I.USER_CD_2 AS COMM_USER_CD_2,
I.USER_CD_3 AS COMM_USER_CD_3,
I.USER_CD_4 AS COMM_USER_CD_4,
I.USER_CD_5 AS COMM_USER_CD_5,
I.PRIMARY_COMM_NO AS NOMOR_KOMITMEN_INDUK,
H.PK_NO AS NOMOR_PK,
TO_DATE(A.POSTING_DATE_KEY,'J') AS POSTING_DATE_KEY

FROM

BCALS.ALL_NOTES_DAILY_BAL_FACT A
JOIN BCADW.REGION_DIMENSION B
ON A.REGION_KEY = B.REGION_KEY

JOIN BCADW.BRANCH_DIMENSION C
ON A.KCU_KEY = C.BRANCH_KEY

JOIN BCADW.BRANCH_DIMENSION D
ON A.BRANCH_KEY = D.BRANCH_KEY

JOIN BCALS.LN_PRODUCT_SUB_TYPE_DIMENSION P
ON A.PRODUCT_SUB_TYPE_KEY = P.PRODUCT_SUB_TYPE_KEY

JOIN BCADW.CURRENCY_DIMENSION G
ON A.CURR_KEY = G.CURR_KEY

JOIN BCALS.LOAN_CURR_DIMENSION H
ON A.LN_CURR_KEY = H.LN_CURR_KEY

JOIN BCALS.COMM_CURR_DIMENSION I
ON A.COMM_CURR_KEY = I.COMM_CURR_KEY

JOIN BCALS.ACCT_LOAN_CURR_DIMENSION J
ON A.ACCT_LN_CURR_KEY = J.ACCT_LN_CURR_KEY

WHERE  A.SOURCE_SYSTEM ='ILS'

FETCH FIRST 100 ROWS ONLY
```

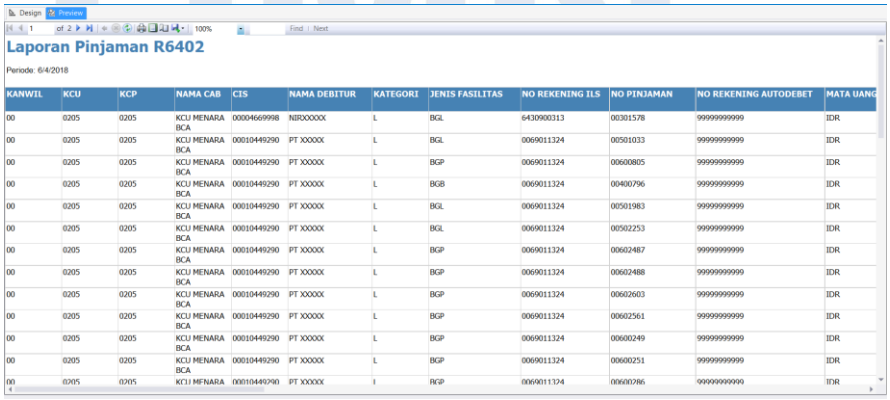
Gambar 3.11 *SQL Query* Pembuatan Tabel Pinjaman.

Gambar 3.11 menunjukkan *SQL query* pada tahap pembuatan tabel pinjaman, yang menjadi sumber utama dalam penyusunan laporan pinjaman. Pada bagian *select*, *query* mengambil berbagai atribut terkait data pinjaman, seperti informasi wilayah, identitas nasabah, jenis fasilitas, kategori kredit, tanggal realisasi, jatuh tempo,

suku bunga, indeks bunga, dan lainnya. Selain itu, kolom lain seperti produk, mata uang, dan detail cabang juga disertakan untuk memberikan gambaran mengenai kondisi pinjaman.

Data tersebut diambil dari *fact table* dan digabungkan (*join*) dengan berbagai tabel dimensi seperti *region, branch, product type, currency, collector, dan loan current dimension*, lalu ditambahkan filter berdasarkan kondisi tertentu untuk memastikan bahwa hanya data pinjaman dari sistem tertentu yang diambil. Hasil akhir *query* ini kemudian digunakan untuk membentuk tabel final yang sudah terstruktur siap digunakan sebagai sumber laporan komitmen di SSRS. Berdasarkan berbagai proses yang dilakukan menggunakan SSIS ini, terdapat perubahan pada data yang digunakan untuk pembuatan laporan pinjaman, dimana diproses sebelum data diproses, data pinjaman masih dari *fact* dan membutuhkan penggabungan (*join*) dengan tabel dimensi lain serta *filter* agar hanya data relevan yang masuk laporan. Setelah diproses, terbentuk tabel pinjaman final untuk SSRS dan *link* laporan dapat didistribusikan otomatis via email.

3.3.1.5 Membuat Tampilan Laporan Pinjaman Menggunakan SSRS dan UAT



KANWIL	KCU	KCP	NAMA CAB	CIS	NAMA DEBITUR	KATEGORI	JENIS FASILITAS	NO REKENING ILS	NO PINJAMAN	NO REKENING AUTODEBET	MATA UANG
00	0205	0205	KCU MENARA BCA	00004669998	NIROXXXX	L	BGL	6430900313	00301578	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGL	0069011324	00501033	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00600805	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGB	0069011324	00400796	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGL	0069011324	00501983	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGL	0069011324	00502253	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00602487	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00602488	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00602603	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00602561	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00600249	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00600251	99999999999	IDR
00	0205	0205	KCU MENARA BCA	00010449290	PT XXXXX	L	BGP	0069011324	00600286	99999999999	IDR

Gambar 3.12 Tampilan Tabel Pinjaman.

Gambar 3.12 menunjukkan tampilan laporan pinjaman yang dibangun menggunakan SSRS, dimana dapat dilihat bahwa terdapat beberapa kolom dan parameter seperti periode laporan tersebut. Selain

itu, terdapat penambahan *watermark*, *header*, dan *footer* terkait informasi laporan pinjaman.

Tabel 3.3.5 Tabel UAT Laporan Pinjaman.

Scenario	Input	Expected Output	Result
Normal	*Run package* dan *report*.	Tabel agunan terbentuk dan *report* pinjaman tampil	Berhasil membuat tabel dan menampilkan *report* pinjaman
Data periode kosong	*Run report*.	Tidak terdapat data pada periode ini.	Muncul pesan "Tidak terdapat data pada periode ini"

Tabel 3.3.5 menunjukkan hasil UAT untuk Laporan Pinjaman dengan dua skenario utama, yaitu skenario normal dan skenario periode data kosong. Pada skenario normal, paket SSIS dan *report* dijalankan dan diharapkan tabel pinjaman terbentuk serta *report* pinjaman dapat ditampilkan, dengan hasil uji berhasil membuat tabel dan menampilkan *report*. Pada skenario periode kosong, *report* dijalankan pada rentang waktu tanpa data dan sistem diharapkan menampilkan informasi bahwa tidak terdapat data pada periode tersebut, dengan hasil uji berupa munculnya pesan "Tidak terdapat data pada periode ini".

Tabel 3.3.6 Estimasi Waktu Proses dan Frekuensi Eksekusi Laporan Pinjaman.

Laporan	Estimasi Waktu Run	Frekuensi
Laporan Pinjaman	10 Menit (Run SSIS dan render SSRS)	Mingguan

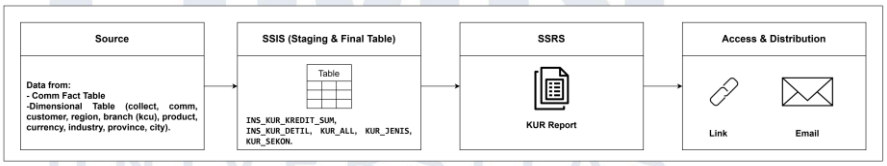
Tabel 3.3.6 menunjukkan estimasi waktu yang dibutuhkan untuk menjalankan proses laporan pinjaman (SSIS hingga render SSRS) serta frekuensi pelaksanaannya, sehingga memudahkan pemantauan kebutuhan operasional dan jadwal distribusi laporan.

3.3.1.6 Merancang Control Flow dan SQL Query Laporan KUR Menggunakan SSIS

Laporan KUR merupakan laporan pemantauan Kredit Usaha Rakyat yang menyajikan data kredit KUR dalam bentuk ringkasan dan detail yang berfungsi untuk mengevaluasi penyaluran dan kualitas kredit. Pada perancangan laporan KUR ini, proses SSIS membentuk tabel summary yang tervalidasi dan tabel turunan per kategori sehingga penyajian data di SSRS lebih cepat dan konsisten. Tabel 3.3.7 menunjukkan alur pengolahan data Laporan KUR mulai dari sumber data, tahapan pemrosesan, hingga output yang dihasilkan.

Tabel 3.3.7 Alur Data Laporan KUR.

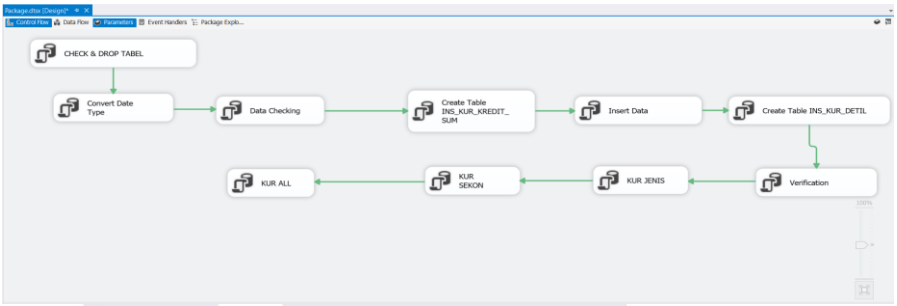
Input (Source)	Process	Output
Data kredit dari berbagai tabel.	convert date type, data checking, pembuatan tabel ins_kur_kredit_ sum, ins_kur_detail, kur all, jenis, dan sekon.	Tabel summary KUR, Tabel detail KUR, tabel kategori (all, jenis, sekon).
Data detail kur, all, sekon, dan jenis (tabel final).	Bangun *report* tabular di SSRS.	Laporan KUR.



Gambar 3.13 Alur Arsitektur Proyek Laporan KUR.

Gambar 3.13 menunjukkan arsitektur proyek Laporan KUR dimulai dari sumber data berupa *fact table* dari tabel komitmen harian dan tabel dimensi (kolektibilitas, komitmen, nasabah, *region*, cabang, KCU, produk, *currency*, industri, provinsi, dan kota). Data tersebut diproses di SSIS untuk membentuk staging dan tabel final, serta tabel turunan klasifikasi (all, sekon, jenis) agar siap dilaporkan. Selanjutnya

SSRS menggunakan tabel – tabel ini untuk menghasilkan KUR *Report*, lalu laporan diakses dan didistribusikan melalui *link* portal SSRS atau email.



Gambar 3.14 *Control Flow* Paket SSIS Laporan KUR.

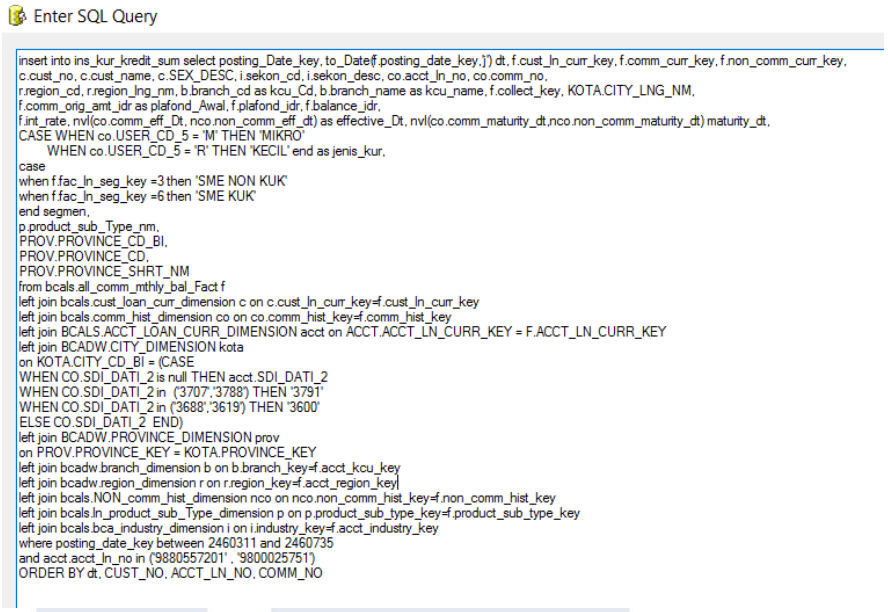
Gambar 3.14 menunjukkan *control flow* pada SSIS yang menangani proses ETL dan *workflow* manajemen untuk pembuatan laporan KUR. Proses dimulai dengan tahap *Check & Drop Tabel* untuk memastikan tabel sementara yang digunakan sebelumnya dihapus, lalu dilanjutkan dengan *Convert Date Type* dan *Data Checking* untuk menyesuaikan format data serta memvalidasi kelengkapan dan konsistensinya. Setelah data siap, sistem membuat tabel agregat KUR melalui *task Create Table ins_kur_kredit_sum* dan mengisi datanya menggunakan *Insert Data*. Selanjutnya, membuat tabel detail KUR *ins_kur_detail* dengan menyalin dan memfilter data dari *ins_kur_kredit_sum*. Lalu, *flow* menjalankan proses *verification* untuk memastikan bahwa seluruh data yang telah diproses dan dimuat sudah sesuai dengan kriteria dan membuat 3 tabel yang memuat data lebih spesifik per kategori, termasuk pembagian berdasarkan keseluruhan data (kur all), per segmen (kur segmen), dan per jenis (kur jenis), sehingga tabel tersebut siap digunakan sebagai sumber laporan KUR. Secara keseluruhan, *flow* ini menggabungkan proses ETL dan pengelolaan alur kerja untuk membangun dataset laporan KUR yang akurat dan terstruktur.

Enter SQL Query

