GenC Bank Management System

Project Name: Bank Management

Problem Statement:

The Bank Management System Application is designed to efficiently manage banking services and handle day-to-day transactions. It ensures that client details and their accounts are securely managed within the Salesforce org. The application includes essential functionalities such as account management, transaction processing, and customer service. It also supports various business use cases, including loan processing, account opening, and customer relationship management. By integrating these features, the application provides a comprehensive solution for banks to streamline their operations and enhance customer satisfaction. This robust system ensures that all banking activities are conducted smoothly and efficiently, making it an invaluable tool for modern banking institutions.

Use Case Stories (ideation)

Account Opening

An employee of a bank will be creating a bank account for a person using interactive screen flow.

Transaction History

An account holder can see his transaction history using a visually appealing dashboards and reports

Customer Service

An account holder can log a case issue through an interactive screen or can also raise a case through email service.

Loan Processing

An account holder will be able to apply for loans and the bank employees need to have an approval system to approve these loan applications.An account holder will also be able to pay his monthly loan amount through a one click pay button screen.

Transfer Amount

An account holder should be able to transfer amounts from one account to another account (currently the same bank).

Referrals

An account holder should be able to refer other people to create accounts in this bank or take a loan from the bank.

Flows

- A Record trigger flow to trigger an email when an account is created or the detail in the account is updated. (creation can be done through a screen flow).
- A Screen Flow to freeze or unfreeze the account.
- A flow and approval process for kyc(Know your customer) verification.
- A screen flow to transfer amount from one account to the other.

Use Cases Implemented

- ❖ **Account Management:**
  - ➢ Sending the account creation email and submitting it for KYC approval.
  - ➢ PAN Card is being uploaded while creating the account.
  - ➢ Freezing or unfreezing the account, sending an email about account status.
  - ➢ Creation of an account by checking the PAN duplication.
  - ➢ Checkbox for current address to be same as permanent address.
  - ➢ Email sent to KYC approver to approve the KYC when an account is created.
- ❖ **Transaction Management:**
  - ➢ Record all banking transactions including transfers, withdrawals, and deposits.
  - ➢ Account balance update on withdrawal, transfer, deposit.

- ➢ Checking the insufficient balance and freeze status before proceeding with the transaction.
- ➢ Sending an email to the customer for every transaction.
- ➢ Initial deposit is greater than 2000.

❖ **Experience Cloud:**

- ➢ Provide a portal for customers to access their account details and perform transactions.

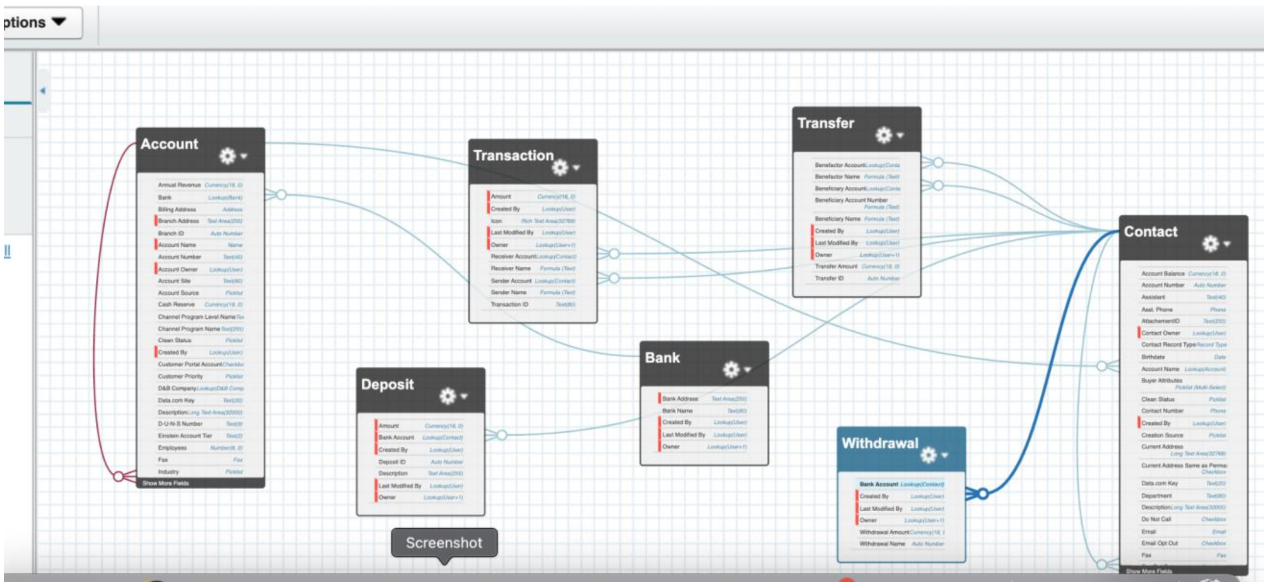
❖ **Reports and Dashboards:**

- ➢ Track daily transfers, withdrawals, and deposits.
- ➢ To display account type (Saving account, Current account).
- ➢ Account filtered by branch per month.

❖ **Custom App:**

- ➢ **App Name: GenC Bank.**
- ➢ Provide a portal for employees to handle their banking services.

Data Modeling



Objects

Bank Account (Std Contact)

Fields:

Field Label	**Field Name**	**Data Type**
Account Balance	Account_Balance__c	Currency
Account Number	Account_Number__c	Auto Number
AttachmentID	AttachementID__c	Text
Contact Number	Contact_Number__c	Phone
Current Address	Current_Address__c	Long Text Area
Current Address Same as Permanent	Current_Address_Same_as_Permanent	Checkbox
Freeze	Freeze__c	Checkbox
Initial Deposit	Initial_Deposit__c	Currency
KYC	KYC__c	Checkbox
Name	Name	Name
Pan Card Image	Pan_Card_Image__c	Formula (Text)
Pan Card Number	Pan_Card_Number	Text
Permanent Address	Permanent_Address__c	Long Text Area

Validation Rules:

- ❖ **Pan Card Image:**
  - ➢ **Field:** Formula (Text)
  - ➢ **Description:** This formula field generates a hyperlink to view the PAN card image associated with a specific record. The hyperlink is constructed using the Salesforce ContentDocument URL format and dynamically includes the Attachment ID and PAN Card Number.
  - ➢ **Formula:** HYPERLINK ('/lightning/r/ContentDocument/' & AttachementID__c & '/view', PAN_Card_Number__c)
  - ➢ **Usage:** When this field is displayed on a record, it provides a clickable link labeled with the PAN card number. Clicking the link opens the PAN card image in Salesforce Lightning, allowing users to view the document directly.
- ❖ **Account Balance:**

- **Rule Name:** Initial_deposit_cannot_be_less_than_2000
- **Description:** Ensures that the initial balance entered for an account is not less than 2000. This rule helps maintain a minimum balance requirement for all accounts.
- **Formula:** Initial_Deposit__c < 2000
- **Error Message:** Initial deposit can't be less than 2000
- **Error Location:** Top of the page.

❖ Pan Card:

- **Rule Name:** Pan_Card
- **Description:** Ensures that the PAN Card field is not left blank. This rule is necessary to comply with regulatory requirements and to ensure that all accounts have a valid PAN Card number.
- **Formula:** ISBLANK(PAN_Card_Number__c)
- **Error Message:** The PAN Card field is required and cannot be left blank.
- **Error Location:** Next to the PAN Card Number field.

❖ Pan Card Number:

- **Rule Name:** pan_card_number
- **Description:** Ensures that the PAN Card number entered follows the correct format. This rule helps maintain data integrity and compliance with regulatory standards.
- **Formula:** NOT (REGEX(PAN_Card_Number__c , "^[A-Z]{5} [0-9]{4} [A-Z]{1} $"))
- **Error Message:** The PAN Card number must be in the format: 5 letters, 4 digits, and 1 letter (e.g., ABCDE1234F).
- **Error Location:** Next to the PAN Card Number field.