```
create table ins_kur_kredit_sum as
select posting_Date_key, to_Date(f.posting_date_key,'j') dt, f.cust_ln_curr_key, f.comm_curr_key, f.non_comm_curr_key,
c.cust_no, c.cust_name, c.SEX_DESC, i.sekon_cd, i.sekon_desc, co.acct_ln_no, co.comm_no,
r.region_cd, r.region_lng_nm, b.branch_cd as kcu_Cd, b.branch_name as kcu_name, f.collect_key,KOTA.CITY_LNG_NM,
f.comm_orig_amt_idr as plafond_Awal, f.plafond_idr, f.balance_idr,
f.int_rate, nvl(co.comm_eff_Dt, nco.non_comm_eff_dt) as effective_Dt, nvl(co.comm_maturity_dt,nco.non_comm_maturity_dt) maturity_dt,
CASE WHEN co.USER_CD_5 = 'M' THEN 'MIKRO'
      WHEN co.USER_CD_5 = 'R' THEN 'KECIL' end as jenis_kur,
case
when f.fac_ln_seg_key =3 then 'SME NON KUK'
when f.fac_ln_seg_key =6 then 'SME KUK'
end segmen,
p.product_sub_Type_nm,
PROV.PROVINCE_CD_BI,
PROV.PROVINCE_CD,
PROV.PROVINCE_SHRT_NM
from bcals.all_comm_daily_bal_Fact f
left join bcals.cust_loan_curr_dimension c on c.cust_ln_curr_key=f.cust_ln_curr_key
left join bcals.comm_hist_dimension co on co.comm_hist_key=f.comm_hist_key
left join BCALS.ACCT_LOAN_CURR_DIMENSION acct on ACCT.ACCT_LN_CURR_KEY = F.ACCT_LN_CURR_KEY
left join BCADW.CITY_DIMENSION kota
on KOTA.CITY_CD_BI = (CASE
WHEN CO.SDI_DATI_2 is null THEN acct.SDI_DATI_2
WHEN CO.SDI_DATI_2 in  ('3707','3788') THEN '3791'
WHEN CO.SDI_DATI_2 in ('3688','3619') THEN '3600'
ELSE CO.SDI_DATI_2  END)
left join BCADW.PROVINCE_DIMENSION PROV
on KOTA.PROVINCE_KEY = PROV.PROVINCE_KEY