❖ Contact Number:

- **Rule Name:** Phone_No
- **Description:** Ensures that the phone number must have only 10 digits.
- **Formula:** Len(Contact_Number__c) <> 10
- **Error Message:** Phone number must have 10 digits

- **Error Location:** Next to the Contact Number Field.

- **Initial Deposit:**
  - **Rule Name:** Prevent_Initial_Deposit_Edit
  - **Description:** Ensures that the Initial_Deposit__c field cannot be changed once the record is created. This rule helps maintain the integrity of the initial deposit amount.
  - **Formula:** ISCHANGED(Initial_Deposit__c) && NOT (ISNEW ())
  - **Error Message:** Once created it cannot be edited.
  - **Error Location:** Next to the Initial Deposit field.

Bank (Custom)

Fields:

Field Label	Field Name	Data Type
Bank Address	Bank_Address__c	Text Area
Bank Name	Name	Text

Branch (Std Account)

Fields:

Field Label	Field Name	Data Type
Bank	Bank__c	Lookup (Bank)
Branch Address	Branch_Address__c	Text Area
Branch ID	Branch_ID__c	Auto Number
Branch Name	Name	Name
Cash Reserve	Cash_Reserve__c	Currency

Deposit (Custom)

Fields:

Field Label	Field Name	Data Type
Amount	Amount__c	Currency
Bank Account	Bank_Account__c	Lookup (Bank Account)
Deposit ID	Name	Auto Number
Description	Description__c	Text Area

Validation Rules:

- ❖ **Amount:**
  - ➢ **Rule Name:** Amount
  - ➢ **Description:** Ensures that the Amount__c field is greater than zero. This rule helps maintain data integrity by preventing negative or zero amounts.
  - ➢ **Formula:** Amount__c <= 0
  - ➢ **Error Message:** Amount Cannot be Negative
  - ➢ **Error Location:** Next to the Amount field.

Transfer (Custom)

Fields:

Field Label	Field Name	Data Type
Benefactor Account	Benefactor_Account__c	Lookup (Bank Account)
Benefactor Name	Benefactor_Name__c	Formula (Text)
Beneficiary Account	Beneficiary_Account__c	Lookup (Bank Account)
Beneficiary Name	Beneficiary_Name)__c	Formula (Text)
Beneficiary Account Number	Beneficiary_Account_Number__c	Formula (Text)
Transfer Amount	Transfer_Amount__c	Currency
Transfer ID	Name	Auto Number

Validation Rules:

- ❖ **Benefactor Name:**
  - ➢ **Rule Name:** Benefactor Name
  - ➢ **Description:** This field concatenates the first name and last name of the benefactor associated with the account. It is used to display the full name of the benefactor in a single field.
  - ➢ **Formula:** Benefactor_Account__r. FirstName& Benefactor_Account__r. LastName
  - ➢ **Usage:** This field is useful for generating reports, displaying benefactor names in user interfaces, and ensuring consistent naming conventions

across records. It simplifies the process of referencing the benefactor's full name without needing to manually combine first and last name fields.

❖ Beneficiary Account Number:

- **Rule Name:** Beneficiary Account Number
- **Description:** This field is a formula field that concatenates the first name and last name of the beneficiary associated with the account. It is used to display the full name of the beneficiary in a single field.
- **Formula:** Beneficiary_Account__r. FirstName & Beneficiary_Account__r. LastName
- **Usage:** This field is useful for generating reports, displaying beneficiary names in user interfaces, and ensuring consistent naming conventions across records. It simplifies the process of referencing the beneficiary's full name without needing to manually combine first and last name fields.

❖ Beneficiary Name:

- **Rule Name:** Beneficiary Name
- **Description:** This field is a formula field that concatenates the first name and last name of the beneficiary associated with the account. It is used to display the full name of the beneficiary in a single field.
- **Formula:** Beneficiary_Account__r.FirstName & Beneficiary_Account__r.LastName
- **Usage:** This field is used to generate comprehensive reports that include the full name of the beneficiary, display the beneficiary's full name in user interfaces for better readability, and ensure consistent naming conventions across records. It simplifies the process by automatically combining the first and last name fields into a single, easily accessible field.

Transaction (Custom)

Fields:

Field Label	Field Name	Data Type
Amount	Amount__c	Currency
Icon	Icon__c	Rich Area Text
Receiver Account	Receiever_Account__c	Lookup (Bank Account)
Receiver Name	Receiver_Name__c	Formula (Text)
Sender Account	Sender_Account__c	Lookup (Bank Account)
Sender Name	Sender_Name__c	Formula (Text)
Transaction ID	Name	Text

Validation Rules:

❖ **Receiver Name:**

- **Rule Name:** Receiver Name
- **Description:** This field is a formula field that concatenates the first name and last name of the receiver associated with the account. It is used to display the full name of the receiver in a single field.
- **Formula:** Receiver_Account__r.FirstName & Receiver_Account__r.LastName
- **Usage:** This field is used to generate comprehensive reports that include the full name of the receiver, display the receiver's full name in user interfaces for better readability, and ensure consistent naming conventions across records. It simplifies the process by automatically combining the first and last name fields into a single, easily accessible field.

❖ **Sender Name:**

- **Rule Name:** Sender Name
- **Description:** This field is a formula field that concatenates the first name and last name of the sender associated with the account. It is used to display the full name of the sender in a single field.
- **Formula:** Sender_Account__r.FirstName & Sender_Account__r.LastName

- **Usage:** This field is used to generate comprehensive reports that include the full name of the sender, display the sender's full name in user interfaces for better readability, and ensure consistent naming conventions across records. It simplifies the process by automatically combining the first and last name fields into a single, easily accessible field.

Withdrawal (Custom)

Fields:

Field Label	Field Name	Data Type
Bank Account	Bank_Account__c	Lookup (Bank Account)
Withdrawal Amount	Withdrawal_Amount__c	Currency
Withdrawal Name	Name	Auto Number

Validation Rules:

- **Withdrawal Amount:**
  - **Rule Name:** Account_Balance_should_be_more_than_Amt
  - **Description:** This validation rule ensures that the account balance is greater than the withdrawal amount. If the account balance is less than the withdrawal amount, an error message will be displayed, preventing the transaction.
  - **Formula:** Bank_Account__r.Account_Balance__c < Withdrawal_Amount__c
  - **Error Message:** Insufficent Balance in your account.
  - **Error Location:** Next to the Withdrawal Amount field.
- **Withdrawal Amount:**
  - **Rule Name:** Amount
  - **Description:** Ensures that the Withdrawal_Amount__c field is greater than zero. This rule helps maintain data integrity by preventing negative or zero withdrawal amounts.
  - **Formula:** Withdrawal_Amount__c <= 0
  - **Error Message:** The withdrawal amount must be greater than zero

- **Error Location:** Next to the Withdrawal Amount field.

- **Withdrawal Amount:**
  - **Rule Name:** Amount_Less_than_Cash_Reserve
  - **Description:** This validation rule ensures that the withdrawal amount does not exceed the cash reserve of the bank account. If the withdrawal amount is greater than the cash reserve, an error message will be displayed at the top of the page, preventing the transaction.
  - **Formula:** Bank_Account__r.Account.Cash_Reserve__c < Withdrawal_Amount__c
  - **Error Message:** Insufficent Balance in Bank Reserve
  - **Error Location:**Top of the page.

Page Layouts

- **Current Account Layout**
  - **Object:** Bank Account
  - **Fields:** Account Holder Name, Branch, PAN Card Number, Account Balance, Initial Deposit, Email, Phone, Permanent Address, Current Address.
  - **Related Lists:** Account History, Open Actives, Activity History, Transfers, Transfers (Beneficiary Account), Deposits, Withdrawals, Contacts, Approval History.
- **Savings Account**
  - **Object:** Bank Account
  - **Fields:** Account Holder Name, Branch, PAN Card Number, Account Balance, Initial Deposit, Email, Phone, Permanent Address, Current Address.
  - **Related Lists:** Account History, Open Activities, Activity History, Transfers, Transfers (Beneficiary Account), Deposits, Withdrawals, Contacts, Approval History.
- **Bank Layout**

- **Object:** Bank
  - **Fields:** Bank Name, Bank Address, Bank ID.
  - **Related Lists:** Open Activities, Activity History, Branches.
- **Branch Layout**
  - **Object:** Branch
  - **Fields:** Branch Name, Branch ID, Branch Manager, Cash Reserve.
  - **Related Lists:** Branch History, Open Activities, Activity History, Accounts.
- **Deposit Layout**
  - **Object:** Deposit
  - **Fields:** Account(ID), Amount, Deposit ID, Description.
  - **Related Lists:** Open Activities, Activity History.
- **Transfer Layout**
  - **Object:** Transfer
  - **Fields:** Beneficiary Account Number, Beneficiary Name, Transfer Amount, Benefactor Account, Beneficiary Account.
  - **Related Lists:** Open Activities, Activity History.
- **Withdrawal Layout**
  - **Object:** Withdrawal
  - **Fields:** Account(ID), Withdrawal Amount.
  - **Related Lists:** Open Activities, Activity History.

Record Types

- **Record Type Name: Savings Account**
  - **Object:** Bank Account
  - **Description:** Record type for Savings Accounts.
  - **Page Layout:** Savings Account Layout
- **Record Type Name: Current Account**
  - **Object:** Bank Account

- **Description:** Record type for Current Accounts.
- **Page Layout:** Current Account Layout

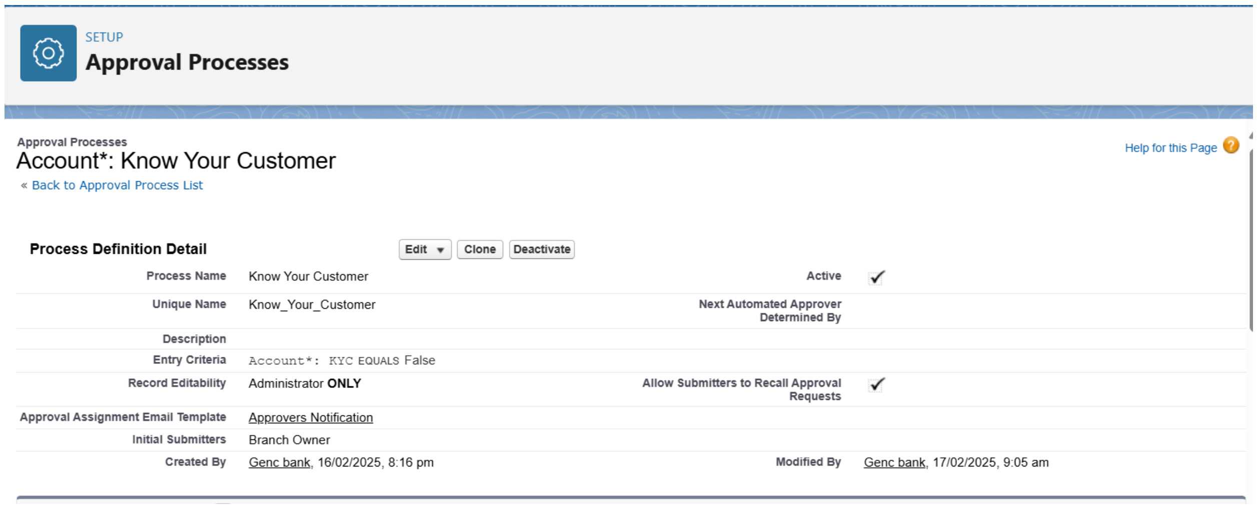
Security and Sharing Settings

- **Profile Name: System Administrator**
  - **Purpose:** The System Administrator profile is designed for users who need full access to the banking application to manage system settings, user permissions, and overall system maintenance. This profile ensures that administrators can perform all necessary tasks to keep the system running smoothly and securely.
- **Profile Name: KYC Manager**
  - **Purpose:** The KYC Manager profile is designed for users responsible for verifying and approving Know Your Customer (KYC) documents. This profile ensures that KYC Managers have the necessary access to review customer information and documents while maintaining data security and privacy.
- **Profile Name: Account Holder**
  - **Purpose:** The account holder profile in a banking application stores important details about customers who have accounts with the bank. This helps the bank manage customer relationships, follow regulations, and offer personalized services.
- **Profile Name: Genc Web Manager**
  - **Purpose:** This account holder in banking application is given to the person who is responsible for enabling the customer interface credentials.

Approval Process

- **Approval Process Name: Know Your Customer**

- **Purpose:** This approval process is designed to ensure that all Know Your Customer (KYC) documents submitted by customers are thoroughly reviewed and approved before the customer is allowed to open an account or access financial services. This process helps prevent fraud, money laundering, and other illegal financial activities.
- **API Version:** 63



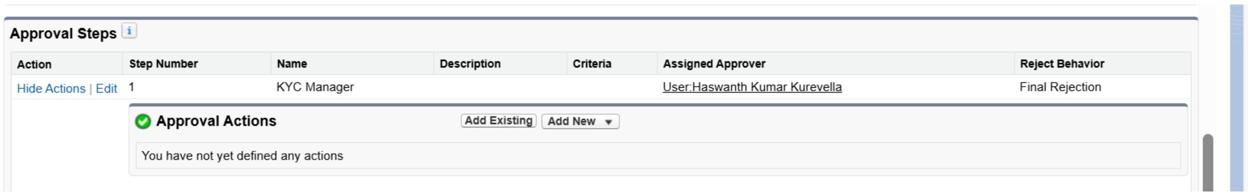
Approval Processes
Account*: Know Your Customer
« Back to Approval Process List

Process Definition Detail

Process Name	Know Your Customer	Active	✓
Unique Name	Know_Your_Customer	Next Automated Approver Determined By	
Description			
Entry Criteria	Account*: KYC EQUALS False		
Record Editability	Administrator ONLY	Allow Submitters to Recall Approval Requests	✓
Approval Assignment Email Template	Approvers Notification		
Initial Submitters	Branch Owner		
Created By	Genc bank, 16/02/2025, 8:16 pm	Modified By	Genc bank, 17/02/2025, 9:05 am

❖ Approval Process steps:

- **Approval Action:** An email notification is sent to the KYC approver to review and approve the KYC.