left join bcadw.branch_dimension b on b.branch_key=f.acct_kcu_key
left join bcadw.region_dimension r on r.region_key=f.acct_region_key
left join bcals.NON_comm_hist_dimension nco on nco.non_comm_hist_key=f.non_comm_hist_key
left join bcals.ln_product_sub_Type_dimension p on p.product_sub_type_key=f.product_sub_type_key
left join bcals.bca_industry_dimension i on i.industry_key=f.acct_industry_key
where posting_date_key = 2460888
AND F.IS_dELETED=0
AND F.IS_eXCLUDE=0
and p.product_type_nm='KMUK'
AND f.fac_ln_seg_key IN (3,6)
AND B.BRANCH_CD IN ('0117', '0020')
ORDER BY dt, CUST_NO, ACCT_LN_NO, COMM_NO
```

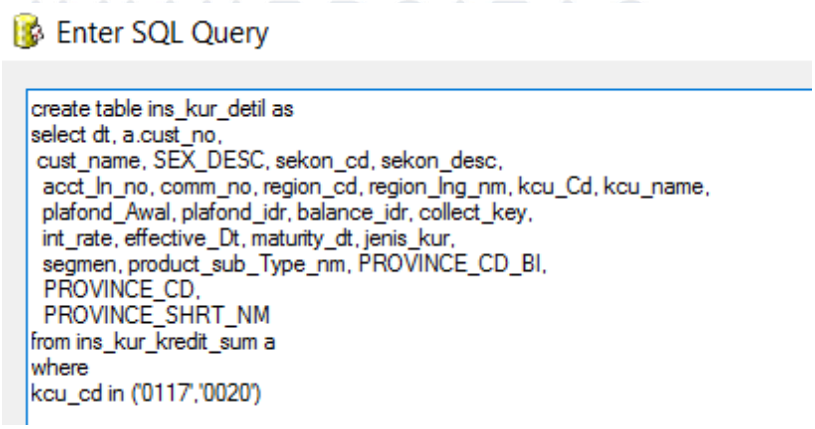
Gambar 3.15 *SQL Query* Pembuatan Tabel Kredit KUR.

Gambar 3.15 menunjukkan *query* pada tahap pembuatan tabel kredit KUR, dimana *query* dimulai dari mengambil (*select*) data dari *fact table* kredit harian, kemudian melengkapinya dengan berbagai atribut dari *dimension table*. Beberapa kolom turunan dibentuk dengan logika *case when* untuk menentukan jenis KUR dan segmen KUR berdasarkan kode *user* dan karakteristik produk, sehingga data dapat dikelompokkan sesuai kebutuhan laporan. Lalu, terdapat filter menggunakan fungsi *where* untuk memfilter hanya kredit yang relevan, misalnya yang termasuk kelompok produk kur/kmk tertentu, masih aktif, dan berasal dari *source system* yang ditentukan, serta membatasi cabang tertentu saja. Hasil akhirnya adalah tabel *summary* kredit KUR yang sudah terstruktur, yang kemudian dipakai sebagai dasar pembuatan laporan KUR di proses SSIS.



Gambar 3.16 *SQL Query* untuk *Insert Data* ke Tabel Kredit KUR.

Gambar 3.16 menunjukkan *query* pada tahap melakukan proses *insert* ke dalam kredit KUR, dengan mengambil data kredit dari *fact table* dan beberapa *dimension table*. Lalu, membentuk membentuk beberapa kolom kategorisasi menggunakan logika *case when*, yaitu kolom jenis kur berdasarkan segmentasi, serta kolom segmentasi tambahan berdasarkan nilai acq_seg_key. Selanjutnya dilakukan proses *join* ke berbagai tabel dimensi. Pada bagian *where*, *query* menyaring hanya data yang relevan, seperti kode produk tertentu yang termasuk kategori tertentuk, data yang tidak berkaitan dengan nomor tertentu, serta transaksi dalam periode tertentu.



Gambar 3.17 *SQL Query* Pembuatan Tabel Detail KUR.

Gambar 3.17 menunjukkan *query* pada tahap pembuatan tabel ins_kur_detil sebagai tabel detail KUR dengan mengambil data dari tabel ins_kur_kredit_sum, sehingga menghasilkan tabel detail yang siap dipakai sebagai sumber utama pembuatan *report*. Pada bagian *select*, dipilih berbagai kolom yang akan digunakan dalam pembuatan laporan KUR. Lalu, terdapat *filter where* berdasarkan kode KCU. Berdasarkan hasil dari pembuatan *ETL flow* menggunakan SSIS pada proyek ini, terdapat perubahan terkait data laporan pinjaman, dimana sebelum diproses, data KUR masih berupa data harian dari fact yang perlu dilengkapi atribut dimensi dan diklasifikasikan berdasarkan jenis agar siap dilaporkan. Setelah diproses, terbentuk tabel terstruktur (summary, detail, dan kategori seperti kur all, sekon, dan jenis) sebagai sumber *report* KUR.

3.3.1.7 Membuat Tampilan Laporan KUR Menggunakan SSRS dan UAT

Gambar 3.18 Tampilan Laporan Pinjaman.

Gambar 3.18 menunjukkan tampilan laporan KUR di SSRS, dimana masing – masing jenis tabel, ditempatkan pada halaman terpisah sehingga yang terlihat hanya KUR JENIS pada halaman pertama. Selain itu, terdapat filter parameter berdasarkan KCU untuk menampilkan data KUR. Pada bagian bawah halaman juga terlihat informasi metadata seperti nama pembuat laporan, waktu *generate*, dan nomor halaman. Tampilan ini menunjukkan bagaimana laporan akan muncul ketika dicetak atau diekspor, sehingga menunjukkan bagaimana laporan akan terlihat ketika digunakan.

Tabel 3.3.8 Tabel UAT Laporan KUR.

Scenario	Input	Expected Output	Result
Normal	Run package dan *report*, masukkan parameter (KCU)	Tabel agunan terbentuk dan *report* KUR tampil	Berhasil membuat tabel dan menampilkan *report* KUR
Data KCU kosong	run *report*	Tidak terdapat data pada KCU ini.	Muncul pesan "Tidak terdapat data pada KCU ini"

Tabel 3.3.8 menunjukkan hasil UAT untuk Laporan KUR dengan dua skenario utama, yaitu skenario normal dan skenario KCU tanpa data. Pada skenario normal, paket SSIS dan *report* dijalankan dengan memasukkan parameter KCU, dan diharapkan tabel KUR terbentuk serta *report* KUR tampil, dengan hasil uji berhasil membuat tabel dan menampilkan *report*. Pada skenario KCU kosong, *report* dijalankan untuk KCU yang tidak memiliki data dan sistem diharapkan menampilkan informasi bahwa tidak terdapat data pada KCU tersebut, dengan hasil uji berupa munculnya pesan "Tidak terdapat data pada KCU ini".

Tabel 3.3.9 Estimasi Waktu Proses dan Frekuensi Eksekusi Laporan KUR.

Laporan	Estimasi Waktu Run	Frekuensi
Laporan KUR	12 Menit (Run SSIS dan render SSRS)	Mingguan

Tabel 3.3.9 menunjukkan estimasi waktu yang dibutuhkan untuk menjalankan proses laporan KUR (SSIS hingga render SSRS) serta frekuensi pelaksanaannya, sehingga memudahkan pemantauan kebutuhan operasional dan jadwal distribusi laporan.

3.3.1.8 Pemahaman Alur Kerja Aplikasi MIRA dan Perencanaan Integrasi API

Pada tahap ini, dilakukan pembahasan dan pemahaman alur kerja (*flow*) aplikasi MIRA yang sebelumnya telah dibangun menggunakan *platform low-code OutSystems*. Aplikasi tersebut telah memiliki sejumlah modul dan proses bisnis dasar, namun sedang melalui tahap pengembangan lanjutan, terutama terkait integrasi dengan API, serta penyempurnaan alur dan tampilan.

Pada langkah awal, dilakukan penelaahan terhadap struktur aplikasi pada OutSystems, termasuk modul – modul utama, *dependency*, *logic flow*, serta halaman – halaman yang terhubung dengan proses bisnis MIRA. Selain mempelajari *flow* yang sudah berjalan, dilakukan pembahasan API yang akan diintegrasikan. API tersebut mencakup beberapa layanan baru yang memerlukan penambahan *endpoint*, dan penyesuaian parameter.

Tahap ini juga mencakup pemahaman mengenai kebutuhan integrasi API yang meliputi proses GET, POST, PUT, maupun DELETE, serta bagaimana API tersebut akan digunakan dalam aplikasi. Lalu, mempelajari contoh *request* dan *response*, struktur *payload*, serta memahami *error handling* yang diperlukan. Pemahaman mendalam terhadap kedua aspek aplikasi dan API menjadi landasan penting sebelum masuk ke tahap pengembangan.

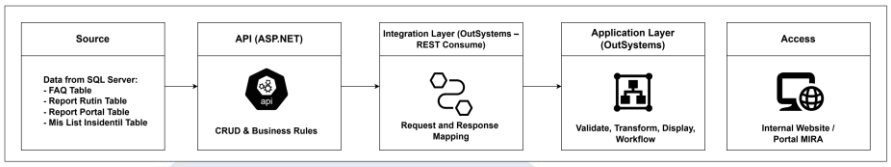
Tabel 3.3.10 Alur Data MIRA.

Method	Input (Source)	Process	Output
GET FAQ	Tabel FAQ.	OutSystems memanggil API GetFAQ lalu mengelompokkan hasil ke kategori rutin, portal, insidentil sebelum ditampilkan.	Daftar FAQ tampil per kategori di halaman User FAQ.
POST FAQ	Created datetime, question, answer, faq type.	Menentukan tipe FAQ dan Memanggil PostFaqAPI untuk menyimpan data baru.	Data FAQ baru tersimpan dan tampil di UI.

PUT FAQ	Created datetime, question, answer, faq type.	Memanggil PutFaqAPI untuk update FAQ.	Data FAQ terbarui dan tampil di UI.
DELETE FAQ	RequestId FAQ.	Memanggil DeleteFaqAPI untuk menghapus data.	Data FAQ terhapus dan list FAQ ter-refresh.
GET REPORT PORTAL	Udomain (role user atau admin) dan data dari tabel report portal.	Memanggil GetreportportalAPI untuk menampilkan data dan dikelompokkan per periode.	List report portal tampil terkelompok per periode di UI.
GET REPORT RUTIN	Udomain (role user atau admin) dan data dari tabel report rutin.	Memanggil GetreportrutinAPI untuk menampilkan data dan dikelompokkan per periode.	List report rutin tampil terkelompok per periode di UI.
GET MIS List Insidentil	Header user-domain dan data dari tabel mis list insidentil.	Memanggil getDataListInsidentil untuk menampilkan data.	Menampilkan daftar data MIS List Insidentil.
Post MIS List Insidentil	Tanggal implementasi, no release, lokasi script, link report, pic, nama pemohon, bpro.	Memanggil PostDataListInsidentil untuk approval data dan ditambahkan ke database.	Status BPRO berubah sesuai approval dan progres bisa dipantau di BPRO utama.
Put MIS List Insidentil	Tanggal implementasi, no release, lokasi script, link report, pic, nama pemohon, bpro.	Memanggil PutDataListInsidentil untuk update status BPRO.	Data penyelesaian tersimpan, status BPRO menjadi Done Implementation.

Tabel 3.3.10 menunjukkan alur *Input – Process – Output* untuk seluruh API yang digunakan pada aplikasi MIRA, mencakup operasi CRUD pada modul FAQ (GET/POST/PUT/DELETE), pengambilan daftar *report* (Report Rutin dan Report Portal), serta *endpoint* MIS List Insidentil (GET untuk mengambil data, POST untuk proses *approval*, dan PUT untuk proses penyelesaian terkait status). Secara umum, setiap baris menunjukkan jenis *request* (input), bagaimana aplikasi memvalidasi dan memprosesnya (misalnya

pemanggilan *endpoint*, pengelompokan data, atau *update* status), dan hasil akhirnya berupa data yang ditampilkan di UI, perubahan status atau progres.

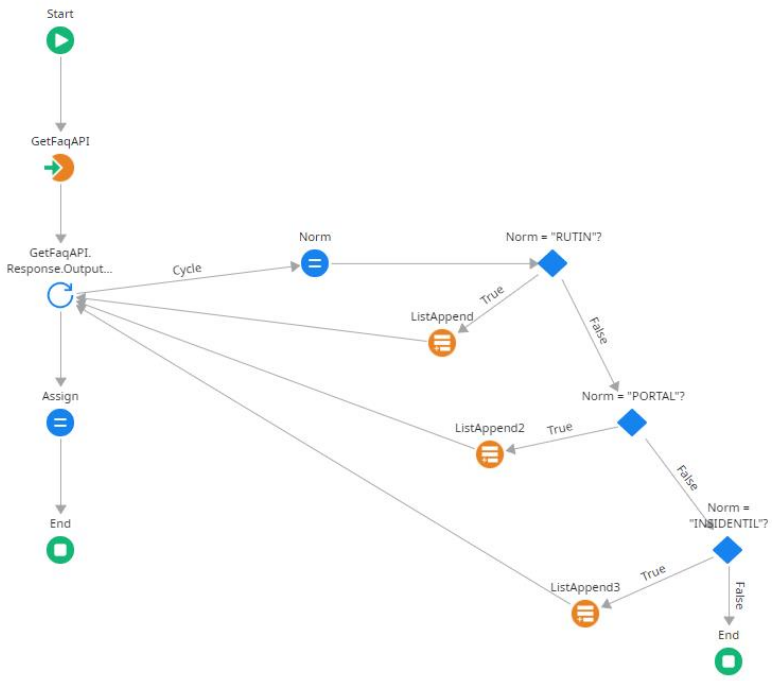


Gambar 3.19 Alur Arsitektur Proyek MIRA.

Gambar 3.19 menunjukkan Arsitektur MIRA API dimulai dari sumber data di SQL Server yang menyimpan tabel FAQ, *report* rutin, *report* portal, dan MIS list insidentil. Data tersebut diakses melalui REST API berbasis ASP.NET (C#) yang menangani operasi CRUD dan aturan bisnis, kemudian dikonsumsi oleh OutSystems pada *integration layer* untuk melakukan pemanggilan *endpoint* serta mapping *request* dan *response*. Selanjutnya *application layer* di OutSystems memproses data (validasi, transformasi, penampilan, dan *workflow* seperti *approval*), dan hasilnya diakses pengguna melalui *website* atau portal internal MIRA.

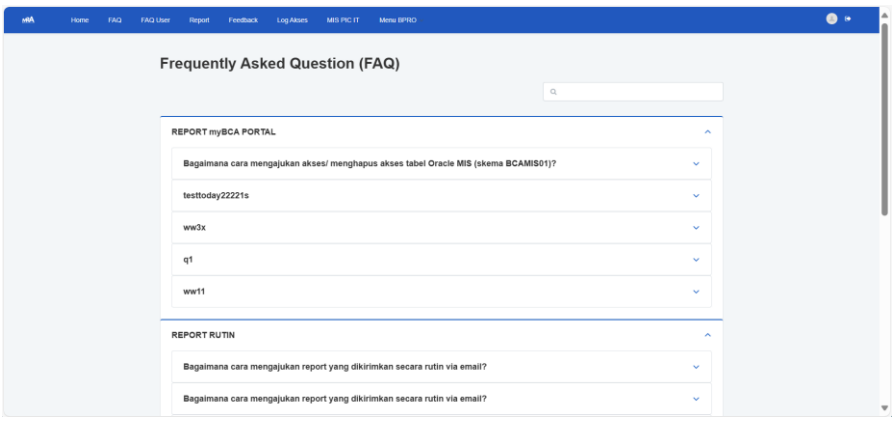
3.3.1.9 Perancangan Flow Integrasi API Pada Halaman User FAQ

Pada tahap ini dilakukan pembuatan *flow OutSystems* yang bertugas mengambil dan menampilkan data FAQ melalui API GetFAQ. Penyesuaian ini diperlukan agar data yang diterima dari API dapat dipisahkan ke dalam tiga kategori, yaitu RUTIN, PORTAL, dan INSIDENTIL, sebelum ditampilkan pada halaman User FAQs.



Gambar 3.20 *Flow* Integrasi API *Get* FAQ dan Pengelompokan FAQ.

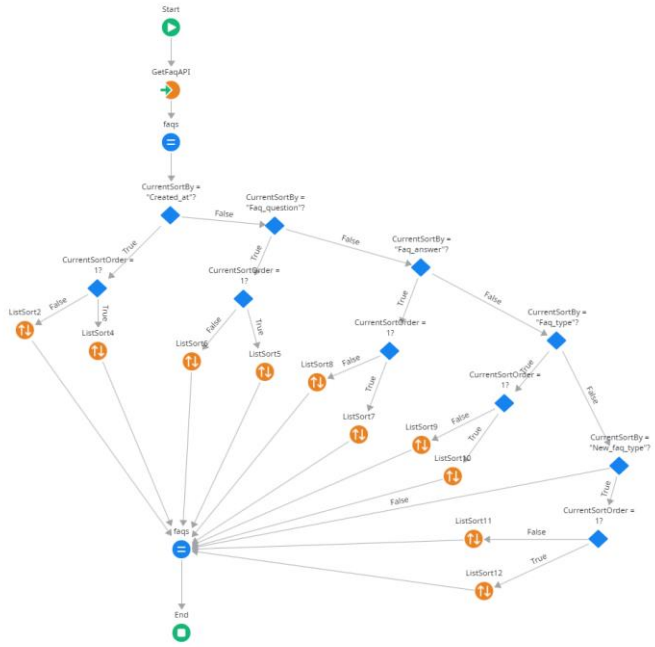
Pada gambar 3.20, proses dimulai dari pemanggilan API GetFAQ melalui *node* GetFaqApi, kemudian setiap data FAQ yang diterima diproses melalui *loop*. Pada setiap iterasi dilakukan pengecekan terhadap atribut kategori, dan berdasarkan hasil pengecekan tersebut data dimasukkan ke dalam list kategori yang sesuai melalui *list append*. Setelah seluruh data selesai diproses, ketiga list ini dikembalikan ke UI dan ditampilkan sebagai kelompok FAQ terpisah.



Gambar 3.21 Tampilan Halaman *FAQ*.

Gambar 3.21 menunjukkan adalah tampilan UI dari halaman FAQ pada aplikasi MIRA setelah proses integrasi dengan API GetFaq berhasil dilakukan. Pada halaman ini, data FAQ yang diterima dari API telah dikelompokkan secara otomatis ke dalam tiga kategori utama, yaitu portal, rutin, dan insidentil.

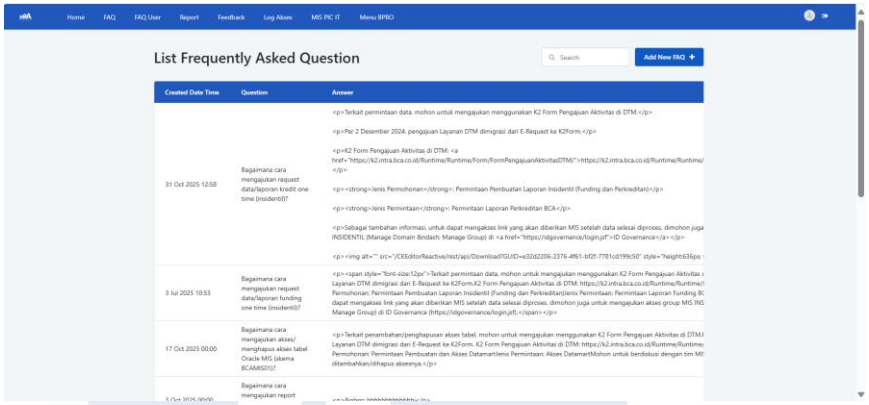
3.3.1.10 Perancangan Flow Integrasi API Pada Halaman Faqdetail



Gambar 3.22 *Flow* LoadFaqs *FAQ Detail.*

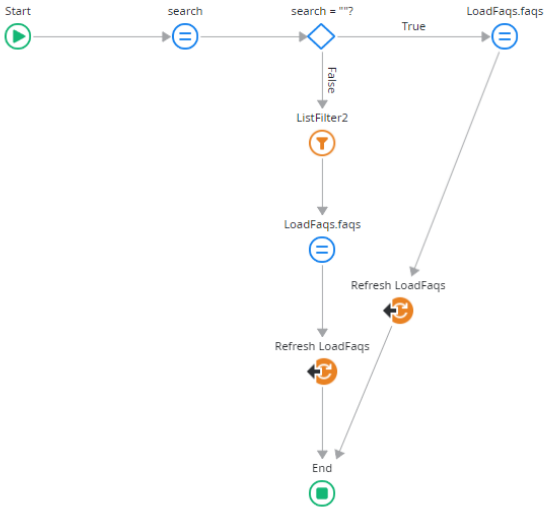
Gambar 3.22 menunjukkan *flow* LoadFaqs yang berfungsi untuk menampilkan data FAQ menggunakan *endpoint* API GetFaqApi. Prosesnya diawali dengan menggunakan *node* GetFaqApi, melalukan *assign* ke variabel faqs dan data FAQ dimasukkan ke proses *sort order*, dimana data diurutkan berdasarkan alfabet yang dikelompokkan fungsinya berdasarkan jenis kolomnya. Fungsi sort order ini dapat dijalankan ketika pengguna menekan nama kolom di halaman FaqDetail dan akan memicu sistem untuk melakukan *refresh* GetFaqAPI pada *event* onSort untuk mengurutkan berdasarkan kolom yang diklik. Lalu

hasil pengambilan data FAQ disimpan di variabel faqs untuk ditampilkan di halaman FaqDetail.



Gambar 3.23 Tampilan Halaman FAQ Detail.

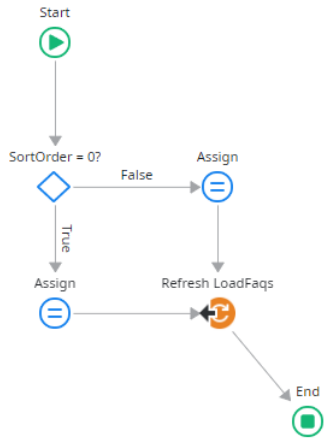