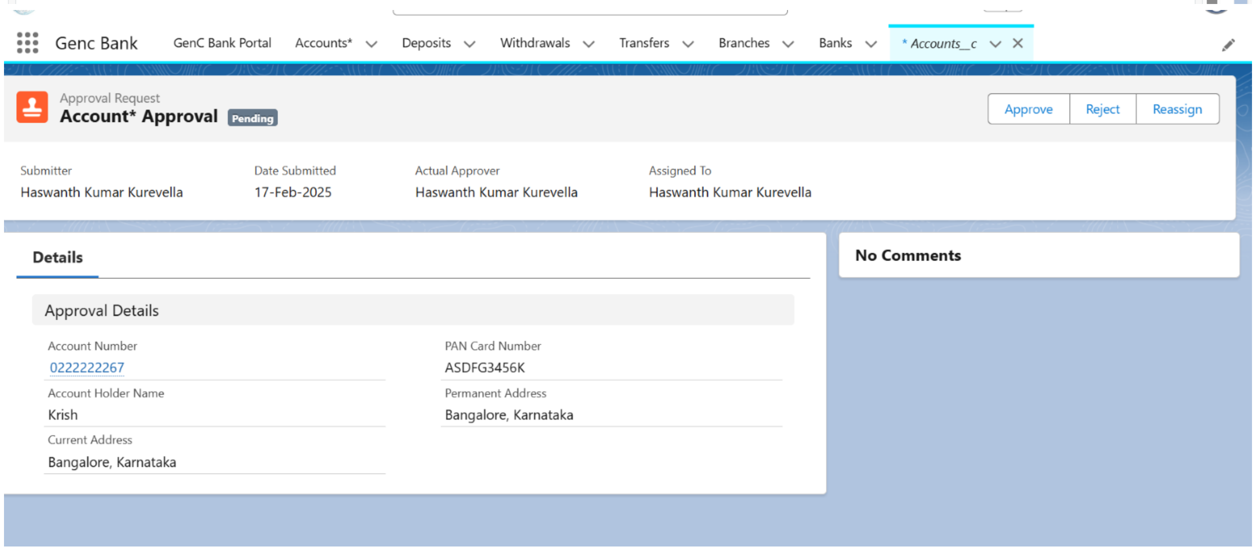


Approval Steps

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Hide Actions | Edit	1	KYC Manager			User:Haswanth Kumar Kurevella	Final Rejection

Approval Actions Add Existing | Add New

You have not yet defined any actions



Approval Request
Account* Approval Pending

Submitter	Date Submitted	Actual Approver	Assigned To
Haswanth Kumar Kurevella	17-Feb-2025	Haswanth Kumar Kurevella	Haswanth Kumar Kurevella

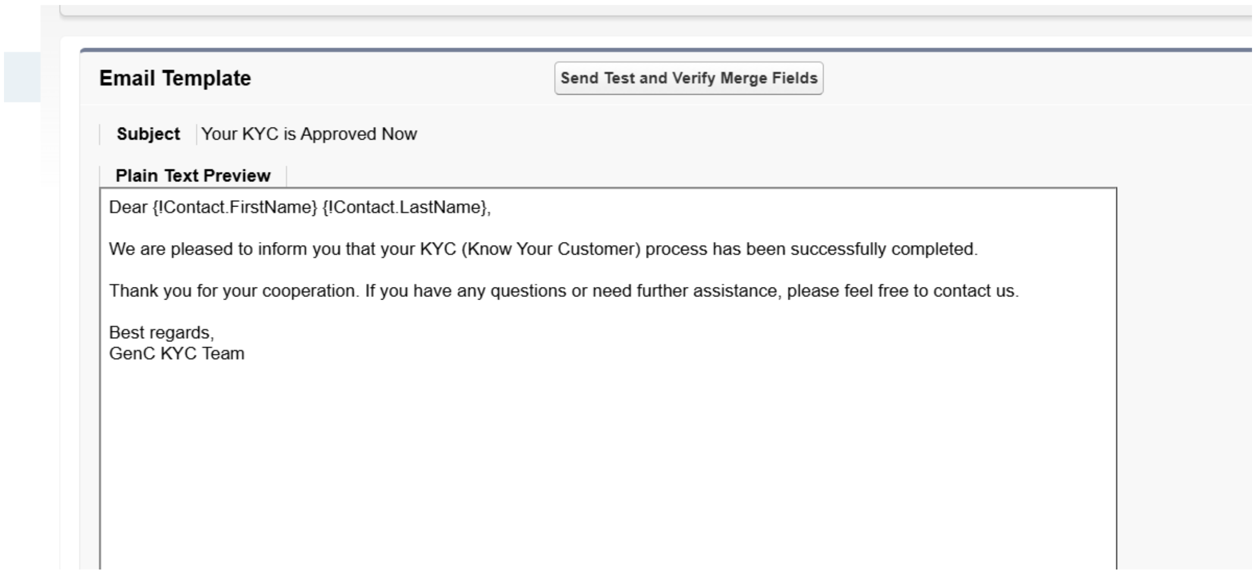
Details

Approval Details

Account Number: 0222222267	PAN Card Number: ASDFG3456K
Account Holder Name: Krish	Permanent Address: Bangalore, Karnataka
Current Address: Bangalore, Karnataka	

No Comments

➢ **Final Approval Actions:**



Email Template

Subject Your KYC is Approved Now

Plain Text Preview

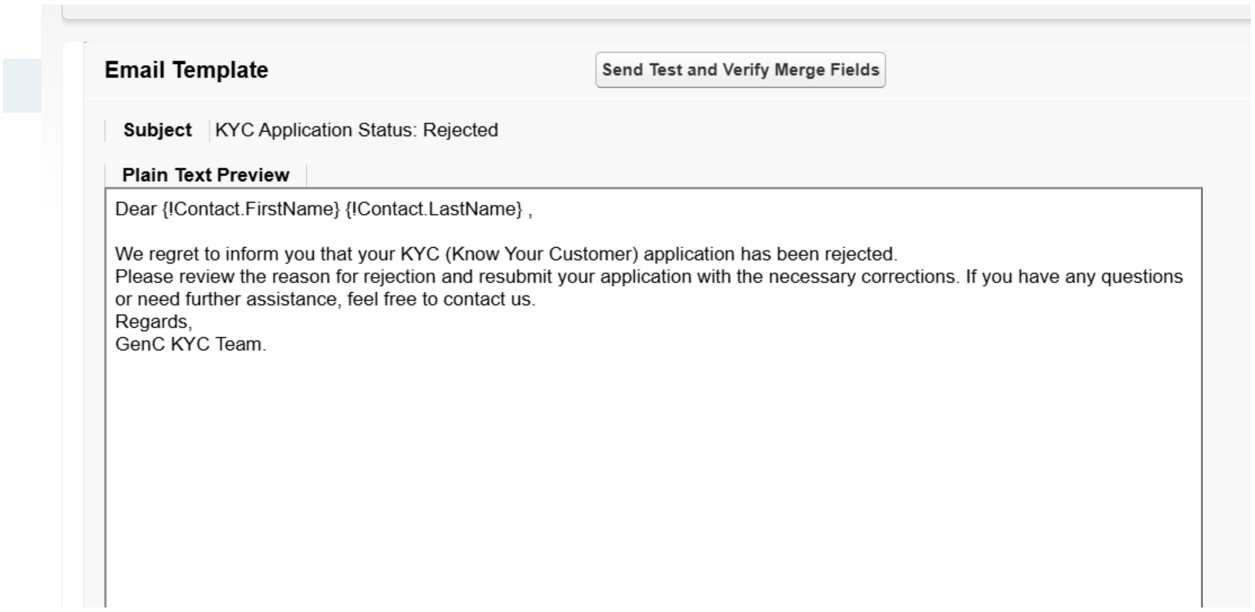
Dear {!Contact.FirstName} {!Contact.LastName},

We are pleased to inform you that your KYC (Know Your Customer) process has been successfully completed.

Thank you for your cooperation. If you have any questions or need further assistance, please feel free to contact us.