Gambar 3.23 menunjukkan tampilan UI dari halaman FAQ detail pada aplikasi MIRA, dimana terdapat beberapa kolom terkait data FAQ dan ada tombol *add*, *edit*, dan *delete* data faq pada baris tersebut. Fungsi CRUD pada halaman ini akan diintegrasikan dengan API pada tahap berikutnya.



Gambar 3.24 *Flow event* OnSearch.

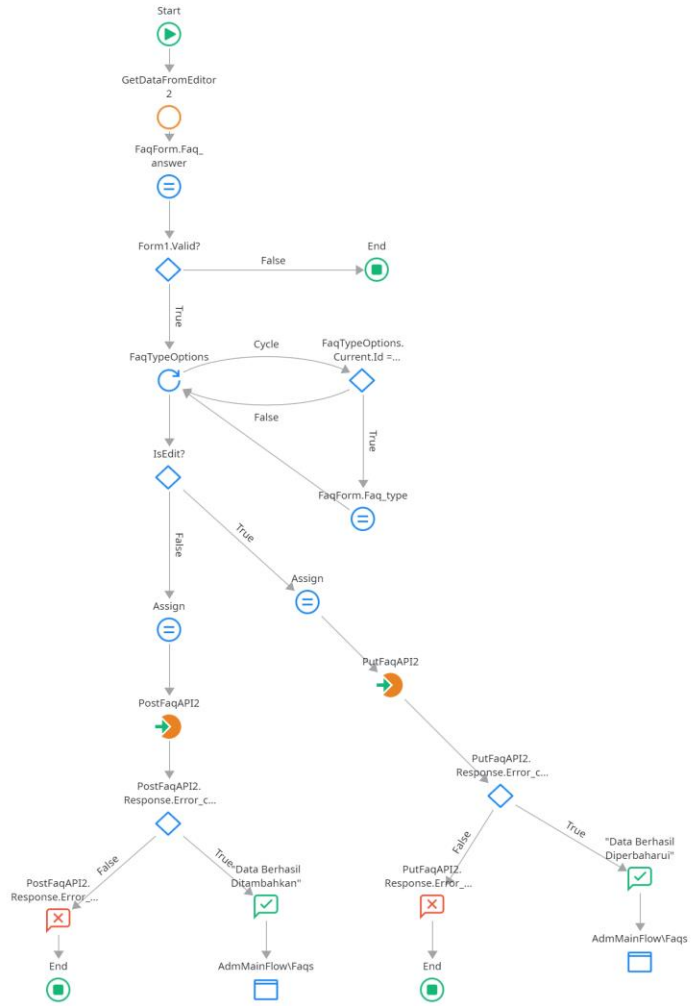
Gambar 3.24 menunjukkan *flow event* OnSearch yang berfungsi untuk menampilkan data yang sesuai dengan *keyword* yang dimasukkan oleh pengguna untuk mencari data tertentu. Proses

diawali dengan *assign* kata kunci yang dimasukkan ke variabel *keyword*, lalu memeriksa jika *keyword* kosong, maka akan menampilkan semua data dengan *refresh node* GetFaqAPI. Apabila *keyword* terdapat sebuah nilai yang dimasukkan, maka data akan difilter sesuai dengan *keyword* yang dimasukkan dan menampilkan data sesuai filter dengan *refresh node* GetFaqAPI.



Gambar 3.25 *Flow event* OnSort.

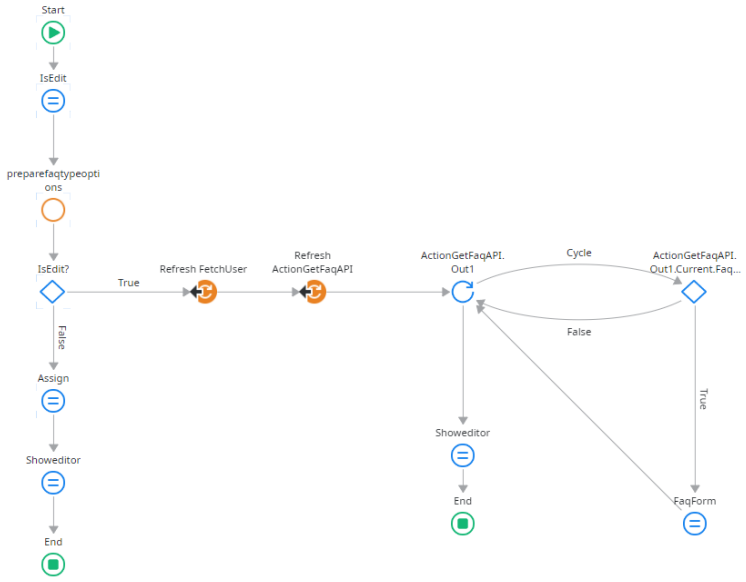
Gambar 3.25 menunjukkan *flow event* OnSort yang berfungsi untuk menampilkan data yang sesuai urutan yang ditentukan ketika pengguna menekan nama kolom tertentu. Proses diawali dengan memeriksa apakah urutan data saat ini *ascending* (0) atau *descending* (1), lalu akan *assign* ke salah satu variabel untuk menukar urutan *ascending* ke *descending* dan sebaliknya, lalu menggunakan *refresh node* GetFaqAPI untuk mengambil data melalui *endpoint* API sesuai dengan urutan yang telah dibuatkan sebelumnya.



Gambar 3.26 *Flow* Integrasi *Post* dan *Put* FAQ.

Pada tahap ini, dilakukan pembuatan alur proses untuk menambahkan (*post*) dan memperbarui (*put*) data FAQ melalui *input form* yang telah tersedia pada aplikasi MIRA. *Flow* ini dimulai ketika sistem mengambil data dari komponen editor untuk mengisi *field* jawaban FAQ. Setelah itu, aplikasi melakukan validasi formulir untuk memastikan seluruh *input* telah terisi dengan benar. Jika validasi gagal, proses dihentikan dan pengguna diminta memperbaiki *input*. Apabila validasi berhasil, sistem melanjutkan dengan menentukan jenis FAQ berdasarkan opsi yang dipilih. Setelah tipe FAQ berhasil dikenali, *flow* melakukan pengecekan

apakah pengguna sedang melakukan proses *edit* atau *tambah data baru*. Jika *form* berada dalam kondisi *edit*, sistem akan melakukan *assign* data kemudian memanggil PutFaqAPI2 untuk memperbarui data FAQ yang sudah ada. Sebaliknya, jika *form* merupakan penambahan data baru, *flow* akan menjalankan PostFaqAPI2 untuk menyimpan data FAQ ke dalam sistem.



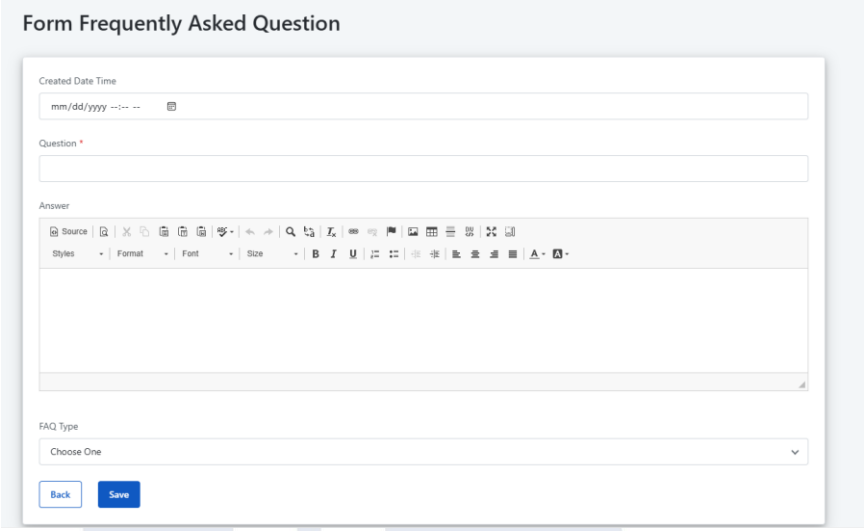
Gambar 3.27 *Flow On Initialize* Pada *Form* FAQ.

Gambar 3.27 menunjukkan *flow* pada *event on initialize* yang berfungsi untuk mengatur tampilan form FAQ, dimana *flow* ini dapat digunakan pada dua kondisi, yaitu *add* dan *edit data* FAQ. Pada awalnya, menginisialisasi variabel isEdit dan memanggil client action preparefaqtypeoption yang berfungsi untuk menyimpan nilai *list drop down* tipe FAQ (portal, rutin, dan insidentil), lalu sistem akan memeriksa apakah *form* dibuka dalam mode *edit* dan mode *add*, dimana pada mode *edit,* data FAQ sudah ada dan pada mode add data masih belum ada. Lalu, jika IsEdit Adalah *True* (*edit data*), maka sistem akan melakukan proses *refresh action* getfaqapi untuk mengambil data FAQ melalui API secara berulang untuk memeriksa nilai FAQ yang sesuai dengan FAQ saat ini yang ingin di-*edit*. Jika

ada, maka nilainya akan disimpan di variabel faqform dan ditampilkan di *form* FAQ untuk *edit data*. Sedangkan, jika IsEdit adalah *False* atau data tidak tersedia, proses pemanggilan GetFaqApi tidak dijalankan dan sistem langsung menampilkan form dalam keadaan kosong untuk memasukkan data yang baru (*add data*), sehingga integrasi API hanya dilakukan saat memang diperlukan untuk melakukan *edit data*.

Gambar 3.28 *Flow Client Action* PrepareFaqTypeOption.

Gambar 3.28 menunjukkan *flow* yang berfungsi untuk mempersiapkan *list* pilihan FAQ *type*, dimana prosesnya diawali dengan ListClear yang berfungsi untuk mengosongkan *list* agar menghindari adanya duplikasi data saat halaman diinisialisasi ulang. Selanjutnya dilakukan *assign* untuk mengatur nilai awal yang diperlukan. Setelah itu, sistem menambahkan item pilihan satu per satu menggunakan ListAppend sesuai dengan jenis FAQ nya, sehingga terbentuk list final berisi seluruh jenis FAQ yang tersedia. Hasil *list* ini digunakan sebagai *data source dropdown* untuk FAQ *type*, sehingga pilihan tipe FAQ sudah siap ditampilkan ketika halaman *form* FAQ dibuka.

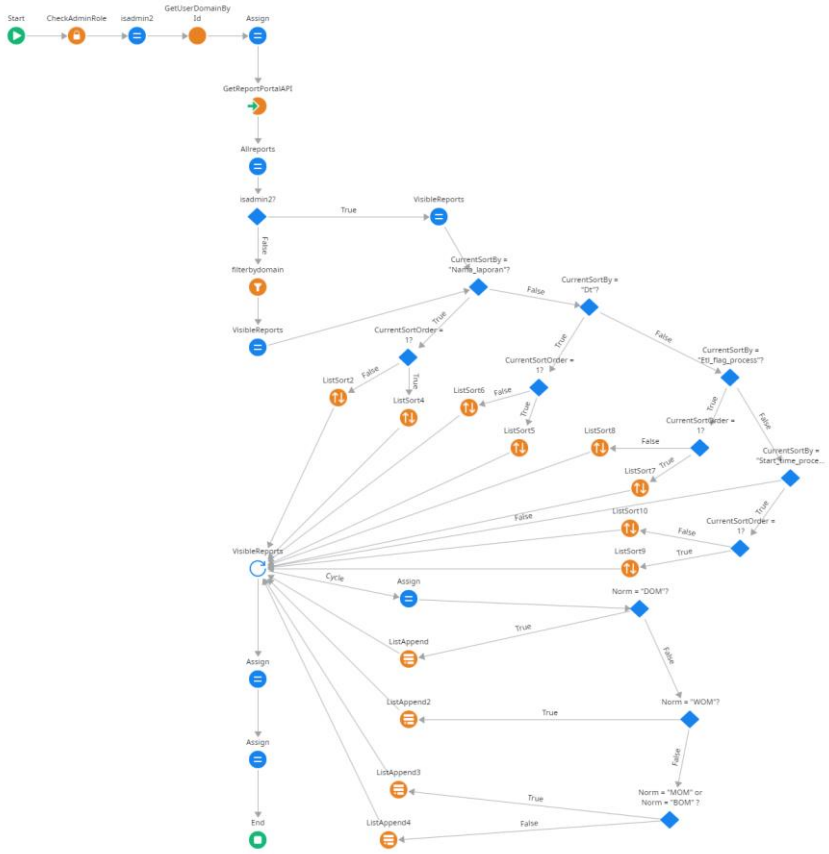


Gambar 3.29 Tampilan Form FAQ.

Gambar 3.29 menunjukkan tampilan UI dari *form* FAQ untuk menambahkan data FAQ (post) atau mengedit data FAQ (put), dimana terdapat *input form* untuk *created date time*, *question*, *answer*, dan *faq type*. Lalu, terdapat tombol *save* untuk melakukan penyimpanan terhadap perubahan data (*edit data*) atau penambahan data (*add data*).

3.3.1.11 Perancangan Flow Integrasi API Pada Halaman *Report*s

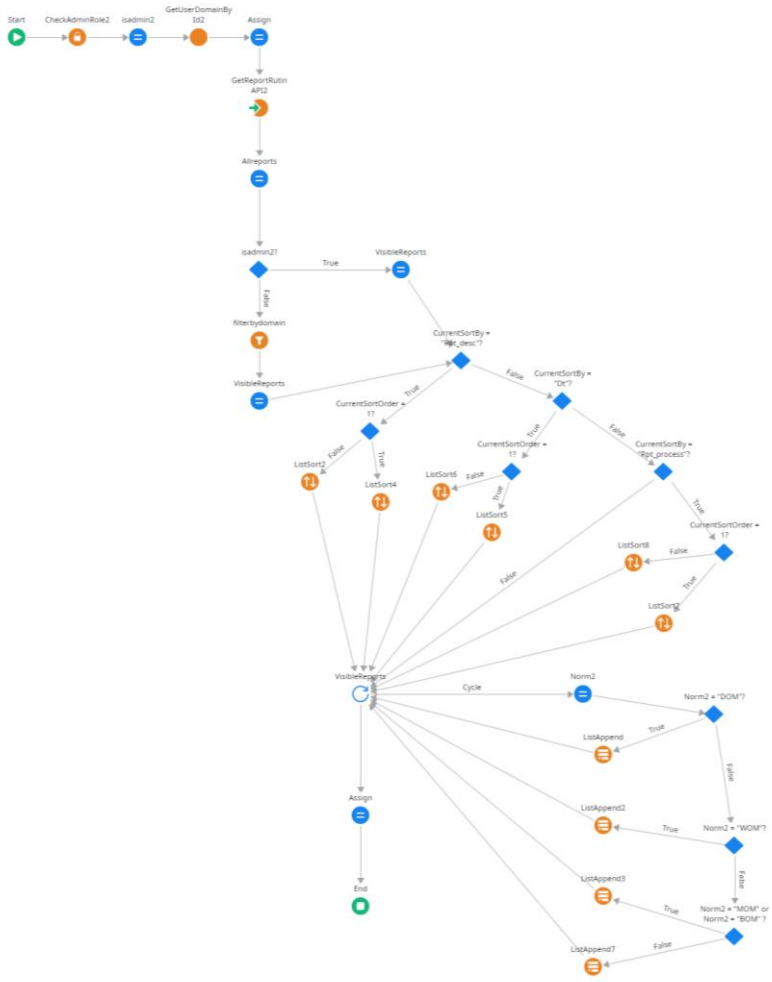
Pada tahap ini, dilakukan pembuatan *flow* untuk menampilkan data *report* menggunakan API, dimana data *report* ini terbagi menjadi 2 yaitu data *report* portal dan *report* rutin, lalu untuk akses datanya terbagi berdasarkan akun *admin* atau *user*, serta datanya dikelompokkan berdasarkan beberapa jenis periode.



Gambar 3.30 *Flow* Integrasi *Get Report* Portal dan Pengelompokan Periode *Report.*

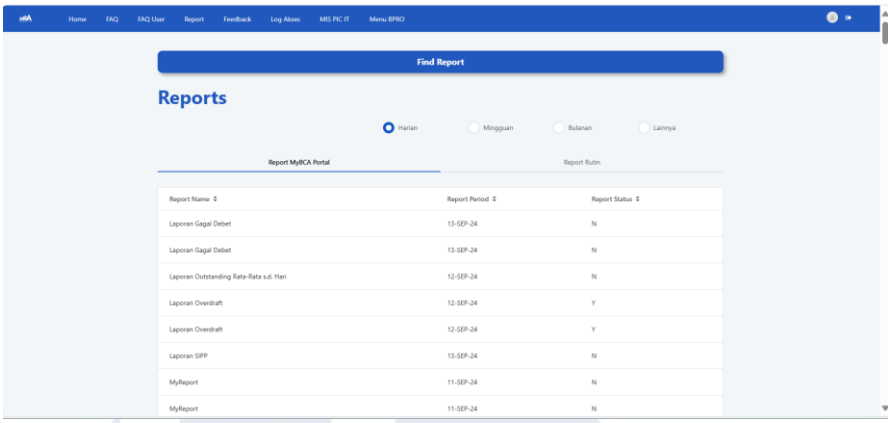
Gambar 3.30 merupakan *flow* untuk menampilkan data *report portal*. *Flow* ini dimulai dari melakukan pemeriksaan akun yang *login* saat ini adalah akun *user* atau *admin* berdasarkan udomain. Lalu, data *reports* portal dimasukkan ke proses *sort order*, dimana data diurutkan berdasarkan alfabet yang dikelompokkan fungsinya berdasarkan jenis kolomnya. Fungsi *sort order* ini dapat dijalankan ketika pengguna menekan nama kolom di halaman *report*. Hal tersebut memicu sistem untuk melakukan *refresh* Get*Report*Rutin API pada *event* onSort untuk mengurutkan berdasarkan kolom yang diklik. Selanjutnya, *looping* dilakukan pengecekan terhadap atribut kategori data dan menggunakan *list append* untuk memasukkan data ke *list* periodenya masing – masing yaitu DOM, WOM, BOM, MOM, dan QOY. Setelah

seluruh data selesai diproses, *list* tersebut dimasukkan ke variabel dan akan digunakan untuk menampilkan data di UI sebagai kelompok *report* terpisah.



Gambar 3.31 *Flow* Integrasi *Get Report* Rutin dan Pengelompokan Periode *Report.*

Gambar 3.31 menunjukkan *flow* untuk menampilkan data *report* rutin menggunakan Endpoint API melalui *node* Get*Report*Rutin, dimana untuk proses sama dengan menampilkan *report* portal, hanya memiliki perbedaan di bagian jenis node API yang digunakan, dimana menggunakan node GetFaq*Report*Rutin dan data yang ditampilkan dikelompokkan ke dalam *list* periodenya masing – masing yaitu DOM, WOM, BOM, MOM, dan QOY.

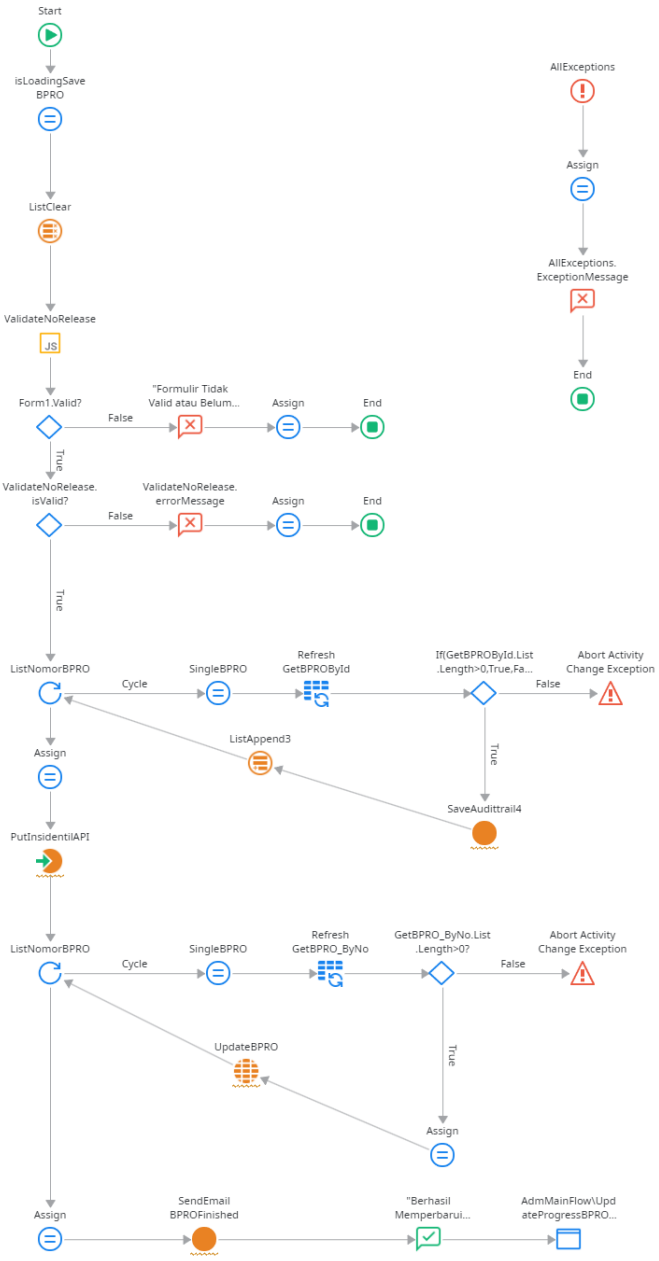


Gambar 3.32 Tampilan Halaman *Report* Portal dan Rutin.

Gambar 3.32 menunjukkan UI atau tampilan dari salah satu halaman *reports* ketika berhasil menampilkan data *report* menggunakan API pada periode tertentu, dimana terbagi menjadi *report* portal dan rutin, serta terbagi dengan beberapa periode yaitu harian, mingguan, bulanan, dan lainnya.

3.3.1.12 Perancangan Flow Integrasi API Pada Form Finish BPRO

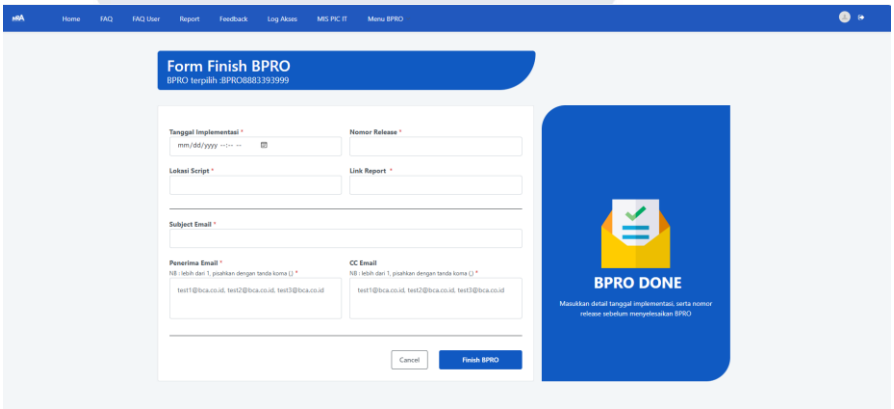
Pada tahap ini, dilakukan pembuatan *flow* untuk integrasi API PutInsidentil, dimana *form finish* bertujuan untuk ketika proses pengerjaan proyek sudah selesai, maka harus mengisi *form* ini agar statusnya menjadi *done*.



Gambar 3.33 *Flow* Integrasi *Put* Insidentil dan Perubahan Status BPRO.

Gambar 3.33 ini menunjukkan *flow* terkait proses bisnis ketika kepala tim menekan tombol *Finish* BPRO dan mengisi *Form Finish* BPRO untuk menyatakan bahwa suatu proyek telah selesai diimplementasikan. Setelah tombol *finish* diklik, sistem terlebih dahulu melakukan pembersihan *list* dan menjalankan beberapa tahapan validasi, seperti mengecek apakah *form* sudah terisi dengan

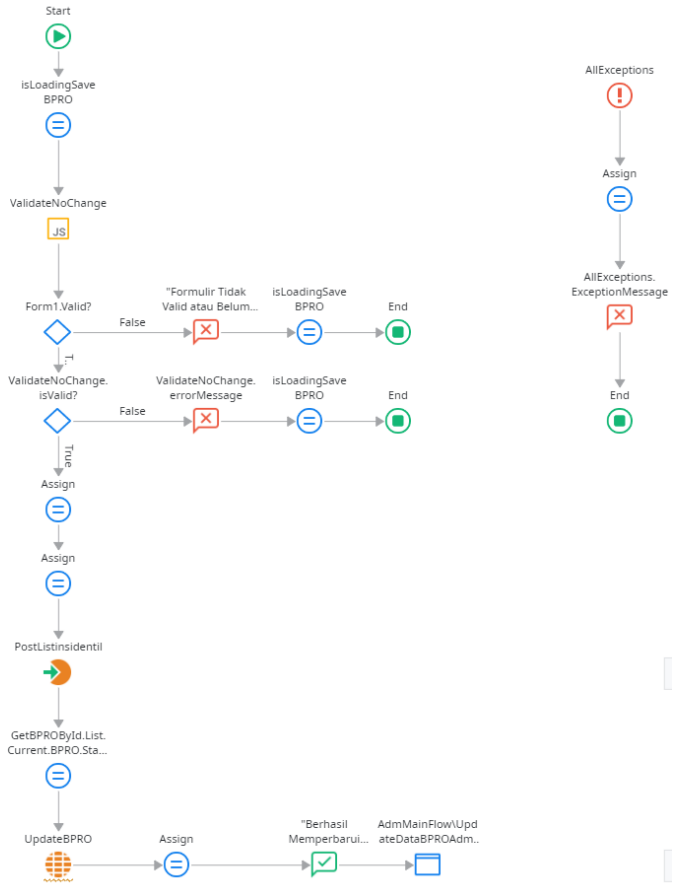
benar dan apakah nomor *release* yang dimasukkan *valid*. Jika terdapat kesalahan, sistem akan menampilkan pesan *error* dan proses dihentikan. Jika seluruh isian *valid*, proses akan lanjut ke *looping* pertama yang digunakan untuk memeriksa setiap nomor *request* BPRO yang diinput, memastikan datanya *valid* dan tersedia sebelum sistem mengirimkan *PUT API* yang menyimpan informasi penyelesaian proyek seperti tanggal implementasi, nomor *release*, lokasi *script*, dan *link* laporan. Setelah API berhasil dijalankan, *looping* kedua digunakan untuk memperbarui status masing – masing BPRO menjadi *Done Implementation* dan kemudian mengirimkan email kepada pihak terkait. Secara keseluruhan, dua *looping* ini memastikan bahwa setiap BPRO diverifikasi, diperbarui, dicatat melalui *audit trail*, dan diinformasikan ke pengguna.



Gambar 3.34 Tampilan Halaman *Form Finish* BPRO.

Gambar 3.34 menunjukkan UI atau tampilan *form finish* BPRO, dimana kepala tim dapat mengisi data terkait proyek yang dikerjakan untuk disetujui bahwa proyek itu telah selesai dan akan mengubah status menjadi *done implementation* di halaman BPRO utama. Bagian atas *form* berisi *field* yang nilainya dikirim melalui PUT API, yaitu tanggal implementasi, nomor *release*, lokasi *script*, dan link *report* yang menggambarkan detail penyelesaian proyek. Sementara itu, bagian bawah *form* berisi *input* terkait pengiriman *email*, seperti *subject*, daftar penerima dan *CC email*.

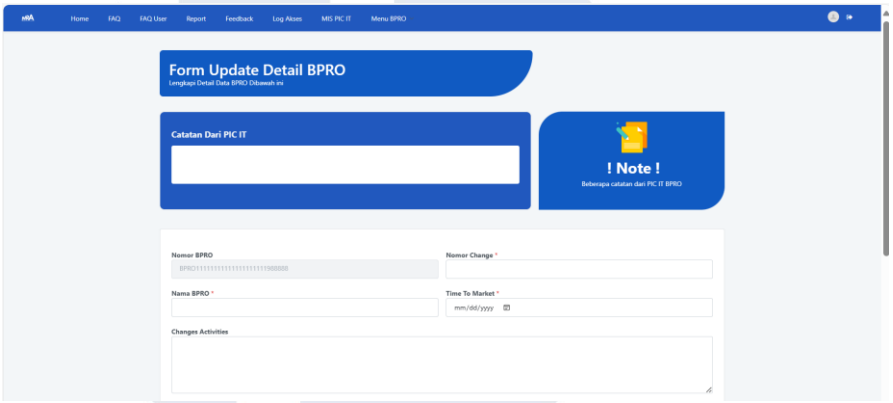
3.3.1.13 Perancangan Flow Integrasi API Pada Form Update Detail BPRO



Gambar 3.35 *Flow* Integrasi *Post* Insidentil dan Status *Approval* BPRO.

Gambar 3.35 ini menunjukkan *flow PostList Insidentil* pada *Form Update Detail* BPRO digunakan dalam proses *approval* terhadap proyek yang diajukan ke tim MIS. Ketika kepala tim atau pihak yang berwenang melakukan *update detail* dan *approval*, sistem terlebih dahulu melakukan validasi *form* untuk memastikan seluruh *input* yang diperlukan telah terisi dan tidak ada data yang salah. Jika *form* tidak *valid*, proses dihentikan dan pesan kesalahan ditampilkan. Namun jika *valid*, *flow* berlanjut ke proses *PostListInsidentil API* yang mengirim data persetujuan tersebut ke *backend* sebagai dasar perubahan status BPRO. Setelah itu sistem mengambil data BPRO terbaru menggunakan *GetBPROById* untuk memastikan status yang

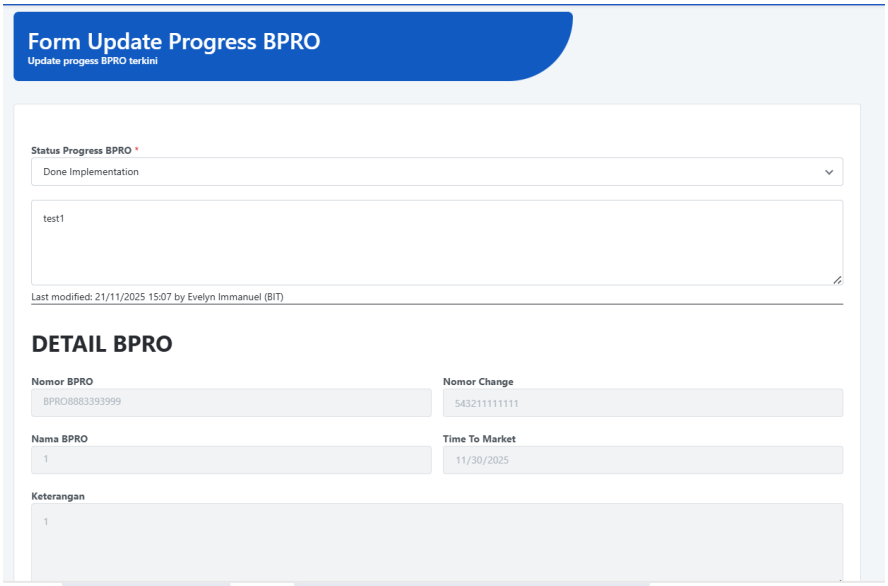
dikirim sudah diterapkan dengan benar. Jika proyek disetujui, sistem akan menjalankan *UpdateBPRO* untuk mengubah status BPRO dari "*Waiting for Approval"* menjadi "*In Backlog*", yang menandakan bahwa proyek tersebut resmi masuk antrean pengerjaan tim MIS. Perubahan status ini memungkinkan BPRO mulai dikerjakan, dipantau progresnya, dan diperbarui tahap demi tahap sesuai *flow* pengembangan yang berlaku.



Gambar 3.36 Tampilan Halaman *Form Update Detail* BPRO.

Gambar 3.36 menunjukkan UI atau tampilan *form update detail* BPRO, dimana *form* ini diisi untuk menyetujui atau menolak proyek yang ditujukan kepada tim MIS. Jika telah disetujui, maka BPRO akan bisa dikerjakan dan diperbaharui status pengerjaannya.

3.3.1.14 Menambahkan Fitur Notes dan Last modified



Gambar 3.37 Tampilan Halaman *Form Update Progress* BPRO.

Gambar 3.37 menunjukkan tampilan halaman *form update progress* BPRO yang berfungsi dilakukan penambahan fitur untuk membuat kolom komentar atau *notes* ketika melakukan *update* pada status pengerjaan proyek. Selain itu, juga menambahkan informasi terkait *last modified* dan nama akun *user* yang memberikan komentar atau *notes*. Dalam melakukan penambahan fitur ini, dilakukan penambahan beberapa kolom baru di *database outsystems* sesuai dengan keperluan.

3.3.1.15 Perancangan Endpoint GET Data Insidentil