Best regards,
GenC KYC Team

➢ **Final Rejection Actions:**



Subject KYC Application Status: Rejected

Plain Text Preview

Dear {!Contact.FirstName} {!Contact.LastName} ,

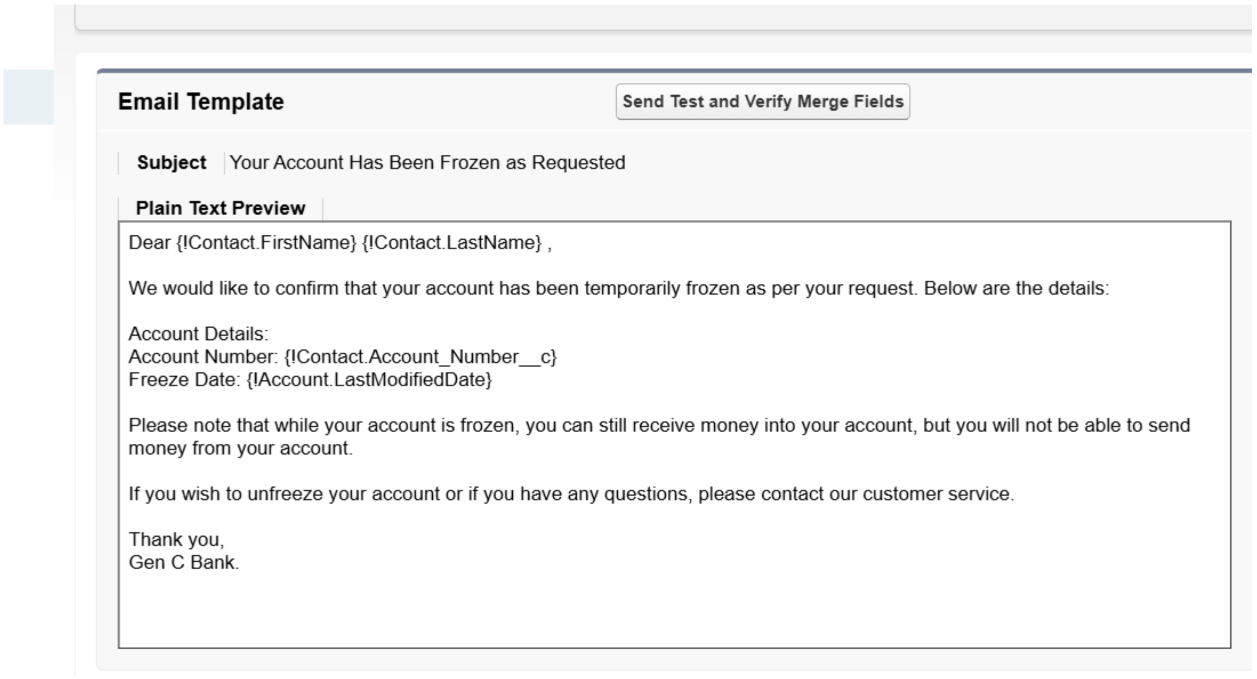
We regret to inform you that your KYC (Know Your Customer) application has been rejected.
Please review the reason for rejection and resubmit your application with the necessary corrections. If you have any questions or need further assistance, feel free to contact us.
Regards,
GenC KYC Team.

- **Description:** This approval process starts when a customer creates a bank account. If all documents are verified and approved, a KYC verification record is created, and the customer is notified of the successful approval. This process ensures compliance with regulatory requirements and helps prevent fraudulent activities.

Flows

Screen Flows

- **Flow Name: Freeze/Unfreeze**
  - **Purpose:** This flow is designed to manage the freezing and unfreezing of Account records through a screen interface. It ensures that appropriate notifications and emails are sent, updates the Account record accordingly, and manages user assignments based on the account's status.
  - **Email Template: Freeze**



Subject Your Account Has Been Frozen as Requested

Plain Text Preview

Dear {!Contact.FirstName} {!Contact.LastName} ,

We would like to confirm that your account has been temporarily frozen as per your request. Below are the details:

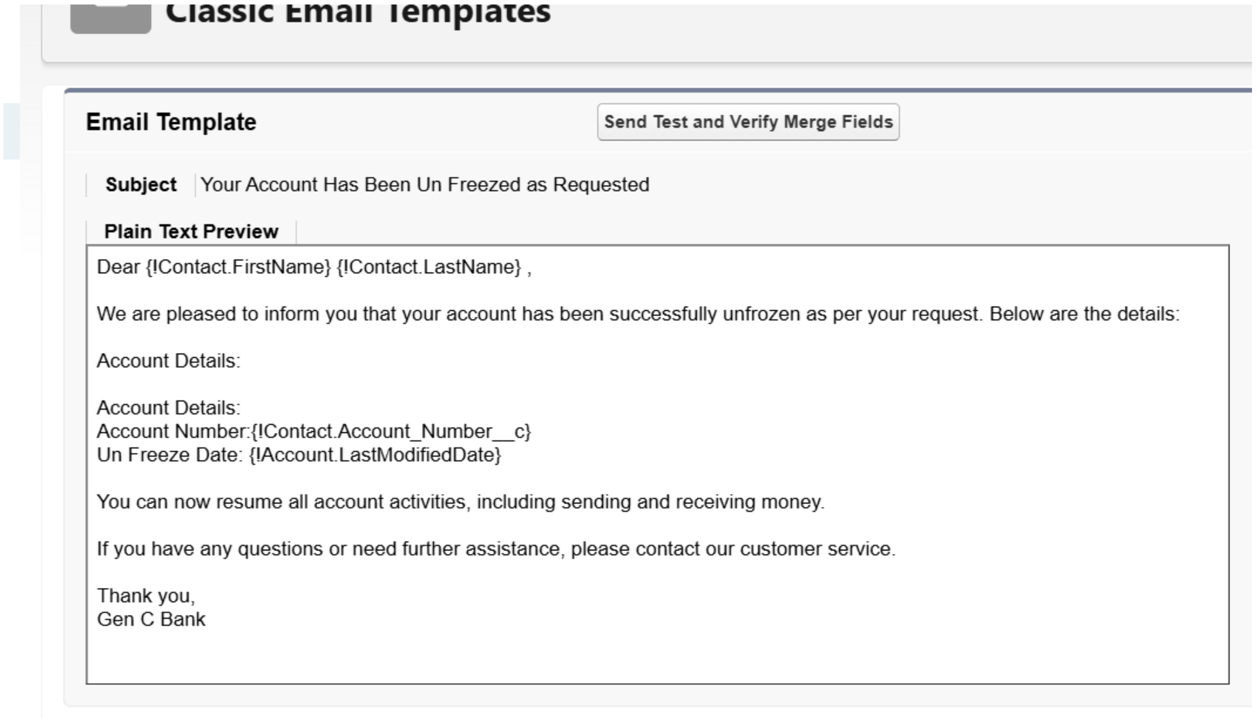
Account Details:
Account Number: {!Contact.Account_Number__c}
Freeze Date: {!Account.LastModifiedDate}

Please note that while your account is frozen, you can still receive money into your account, but you will not be able to send money from your account.

If you wish to unfreeze your account or if you have any questions, please contact our customer service.

Thank you,
Gen C Bank.

➢ Email Template: Unfreeze



Subject Your Account Has Been Un Freezed as Requested

Plain Text Preview

Dear {!Contact.FirstName} {!Contact.LastName} ,

We are pleased to inform you that your account has been successfully unfrozen as per your request. Below are the details:

Account Details:

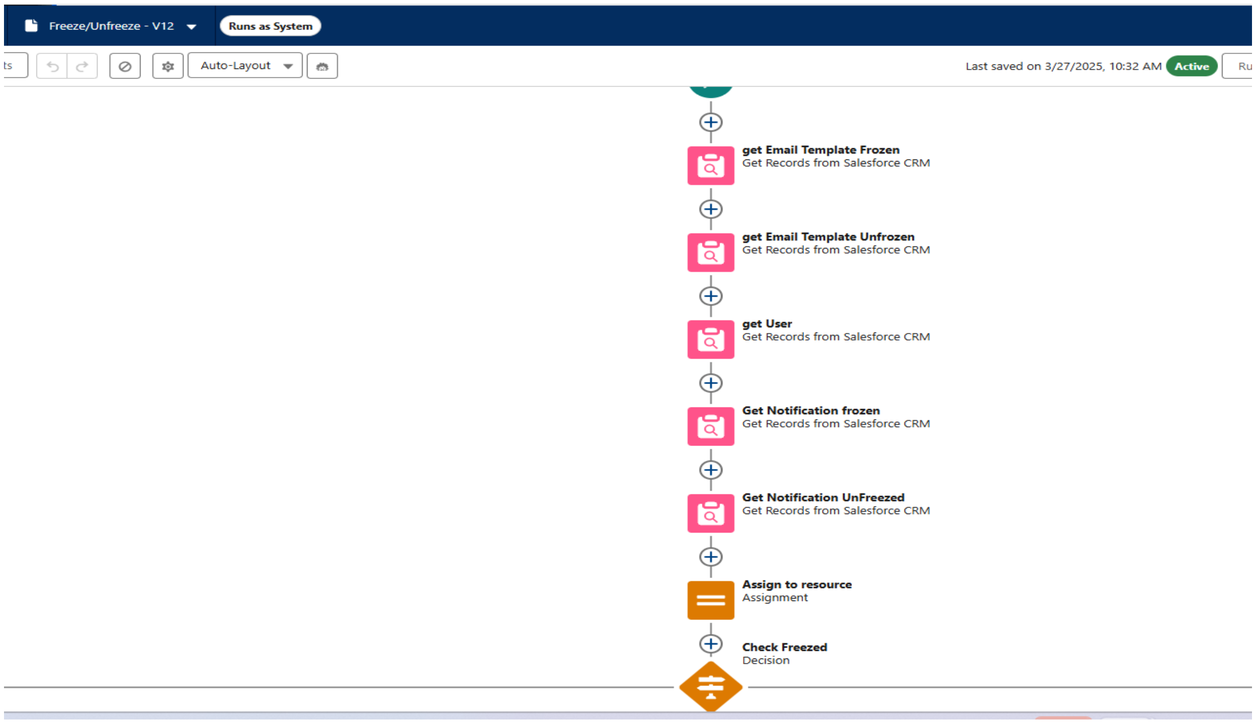
Account Details:
Account Number:{!Contact.Account_Number__c}
Un Freeze Date: {!Account.LastModifiedDate}

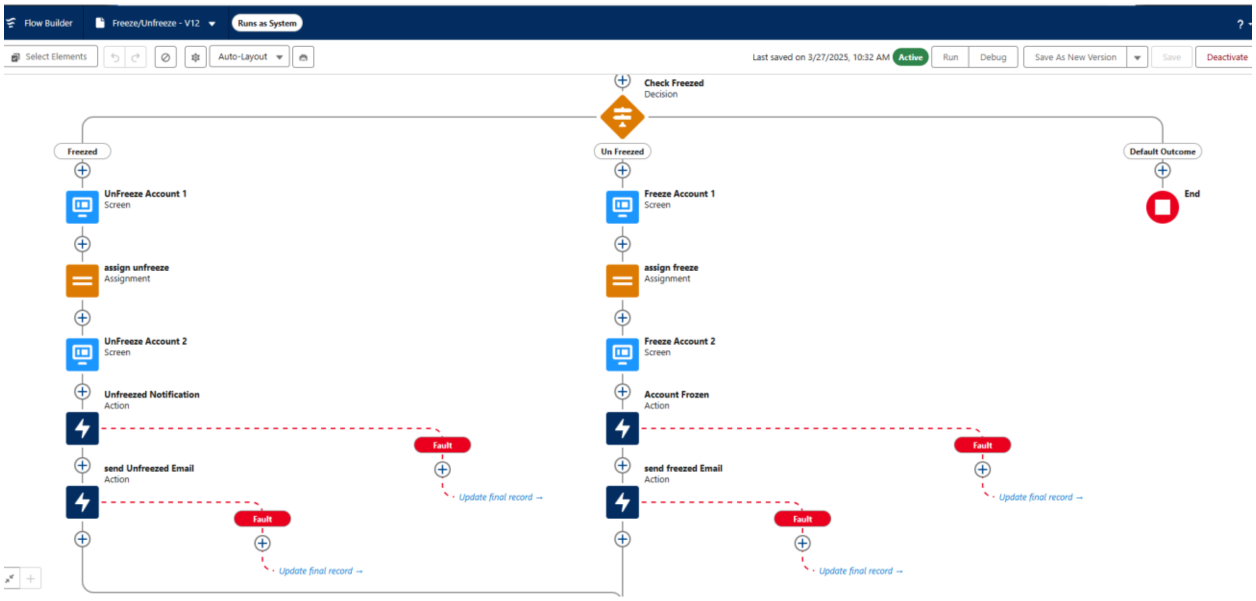
You can now resume all account activities, including sending and receiving money.

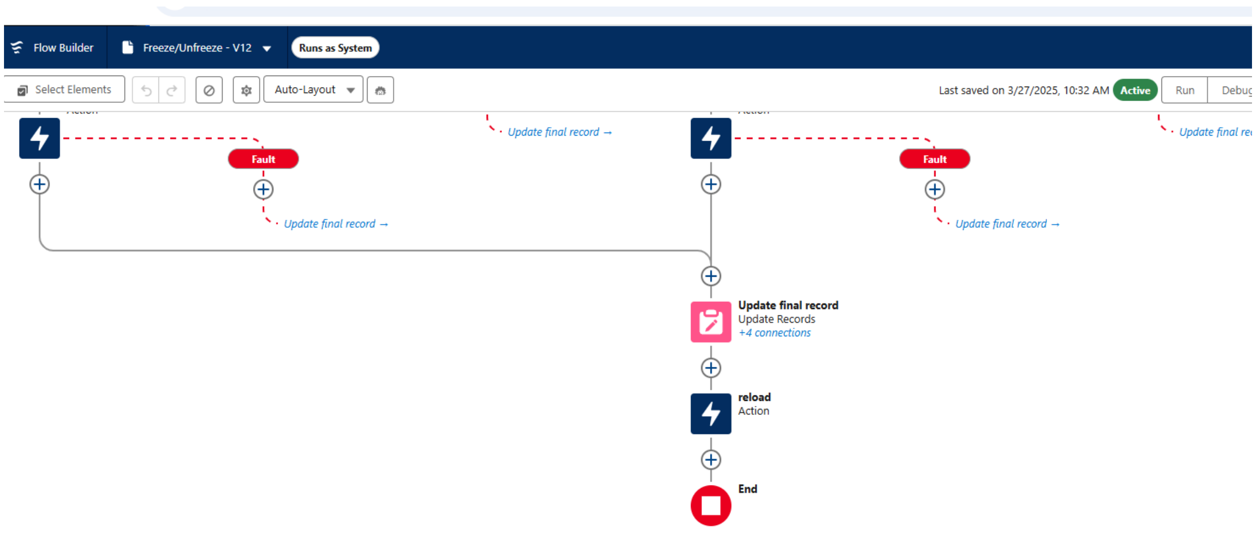
If you have any questions or need further assistance, please contact our customer service.

Thank you,
Gen C Bank

- **Description:** This is a Screen Flow that activates upon the update of an Account record. The flow performs the following actions based on the value of the Freeze__c field:
  - **Check Freeze Status:**
  - **If Freeze__c is False:**
    - The flow checks the Freeze__c field of the Account record.
  - **If Freeze__c is True:**
    - Sends a frozen notification.
    - Send a frozen email.
    - Updates the Account record to reflect the frozen status. Reloads the page to reflect changes.
    - Assigns users based on the account's frozen status.
  - **If Freeze__c is False:**
    - Sends an unfrozen notification.
    - Send an unfrozen email.
    - Updates the Account record to reflect the unfrozen status.
    - Reloads the page to reflect changes.
    - Unassigns users based on the account's unfrozen status.