```
    }
}
[SwaggerResponse(HttpStatusCode.OK, Type = typeof(IEnumerable<GetMisListInsidentil>))]
[Route("mira/mis-list-insidentil")]
0 references
public IHttpActionResult GetListInsidentil()
{
    try
    {
        string userId = "";
        if (Request.Headers.Contains("user-domain") && Request.Headers.GetValues("user-domain").Count() > 0)
        {
            userId = Request.Headers.GetValues("user-domain").First();
        }
        else
        {
            return Json(Output.generateOutput("MIS-001", "Header user-domain tidak ditemukan", "user-domain header not found", null));
        }

        List<GetMisListInsidentil> list = GetMisListInsidentil.getDataListInsidentil();
        if (list.Count() > 0)
        {
            //Historyconn.insertLog(userid, "API List Insidentil", "");
            return Json(Output.generateOutput("MIS-000", "Berhasil", "Success", list.ToArray()));
        }
        else
        {
            //Historyconn.insertLog(userid, "API List Insidentil", "");
            return Json(Output.generateOutput("MIS-002", "Data tidak ditemukan", "Data not found", list.ToArray()));
        }
    }
    catch (Exception e)
    {
        return Content<Output>(HttpStatusCode.InternalServerError, Output.generateOutput("MIS-003", "Maaf, transaksi anda tidak dapat diproses", "Sorry, your transaction can not be processed", null));
    }
}
```

Gambar 3.38 Kode *Controller Endpoint* API Get List Insidentil.

Gambar 3.38 menunjukkan kode untuk membuat *endpoint* API getlistinsidentil() pada *controller layer*, dimana *endpoint* ini menyediakan layanan untuk mengambil data melalui *route mira-list-insidentil* yang terdokumentasi di *Swagger*. Awalnya, controller akan memvalidasi berdasarkan *user-domain*, dimana jika *header* tersebut ada dan memiliki nilai, nilainya diambil dan disimpan ke variabel dan jika tidak ada, maka proses akan berhenti dan mengembalikan *response* JSON sebagai pesan *error*. Apabila validasi *header* berhasil *controller* memanggil *method* untuk mengambil data daftar insidentil dari basis data, kemudian memeriksa apakah *list* hasil *query* memiliki isi. Bila jumlah data lebih dari nol, API mengembalikan *response* sukses dan *payload* berupa *array* datanya.

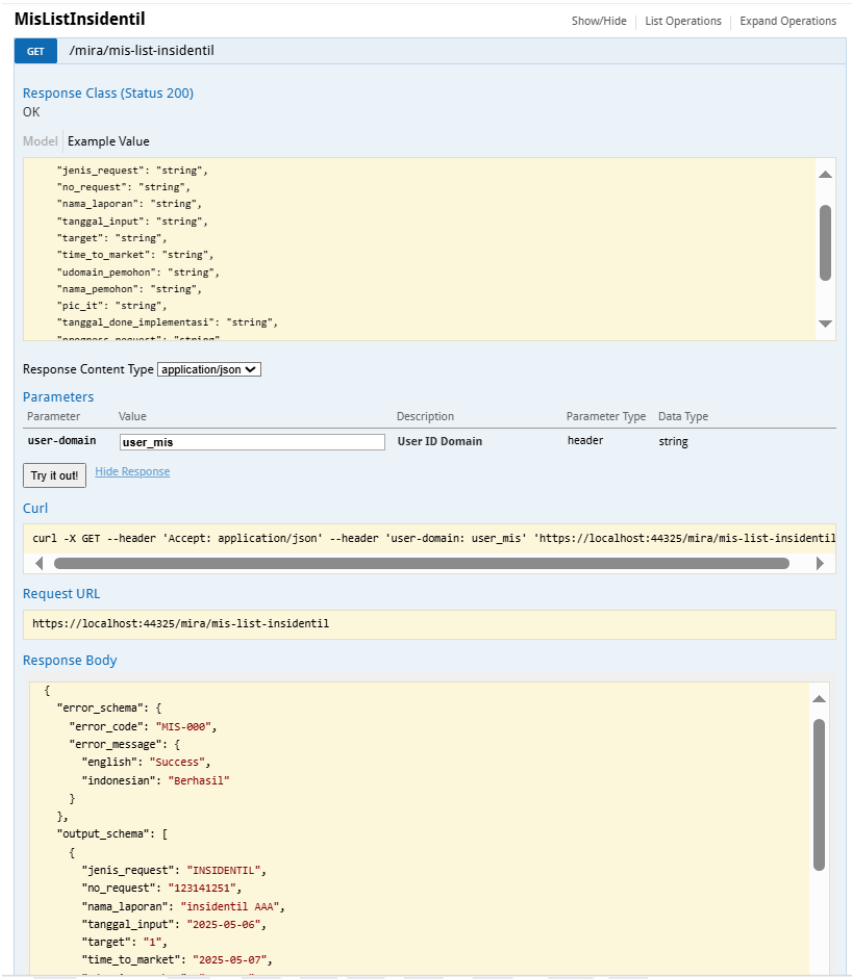
3.3.1.16 Pembuatan Method Pengambilan Data Insidentil Dan Pengujian Endpoint API

```
1 reference
public static List<GetMisListInsidentil> getDataListInsidentil()
{
    OracleDBContext context = new OracleDBContext();

    var sql = @"
SELECT
    mli.JENIS_DESC AS JENIS_REQUEST,
    mli.NO_REQUEST,
    mli.KETERANGAN AS NAMA_LAPORAN,
    TO_CHAR(mli.TANGGAL_INPUT, 'YYYY-MM-DD') AS TANGGAL_INPUT,
    mli.TARGET,
    TO_CHAR(mli.TANGGAL_INPUT + mli.TARGET, 'YYYY-MM-DD') AS TIME_TO_MARKET,
    mli.UDOMAIN_PEMOHON,
    mli.NAMA_PEMOHON,
    mau.USER_NAME AS PIC_IT,
    TO_CHAR(mli.TANGGAL_DONE_IMPLEMENTASI, 'YYYY-MM-DD') AS TANGGAL_DONE_IMPLEMENTASI,
    CASE WHEN mli.TANGGAL_DONE_IMPLEMENTASI IS NULL THEN 'In Backlog' ELSE 'Done Delivered' END AS PROGRESS_REQUEST,
    CASE WHEN mli.TANGGAL_DONE_IMPLEMENTASI IS NULL THEN 'In Backlog' ELSE 'Done' END AS STATUS
FROM MIS_LIST_INSIDENTIL mli
LEFT JOIN MIS_API_USERS mau ON mli.PIC_INITIAL = mau.INISIAL
WHERE mau.SUBTIM = 'R'
  AND mli.JENIS_DESC IN ('AKSES','INSIDENTIL')
  AND (mli.TANGGAL_DONE_IMPLEMENTASI IS NULL
       OR mli.TANGGAL_INPUT >= TRUNC(SYSDATE) - 360)
ORDER BY mli.TANGGAL_INPUT DESC";


    List<GetMisListInsidentil> list = context.Database.SqlQuery<GetMisListInsidentil>(sql).ToList();

    return list;
}
```

Gambar 3.39 Kode *Method* Pengambilan Data Insidentil.

Gambar 3.39 menunjukkan *method* getDataListInsidentil pada sisi model yang berperan sebagai pengakses data ke basis data *Oracle* dan mengembalikan *list* objek GetMisListInsidentil. Data diambil menggunakan *Raw* SQL *Query*, dimana mengambil data dari beberapa kolom berdasarkan filter tertentu menggunakan *case when* dan *where*. Seluruh hasil eksekusi *query* kemudian dikonversi menjadi

list objek GetMisListInsidentil dan dikembalikan ke *controller* untuk selanjutnya ditampilkan melalui *endpoint* API.



Gambar 3.40 Pengujian *Endpoint* API Menggunakan Swagger.

Gambar 3.40 menunjukkan penggunaan *swagger* untuk melakukan uji coba atau *testing endpoint* API Get MisListInsidentil, dimana pengujian dilakukan dengan mengisi *user domain* pada bagian *header* sebagai identitas *domain* pengguna. Setelah itu, *Swagger* mengirim request GET ke *endpoint* yang dituju dan menampilkan detail *request* URL. Hasil pengujian kemudian terlihat pada bagian Response Body dalam format JSON, yang menunjukkan status respon berhasil beserta dengan data *output* berupa daftar informasi *list* insidentil yang dikembalikan oleh API.

Tabel 3.3.11 Tabel UAT *Endpoint* API GET MisListInsidentil.

Scenario	Input	Expected Output	Result
Normal	Header (user-domain)	Sistem membentuk URL *report* dan menampilkan *report* via RptUrl.	MIS-000, "success".
Header tidak ada	Header (user-domain)	Menampilkan pesan error.	MIS-001, "user-domain header not found".
Data kosong	Header (user-domain)	Menampilkan pesan error.	MIS-002, "Data not found"
Exception error	Header (user-domain)	Menampilkan pesan error.	MIS-003, "Sorry, your transaction can not be processed".

Tabel 3.3.11 menunjukkan hasil UAT untuk *endpoint* API (Get List Insidentil) dengan empat skenario utama, yaitu normal, *header* tidak ada, data kosong, dan *exception error*. Pada skenario normal, *request* dengan *header* user-domain yang valid menghasilkan respons sukses (MIS-000) dan sistem menampilkan data sesuai parameter yang dibentuk. Jika *header user-domain* tidak dikirim, sistem mengembalikan MIS-001 dengan pesan "user-domain header not found". Jika *header* valid namun data tidak ditemukan, sistem mengembalikan MIS-002 dengan pesan "Data not found". Jika terjadi kegagalan proses (*exception*), sistem mengembalikan MIS-003 dengan pesan "Sorry, your transaction can not be processed".

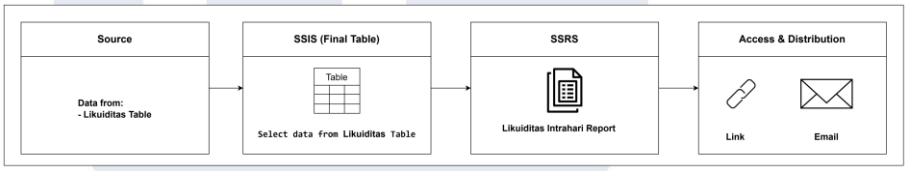
3.3.1.17 Merancang Control Flow, SQL Query, Tampilan Laporan Likuiditas Intrahari, dan UAT.

Laporan Likuiditas Intrahari adalah laporan kondisi likuiditas harian yang dibutuhkan untuk pemantauan posisi dana secara cepat dalam periode waktu tertentu. Pembuatan laporan ini melalui perancangan *flow* ETL menggunakan SSIS untuk pengiriman *link*

laporan melalui email dan penyajian *report* menggunakan SSRS. Tabel 3.3.12 menunjukkan alur pengolahan data Laporan Likuiditas Intrahari mulai dari sumber data, tahapan pemrosesan, hingga *output* yang dihasilkan.

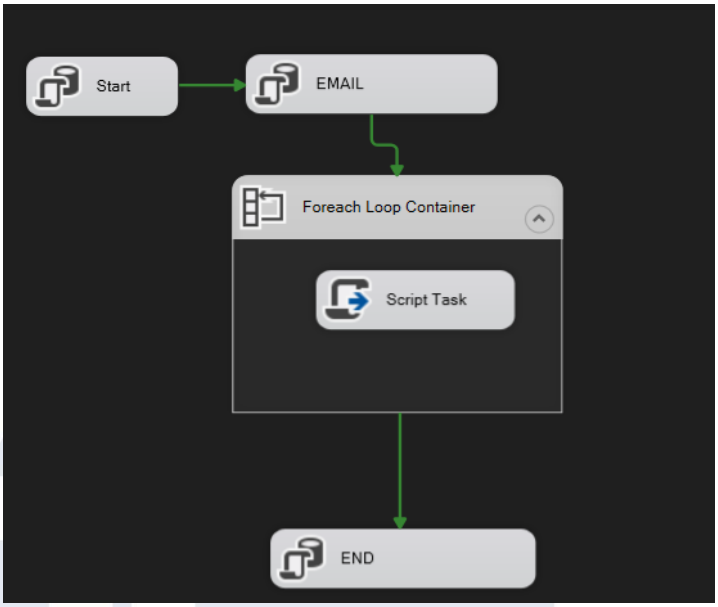
Tabel 3.3.12 Alur Data Laporan Likuiditas Intrahari.

Input (Source)	Process	Output
Data Likuiditas Intrahari	Mengubah format dan nama kolom, serta mangun *report* tabular di SSRS	Laporan Likuiditas Intrahari



Gambar 3.41 Alur Arsitektur Proyek Laporan KUR.

Gambar 3.41 menunjukkan arsitektur proyek Laporan Likuiditas Intrahari dimulai dari sumber data berupa tabel likuiditas yang menjadi acuan perhitungan posisi dana intrahari. SSIS kemudian mengambil data dari tabel tersebut untuk membentuk tabel final yang siap dibaca oleh SSRS. Selanjutnya SSRS menghasilkan Likuiditas Intrahari *Report*, dan hasil laporan didistribusikan kepada pengguna melalui *link* portal SSRS serta pengiriman email sesuai kebutuhan.



Gambar 3.42 *Control Flow* Paket SSIS Laporan Likuiditas Intrahari.

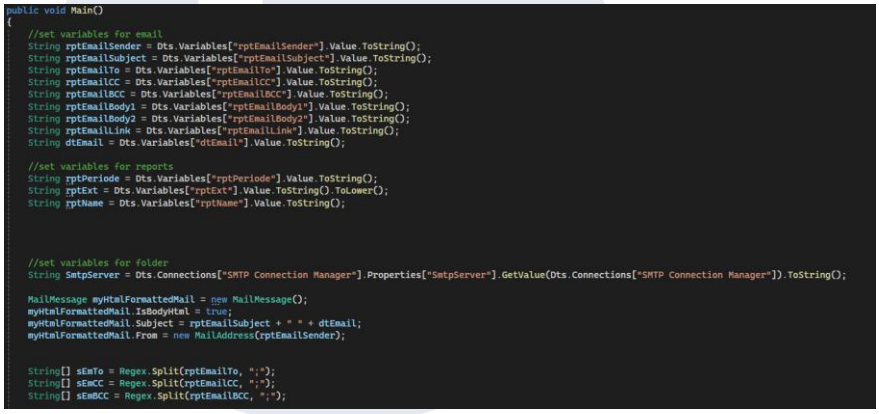
Gambar 3.42 menunjukkan *control flow* pada SSIS yang menangani proses pembuatan laporan likuiditas intrahari, dimana dimulai dari komponen *start* sebagai penanda awal proses dan melakukan *update* ke *database* pada tabel mis_*report*s, bahwa ssis telah dijalankan. Lalu, dilanjutkan dengan menjalankan SQL *task* email dan *foreach loop container* untuk menjalankan *script task* secara berulang untuk melakukan pengiriman *link* laporan dan pesan email berdasarkan format yang telah dibuat. Setelah seluruh proses selesai dijalankan dalam *loop*, *flow* berakhir pada *task* END.

Enter SQL Query

```
SELECT
RPT_KEY,
TRIM(TO_CHAR(dt,'dd')) || ' '
|| TRIM(TO_CHAR(dt,'Mon')) || ' '
|| TRIM(TO_CHAR(dt,'yyyy'))
DT_EMAIL,
RPT_EMAIL_TO,
RPT_EMAIL_CC,
RPT_EMAIL_BCC,
RPT_EMAIL_LINK,
RPT_PERIODE,
RPT_EXT
FROM MIS_REPORTS
WHERE RPT_CD = 'LKINTB'
```

Gambar 3.43 SQL *Query* Untuk Mengambil Data Email Laporan.

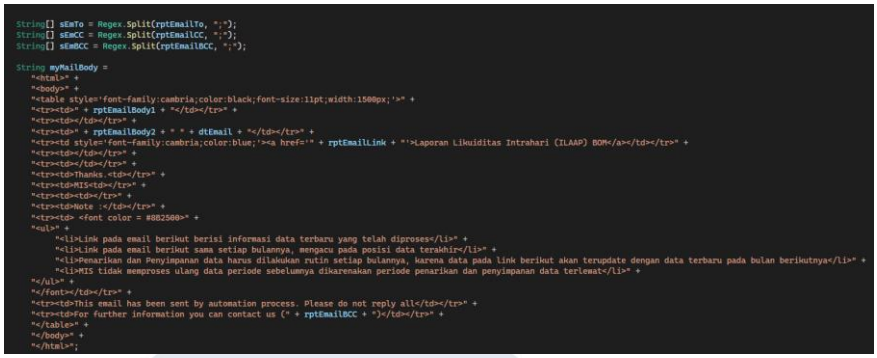
Gambar 3.43 menunjukkan SQL *query* pada SQL *task* email, dimana *query* ini mengambil data untuk pengiriman email laporan dari sebuah tabel mis_*report*s. Lalu, terdapat filter menggunakan *where* yang berfungsi untuk memastikan hanya data untuk jenis laporan tertentu yang diambil. Hasil dari *query* ini digunakan sebagai *input* sebelum *foreach loop container*, sehingga dapat melakukan *looping* untuk mengirimkan email berdasarkan setiap data atau *value* yang diambil dari *sql task* email.



```
public void Main()
{
    //set variables for email
    String rptEmailSender = Dts.Variables["rptEmailSender"].Value.ToString();
    String rptEmailSubject = Dts.Variables["rptEmailSubject"].Value.ToString();
    String rptEmailTo = Dts.Variables["rptEmailTo"].Value.ToString();
    String rptEmailCC = Dts.Variables["rptEmailCC"].Value.ToString();
    String rptEmailBCC = Dts.Variables["rptEmailBCC"].Value.ToString();
    String rptEmailBody1 = Dts.Variables["rptEmailBody1"].Value.ToString();
    String rptEmailBody2 = Dts.Variables["rptEmailBody2"].Value.ToString();
    String rptEmailLink = Dts.Variables["rptEmailLink"].Value.ToString();
    String dtEmail = Dts.Variables["dtEmail"].Value.ToString();

    //set variables for reports
    String rptPeriode = Dts.Variables["rptPeriode"].Value.ToString();
    String rptExt = Dts.Variables["rptExt"].Value.ToString().ToLower();
    String rptName = Dts.Variables["rptName"].Value.ToString();



    //set variables for folder
    String SmtpServer = Dts.Connections["SMTP Connection Manager"].Properties["SmtpServer"].GetValue(Dts.Connections["SMTP Connection Manager"]).ToString();

    MailMessage myHtmlFormattedMail = new MailMessage();
    myHtmlFormattedMail.IsBodyHtml = true;
    myHtmlFormattedMail.Subject = rptEmailSubject + " " + dtEmail;
    myHtmlFormattedMail.From = new MailAddress(rptEmailSender);


    String[] sEmTo = Regex.Split(rptEmailTo, ";");
    String[] sEmCC = Regex.Split(rptEmailCC, ";");
    String[] sEmBCC = Regex.Split(rptEmailBCC, ";");
```

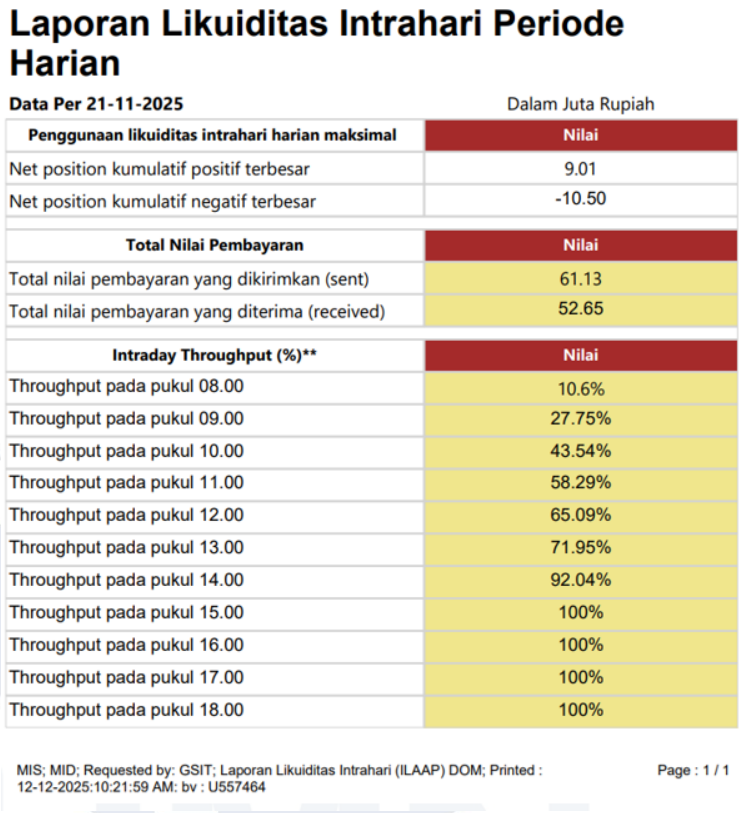
Gambar 3.44 Kode *Script Task* Untuk Inisialisasi Parameter Pengiriman Email.

Gambar 3.44 menunjukkan kode untuk inisialisasi parameter email pada *script task* di SSIS, dimana terdapat variabel yang telah didefinisikan untuk pengiriman email dan *report*. Lalu, menggunakan konfigurasi SMTP dari *SMTP Connection Manager* agar proses pengiriman mengikuti konfigurasi koneksi yang telah disiapkan di SSIS. Selanjutnya, *script* membentuk sebuah objek MailMessage dengan format HTML dan menyusun *subject* dengan menambahkan tanggal, menetapkan alamat pengirim, lalu memecah daftar email To/CC/BCC menggunakan fungsi Regex.Split, sehingga dalam satu variabel dapat berisi banyak alamat email yang dipisahkan tanda titik koma.



Gambar 3.45 Kode *Script Task* untuk Penyusunan *Body* Email HTML dan Memasukkan *Link* Laporan.

Gambar 3.45 menunjukkan kode untuk menyusun *body* email dengan format HTML dan memasukkan link laporan yang akan dikirimkan. Terdapat variabel myMailBody yang berisi *tag* HTML. Lalu, terdapat *hyperlink* yang mengarahkan ke rptemaillink, dimana berisi laporan likuiditas intrahari. Kode ini memungkinkan email yang dikirimkan secara dinamis dengan format yang telah ditentukan. Berdasarkan hasil dari pembuatan *ETL flow* menggunakan SSIS pada proyek ini, terdapat perubahan terkait data laporan likuiditas intrahari, dimana sebelum diproses, daftar penamaan dan format data seperti nilai *net position, income, outcome,* dan *throughput* masih sebelum disesuaikan. Setelah diproses, format data telah berubah sesuai dengan kebutuhan *user*.



Laporan Likuiditas Intrahari Periode Harian

Data Per 21-11-2025 Dalam Juta Rupiah

Penggunaan likuiditas intrahari harian maksimal	Nilai
Net position kumulatif positif terbesar	9.01
Net position kumulatif negatif terbesar	-10.50

Total Nilai Pembayaran	Nilai
Total nilai pembayaran yang dikirimkan (sent)	61.13
Total nilai pembayaran yang diterima (received)	52.65

Intraday Throughput (%)**	Nilai
Throughput pada pukul 08.00	10.6%
Throughput pada pukul 09.00	27.75%
Throughput pada pukul 10.00	43.54%
Throughput pada pukul 11.00	58.29%
Throughput pada pukul 12.00	65.09%
Throughput pada pukul 13.00	71.95%
Throughput pada pukul 14.00	92.04%
Throughput pada pukul 15.00	100%
Throughput pada pukul 16.00	100%
Throughput pada pukul 17.00	100%
Throughput pada pukul 18.00	100%

MIS; MID; Requested by: GSIT; Laporan Likuiditas Intrahari (ILAAP) DOM; Printed : 12-12-2025:10:21:59 AM: bv : U557464 Page : 1 / 1

Gambar 3.46 Tampilan Laporan Likuiditas Intrahari.

Gambar 3.46 menunjukkan tampilan laporan likuiditas intrahari yang dibangun menggunakan SSRS. Laporan ini juga menggunakan filter terkait tanggal untuk menampilkan data likuiditas. Selain itu, terdapat penambahan periode data menggunakan *expression*, *watermark*, *header*, dan *footer* terkait informasi laporan likuiditas intahari.

Tabel 3.3.13 Tabel UAT Laporan Likuiditas Intrahari.

Scenario	Input	Expected Output	Result
Normal	*Run package* dan *report*.	Tabel agunan terbentuk dan *report* pinjaman tampil	Berhasil membuat tabel dan menampilkan *report* pinjaman
Data periode kosong	*Run report*.	Tidak terdapat data pada periode ini.	Muncul pesan "Tidak terdapat

			data pada periode ini”

Tabel 3.3.13 menunjukkan hasil UAT untuk Laporan Pinjaman dengan dua skenario utama, yaitu skenario normal dan skenario periode data kosong. Pada skenario normal, paket SSIS dan *report* dijalankan dan diharapkan tabel pinjaman terbentuk serta *report* pinjaman dapat ditampilkan, dengan hasil uji berhasil membuat tabel dan menampilkan *report*. Pada skenario periode kosong, *report* dijalankan pada rentang waktu tanpa data dan sistem diharapkan menampilkan informasi bahwa tidak terdapat data pada periode tersebut, dengan hasil uji berupa munculnya pesan “Tidak terdapat data pada periode ini”.

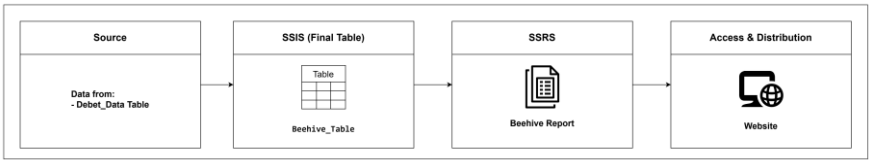
Tabel 3.3.14 Estimasi Waktu Proses dan Frekuensi Eksekusi Laporan Likuiditas Intrahari.

Laporan	Estimasi Waktu Run	Frekuensi
Laporan Likuiditas Intrahari	4-5 Menit (Run SSIS dan render SSRS)	Harian

Tabel 3.3.14 menunjukkan estimasi waktu yang dibutuhkan untuk menjalankan proses laporan likuiditas intrahari (SSIS hingga render SSRS) serta frekuensi pelaksanaannya, sehingga memudahkan pemantauan kebutuhan operasional dan jadwal distribusi laporan.

3.3.1.18 Pembuatan Website Proyek Laporan Beehive

Laporan Beehive adalah laporan transaksi yang diakses melalui SSRS berdasarkan parameter yang diinput user melalui sebuah *website* yang dibuat menggunakan *framework* ASP.NET yang dibuat mengumpulkan *filter*, memvalidasi input, lalu membentuk URL SSRS secara konsisten sehingga akses laporan menjadi lebih mudah.



Gambar 3.47 Alur Arsitektur Proyek Laporan Beehive.

Gambar 3.47 menunjukkan arsitektur proyek Laporan Beehive dimulai dari sumber data Debet_Data sebagai data transaksi, kemudian SSIS membentuk tabel final Beehive_Table agar struktur data konsisten dan siap digunakan untuk pelaporan. SSRS menggunakan tabel tersebut untuk menghasilkan Beehive *Report*. Akses laporan dilakukan melalui website internal yang menerima input filter dari pengguna dan membentuk *link* atau URL SSRS sehingga *report* dapat dibuka langsung sesuai parameter yang dipilih.

```
[HttpPost]
0 references
public ActionResult ViewReport(ReportViewModel filter)
{
    InitDropdown(filter);
    ValidateDates(filter);

    var query = BeehiveDummy.getAllTransaction().AsQueryable();

    if (filter.TanggalAwal.HasValue)
        query = query.Where(x => x.TanggalTransaksi >= filter.TanggalAwal);

    if (filter.TanggalAkhir.HasValue)
        query = query.Where(x => x.TanggalTransaksi <= filter.TanggalAkhir);

    if (!string.IsNullOrEmpty(filter.JenisTransaksi))
        query = query.Where(x => x.JenisTransaksi == filter.JenisTransaksi);

    if (!string.IsNullOrEmpty(filter.KodePerusahaan))
        query = query.Where(x => x.KodePerusahaan == filter.KodePerusahaan);

    if (!string.IsNullOrEmpty(filter.NamaFile))
        query = query.Where(x => x.NamaFile == filter.NamaFile);

    filter.Result = query.ToList();



    string tglAwal = filter.TanggalAwal.HasValue ? filter.TanggalAwal.Value.ToString("yyyy-MM-dd") : "";
    string tglAkhir = filter.TanggalAkhir.HasValue ? filter.TanggalAkhir.Value.ToString("yyyy-MM-dd") : "";
    string jenis = filter.JenisTransaksi ?? "";
    string kodePerusahaan = filter.KodePerusahaan ?? "";
    string namaFile = filter.NamaFile ?? "";
```

Gambar 3.48 Kode *Action* View*Report* untuk Pemrosesan Filter *Report.*

Gambar 3.48 menunjukkan *action* View*Report* dengan atribut [HttpPost] dan menerima objek *Report*ViewModel sebagai parameter filter. Awalnya, memanggil fungsi untuk pilihan *dropdown* dan memvalidasi rentang tanggal yang diinput pengguna. Selanjutnya, *controller* mengambil seluruh data transaksi dan menerapkan beberapa kondisi *where* secara bertahap berdasarkan nilai yang diisi

pada *form*, yaitu tanggal awal, tanggal akhir, jenis transaksi, kode perusahaan, dan nama *file*, dimana hanya parameter yang terisi saja yang digunakan sebagai filter, sehingga pencarian menjadi dinamis sesuai *input* pengguna. Hasil akhir *query* kemudian dikonversi menjadi *list* dan disimpan ke dalam filter.Result untuk ditampilkan kembali pada halaman web. Setelah proses pemfilteran, nilai – nilai parameter tersebut juga diubah ke bentuk *string* dan disimpan dalam variabel parameter (tglAwal, tglAkhir, jenis, kodePerusahaan, dan namaFile) yang nantinya akan digunakan sebagai parameter dalam URL *report* SSRS.

```
    string reportUrl = string.Format(
        ConfigurationManager.AppSettings["ReportBeehive"],
        tglAwal,
        tglAkhir,
        jenis,
        kodePerusahaan,
        namaFile
    );

    filter.RptUrl = reportUrl;

    return View(filter);
}


2 references
private void InitDropdown(ReportViewModel filter)
{
    filter.ListJenisTransaksi = new List<SelectListItem>
{
    new SelectListItem { Text = "ALL", Value = "" },
    new SelectListItem { Text = "Data Transaksi Debet", Value = "TRXDB" },
    new SelectListItem { Text = "Data Transaksi Kredit (PBK/CPU/PBY) Normal", Value = "TRXKN" },
    new SelectListItem { Text = "Data Transaksi Kredit KU Normal", Value = "TRXKUN" },
    new SelectListItem { Text = "Data Tranasksi Kredit (PBK) Oneflow", Value = "TRXKO" },
    new SelectListItem { Text = "Data Transaksi Kredit KU Oneflow", Value = "TRXKUO" },
};
}
```

Gambar 3.49 Pembentukan URL *Report* SSRS dan Inisialisasi *Dropdown* Jenis Transaksi.

Gambar 3.49 menunjukkan kode yang membentuk URL *report* untuk SSRS. URL tersebut dibangun menggunakan string.Format dengan *template* yang diambil dari konfigurasi aplikasi, kemudian diisi dengan parameter yang telah disiapkan sebelumnya, yaitu tglAwal, tglAkhir, jenis, kodePerusahaan, dan namaFile. Hasil format URL disimpan di properti filter.RptUrl. Selanjtunya, terdapat *method* InitDropdown, yang digunakan untuk menginisialisasi menampilkan *list* jenis dalam bentuk *dropdown*. Daftar ini berisi

pilihan jenis data transaksi dengan pasangan nilai kode transaksi yang akan dikirim ke *backend*.

```
1 reference
private void ValidateDates(ReportViewModel filter)
{
    if (filter.TanggalAwal.HasValue)
    {
        var minDate = DateTime.Today.AddYears(-5);
        if (filter.TanggalAwal < minDate)
        {
            filter.TanggalAwal = minDate;
            ViewBag.Message = "Tanggal awal maksimal 5 tahun ke belakang.";
        }
    }

    if (filter.TanggalAwal.HasValue && filter.TanggalAkhir.HasValue)
    {
        var maxRange = filter.TanggalAwal.Value.AddYears(1);
        if (filter.TanggalAkhir > maxRange)
        {
            filter.TanggalAkhir = maxRange;
            ViewBag.Message = "Range tanggal maksimal 1 tahun.";
        }
    }
}
```

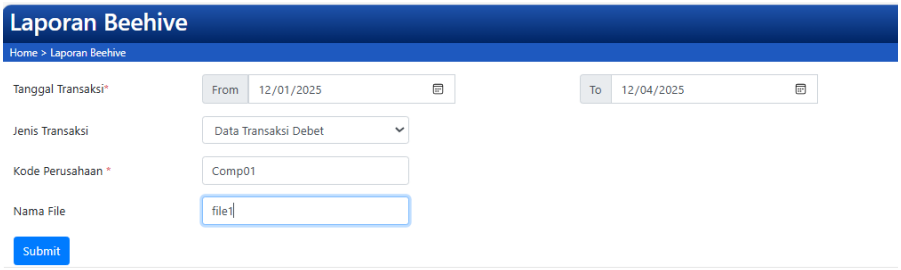
Gambar 3.50 Kode Fungsi Validasi Rentang Tanggal *Filter Report.*

Gambar 3.50 menunjukkan method ValidateDates yang yang berfungsi untuk melakukan validasi terhadap input tanggal pada *Report*ViewModel, dimana pada tanggal awal akan dibatasi agar tidak lebih lama dari lima tahun ke belakang dari tanggal hari ini. Lalu, antara tanggal awal dan akhir akan menghitung batas maksimum rentang satu tahun dari tanggal awal. Jika tanggal akhir melebihi batas ini, nilai tanggal akhir akan disesuaikan ke tanggal maksimum yang diperbolehkan dan pesan peringatan lain ditampilkan.

```
@model ReportViewModel
<div class="container-fluid pt-3">
    @using (Html.BeginForm("viewreport", "Home", FormMethod.Post, new { target = "_blank" }))
    {
        <div class="form-group row">
            <label class="col-lg-1 col-form-label">Tanggal Transaksi<span class="text-danger">*</span></label>
            <div class="col-lg-2">
                <div class="input-group">
                    <div class="input-group-prepend">
                        <span class="input-group-text">From</span>
                    </div>
                    @Html.TextBoxFor(m => m.TanggalAwal, "{0:yyyy-MM-dd}", new { @class = "form-control", type = "date", id = "txtTanggalAwal", required = "required" })
                    @Html.ValidationMessageFor(m => m.TanggalAwal, "", new { @class = "text-danger" })
                </div>
            </div>
            <div class="col-lg-2">
                <div class="input-group">
                    <div class="input-group-prepend">
                        <span class="input-group-text">To</span>
                    </div>
                    @Html.TextBoxFor(m => m.TanggalAkhir, "{0:yyyy-MM-dd}", new { @class = "form-control", type = "date", id = "txtTanggalAkhir", required = "required" })
                    @Html.ValidationMessageFor(m => m.TanggalAkhir, "", new { @class = "text-danger" })
                </div>
            </div>
        </div>
        <div class="form-group row">
            <label for="SelectedCabang" class="col-lg-1 col-form-label">Jenis Transaksi</label>
            <div class="col-lg-4">
                @Html.DropDownListFor(m => m.JenisTransaksi, Model.ListJenisTransaksi, new { @class = "form-control", id = "ddlCabang" })
            </div>
        </div>
```

Gambar 3.51 Kode Pembuatan *Form* Input Parameter Laporan Beehive.

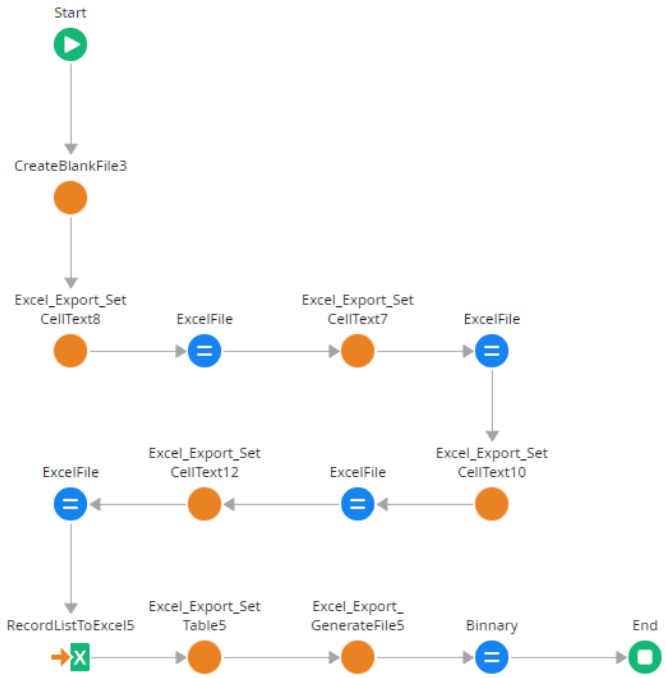
Gambar 3.51 merupakan kode bagian *view* yang digunakan untuk membangun *form input* pada *website* beehive. Form dibuat menggunakan metode POST yang berarti saat tombol submit ditekan, data form akan dikirim sebagai data *request* di *body* ke *action* View*report* pada HomeController, dan hasilnya dibuka di tab baru. Input pada form dibuat dengan komponen *framework* yang langsung terhubung ke data model, sehingga ketika pengguna mengisi *field* di *input form*, maka nilainya otomatis masuk ke variabel yang sesuai. Pada *field* tanggal, input diatur sebagai tipe *date* supaya muncul *date picker*. Selain itu, ada komponen pesan validasi yang akan menampilkan peringatan kalau input belum diisi atau tidak sesuai syaratnya. Berdasarkan pengerjaan proyek ini, terlihat bahwa, sebelum diproses, pengguna tidak bisa menentukan parameter pencarian secara dinamis untuk menemukan data yang dibutuhkan. Setelah diproses, website ASP.NET menerapkan filter dinamis dan membentuk URL SSRS otomatis sehingga *report* bisa dibuka langsung sesuai parameter.



Gambar 3.52 Tampilan *Website* Laporan Beehive.

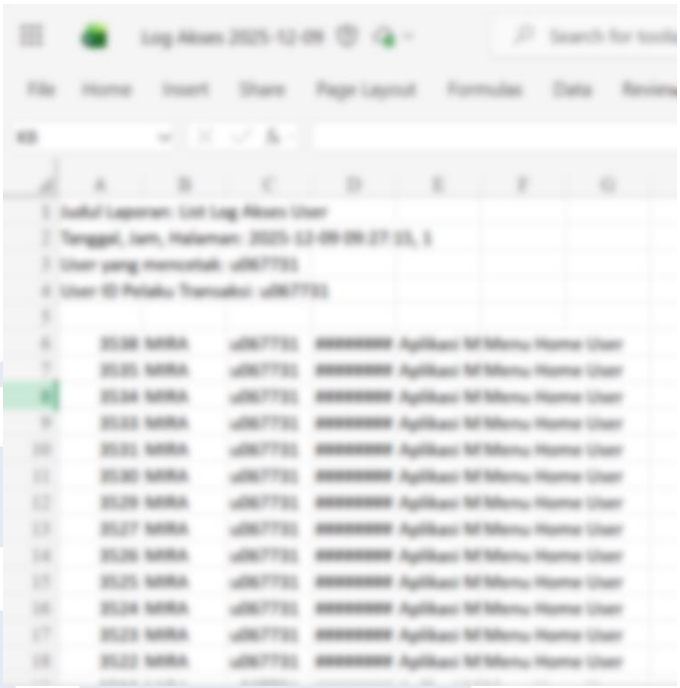
Gambar 3.52 merupakan tampilan *website* laporan beehive yang berfungsi untuk mengisi parameter yang akan ditujukan ke sebuah laporan SSRS laporan beehive, sehingga data yang ditampilkan sesuai dengan data yang diisi di *input form*.

3.3.1.19 Menambahkan Header Pada Excel Hasil Download Pada Aplikasi MIRA



Gambar 3.53 *Flow* Penambahan *Header* Pada File Excel.

Gambar 3.53 menunjukkan *flow* untuk menghasilkan *header* pada *file* excel terkait *log* akses *user* yang diunduh, dimana *flow* dimulai dari inisialisasi proses dan pembuatan file kosong, lalu terdapat fungsi untuk membuat file kosong. Selanjutnya, terdapat pengisian *value* pada *cell* yang akan dijadikan header seperti judul laporan, tanggal/jam, halaman, dan informasi user. Setelah itu, *flow* melanjutkan ke proses pengisian data utama ke dalam *sheet*, kemudian file Excel di-*generate* menjadi file akhir dan dikonversi ke format binary agar dapat di-download oleh *user*.



Gambar 3.54 Tampilan File Excel Dengan *Header.*

Gambar 3.54 menunjukkan tampilan hasil *file* excel yang telah diunduh, dimana terdapat *header*, lalu dibawahnya terdapat data terkait *log* akses *user*.

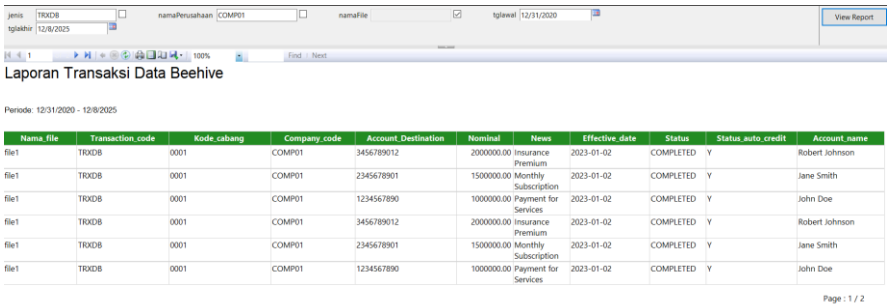
3.3.1.20 Membuat Laporan Beehive Menggunakan SSRS dan UAT

Query:

```
SELECT
  NAMA_FILE,  TRANSACTION_CODE,   KODE_CABANG,
  COMPANY_CODE,   ACCOUNT_DESTINATION,   NOMINAL,
  NEWS,  EFFECTIVE_DATE,  STATUS,
  STATUS_AUTO_CREDIT,  ACCOUNT_NAME
FROM MIS.COMBINED_DEBET_DATA
WHERE
  (EFFECTIVE_DATE >= ? AND EFFECTIVE_DATE <= ?)
  AND (? IS NULL OR TRIM(?) = '' OR UPPER(TRIM(TRANSACTION_CODE)) = UPPER
(TRIM(?)))
  AND (? IS NULL OR TRIM(?) = '' OR UPPER(TRIM(COMPANY_CODE)) = UPPER
(TRIM(?)))
  AND (? IS NULL OR TRIM(?) = '' OR UPPER(TRIM(NAMA_FILE)) = UPPER(TRIM(?)))
```

Gambar 3.55 SQL *Query* Pengambilan Data Laporan Beehive.

Gambar 3.55 menunjukkan SQL *query* untuk mengambil data dari tabel mis.combined_debet_data, dimana menggunakan filter *where* sesuai dengan parameter yang dimasukkan. Nilai dari parameter ini dimasukkan dari hasil *input* di *website* beehive.



Gambar 3.56 Tampilan Laporan Beehive Pada SSRS.

Gambar 3.56 menunjukkan tampilan laporan beehive pada SSRS, dimana terdapat beberapa kolom yang berisi data beehive pada tabel tersebut. Lalu terdapat periode data yang ditampilkan menggunakan *expression* dengan data dari *value* parameter yang dimasukkan.

Tabel 3.3.15 Tabel UAT Laporan Beehive.

Scenario	Input	Expected Output	Result
Normal	Input filter di website.	Sistem membentuk URL *report* dan menampilkan *report* via RptUrl.	Berhasil menampilkan *report* Beehive.
Tanggal invalid	Tanggal awal > akhir (01-01-2023 – 01-01-2022)	Menampilkan pesan error.	Muncul pesan “Tanggal awal tidak boleh lebih besar dari tanggal akhir”.
Tanggal invalid	Rentang tanggal terlalu lebar (01-01-2023 – 01-01-2025)	Menampilkan pesan error.	Muncul pesan “Rentang tanggal maksimal 1 tahun”.

Tanggal invalid	Tanggal awal melewati batas historir (01-01-2018 – 01-01-2019)	Menampilkan pesan error.	Muncul pesan "Tanggal awal maksimal 5 tahun ke belakang dari hari ini".

Tabel 3.3.15 menunjukkan menunjukkan hasil UAT untuk *website* Beehive dalam membentuk *link* SSRS berdasarkan filter yang diinput pengguna. Pada skenario normal, pengguna mengisi filter di *website* dan sistem berhasil membentuk URL report serta menampilkan *report* Beehive melalui RptUrl. Selain itu, tabel juga menguji validasi input tanggal, dimana ketika tanggal awal lebih besar dari tanggal akhir, rentang tanggal melebihi batas maksimal satu tahun, dan tanggal awal melewati batas histori. Pada masing – masing kondisi tersebut sistem menolak proses dan menampilkan pesan *error* yang sesuai ("Tanggal awal tidak boleh lebih besar dari tanggal akhir", "Rentang tanggal maksimal 1 tahun", dan "Tanggal awal maksimal 5 tahun").

Tabel 3.3.16 Estimasi Waktu Proses dan Frekuensi Eksekusi Laporan Beehive.

Laporan	Estimasi Waktu Run	Frekuensi
Laporan Beehive	4-5 Menit (Run SSIS dan render SSRS)	Harian

Tabel 3.3.16 menunjukkan estimasi waktu yang dibutuhkan untuk menjalankan proses laporan Beehive (SSIS hingga render SSRS) serta frekuensi pelaksanaannya, sehingga memudahkan pemantauan kebutuhan operasional dan jadwal distribusi laporan.

3.3.2 Kendala yang Ditemukan

Kendala yang ditemukan selama periode kerja magang di perusahaan Bank Central Asia adalah sebagai berikut.

1. Tidak memperoleh pelatihan khusus terkait cara teknis penggunaan beberapa alat, seperti *OutSystem*. Mahasiswa belum

memiliki pengetahuan dan pengalaman dalam penggunaan alat tersebut.

2. Terdapat keterbatasan dan ketidaklengkapan data pada *environment development*. Beberapa skenario uji tidak dapat dilakukan secara optimal karena *data development* lebih sedikit dari *data production*, bahkan untuk beberapa kebutuhan tidak tersedia *data development* sama sekali.

3.3.3 Solusi atas Kendala yang Ditemukan

Solusi atas kendala yang ditemukan selama proses kerja magang pada perusahaan Bank Central Asia adalah sebagai berikut.

1. Dalam mengatasi keterbatasan pemahaman terkait *OutSystems*, mahasiswa melakukan eksplorasi mandiri melalui referensi internal, sumber eksternal, dokumentasi, dan praktik langsung pada modul yang dikerjakan. Selain itu, mahasiswa juga berdiskusi dengan pembimbing dan rekan kerja ketika menemukan kendala teknis yang spesifik.
2. Dalam mengatasi keterbatasan data pada *environment develnakopment*, mahasiswa menyusun dan menggunakan data *dummy* yang disesuaikan dengan struktur dan kebutuhan pengujian, sehingga proses pengembangan dan pengujian tetap dapat berjalan tanpa mengganggu data operasional pada *environment production*.