Freeze/Unfreeze - V12
Runs as System
Auto-Layout
Last saved on 3/27/2025, 10:32 AM
Active
get Email Template Frozen
Get Records from Salesforce CRM
get Email Template Unfrozen
Get Records from Salesforce CRM
get User
Get Records from Salesforce CRM
Get Notification frozen
Get Records from Salesforce CRM
Get Notification UnFreezed
Get Records from Salesforce CRM
Assign to resource
Assignment
Check Freezed
Decision

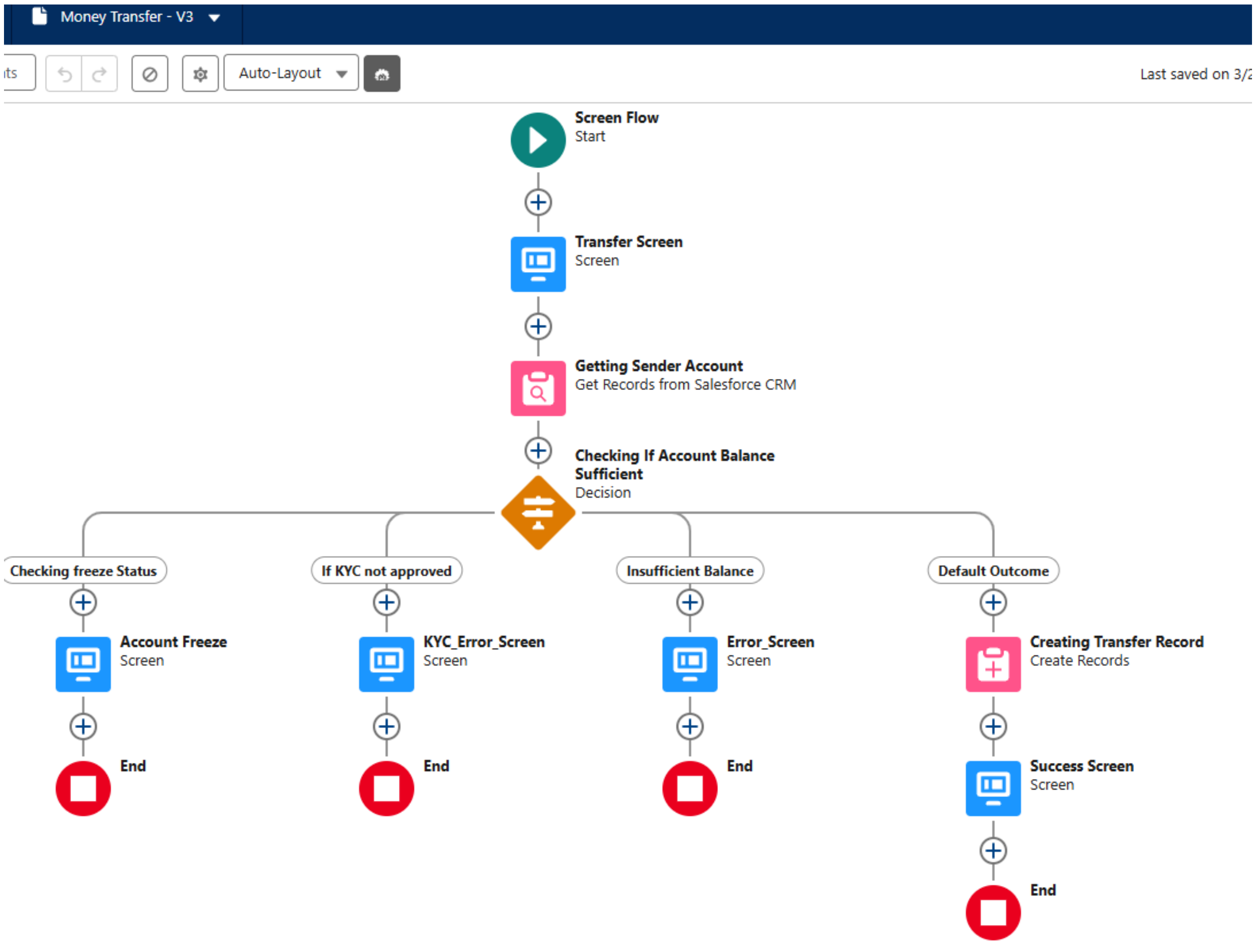

Flow Builder
Freeze/Unfreeze - V12
Runs as System
Select Elements
Auto-Layout
Last saved on 3/27/2025, 10:32 AM
Active
Run
Debug
Save As New Version
Save
Deactivate
Check Freezed
Decision
Freezed
UnFreeze Account 1
Screen
assign unfreeze
Assignment
UnFreeze Account 2
Screen
Unfreezed Notification
Action
Fault
Update final record
send Unfreezed Email
Action
Fault
Update final record
Un Freezed
Freeze Account 1
Screen
assign freeze
Assignment
Freeze Account 2
Screen
Account Frozen
Action
Fault
Update final record
send freezed Email
Action
Fault
Update final record
Default Outcome
End



❖ Flow Name: Money Transfer

- **Purpose:** This flow is designed to facilitate the transfer of funds between accounts through a screen interface. It ensures that all necessary checks are performed before creating a transfer record and provides appropriate feedback to the user.
- **Description:** This is a Screen Flow that initiates when a user navigates to the 'Transfer' screen. The flow performs the following actions:
  - **Retrieve Sender's Account Details:**
    - The flow retrieves the sender's account details based on the selected 'Sender Account'. Perform Checks:
  - **Perform Checks:**
    - **Frozen Account Check:** Verifies if the sender's account is frozen.
    - **KYC Approval Check:** Verifies if the sender's KYC is approved.
    - **Balance Check:** Verifies if the sender's account has sufficient balance for the transfer. If All Checks Pass:
  - **If All Checks Pass:**
    - Creates a new 'Transfer' record. Displays a success message to the user.

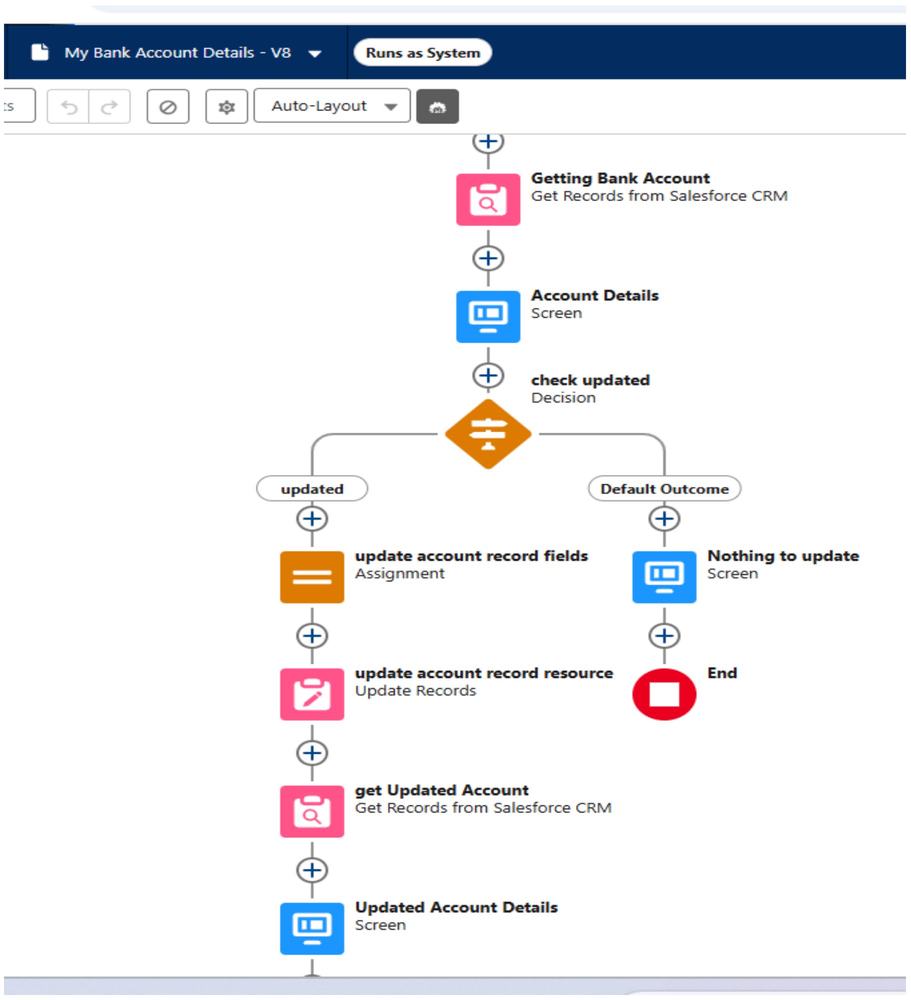
- **If Any Check Fails:**
  - Displays an error message with relevant information to the user. Objects and Fields:



❖ Flow Name: My Bank Account Details

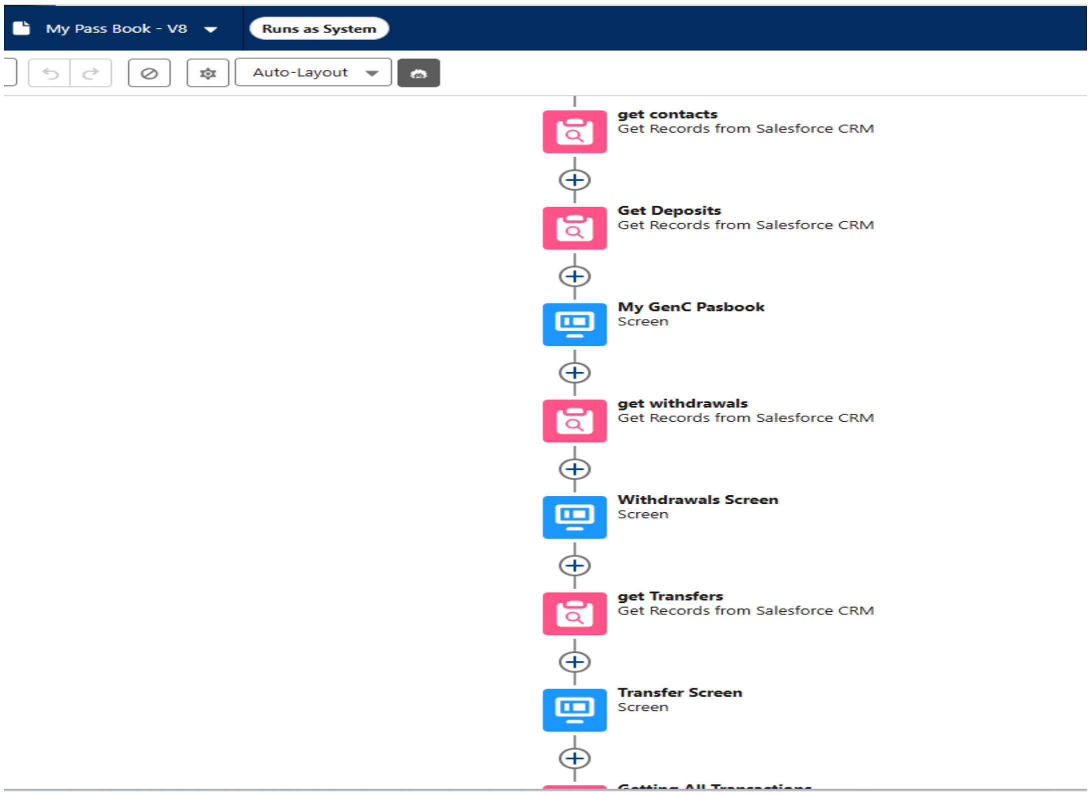
- **Purpose:** This flow is designed to allow users to review and update their contact and associated bank account details through a screen interface. It ensures that any changes made by the user are reflected in the contact record.
- **Description:** This is a Screen Flow that performs the following actions:
  - **Retrieve User's Contact and Bank Account Details:** The flow retrieves the user's contact record and their associated bank account details.

- **Present Details for Review:** The flow presents these details on a screen for the user to review. Allow Editing of Specific Fields:
- **Allow Editing of Specific Fields:** The user can edit certain fields such as phone number, permanent address, and current address.
- **Update Contact Record:** If any of these fields have been changed, the flow updates the contact record. Displays the updated information on a separate screen.
- **No Changes Made:** If no changes were made, it displays a message indicating that there is nothing to update.



❖ Flow Name: Pass Book

- **Purpose:** This flow is designed to provide users with an overview of their financial activities, including deposits, withdrawals, transfers, and all transactions, through a screen interface. It allows users to navigate between different screens to view specific types of transactions.
- **Description:** This is a Screen Flow that performs the following actions:
    - **Retrieve User's Contact Record:**
        - The flow retrieves the user's contact record using the Contact - Id field.
    - **Fetch and Display Deposits:**
        - Retrieves deposit records using the Deposit__c object and fields: CreatedDate, Name, Amount__c, and Description__c. Displays these deposit records on a screen.
    - **Fetch and Display Withdrawals:**
        - Retrieves withdrawal records using the Withdrawal__c object and fields: CreatedDate, Name, and Withdrawal_Amount__c. Displays these withdrawal records on a screen.
    - **Fetch and Display Transfers:**
        - Retrieves transfer records using the Transfer__c object and fields: CreatedDate, Name, Beneficiary_Name__c, and Transfer_Amount__c. Displays these transfer records on a screen.
    - **Fetch and Display All Transactions:**
        - Retrieves all transaction records using the Transaction__c object and fields: CreatedDate, Name, Sender_Name__c, Receiver_Name__c, and Amount__c. Displays these transaction records on a screen.
    - **Navigation Between Screens:**
        - Allows users to navigate between screens to view different types of transactions.



get contacts
Get Records from Salesforce CRM

Get Deposits
Get Records from Salesforce CRM

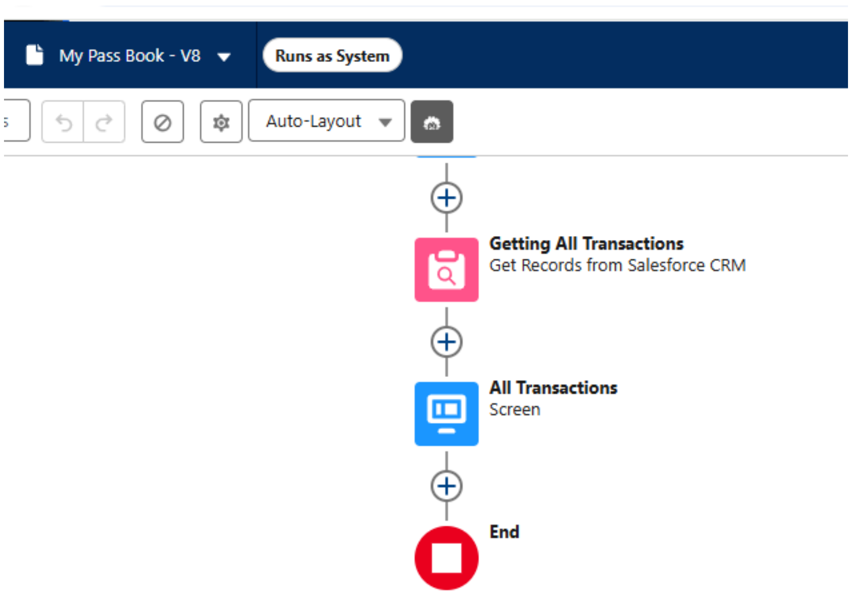
My GenC Pasbook
Screen

get withdrawals
Get Records from Salesforce CRM

Withdrawals Screen
Screen

get Transfers
Get Records from Salesforce CRM

Transfer Screen
Screen



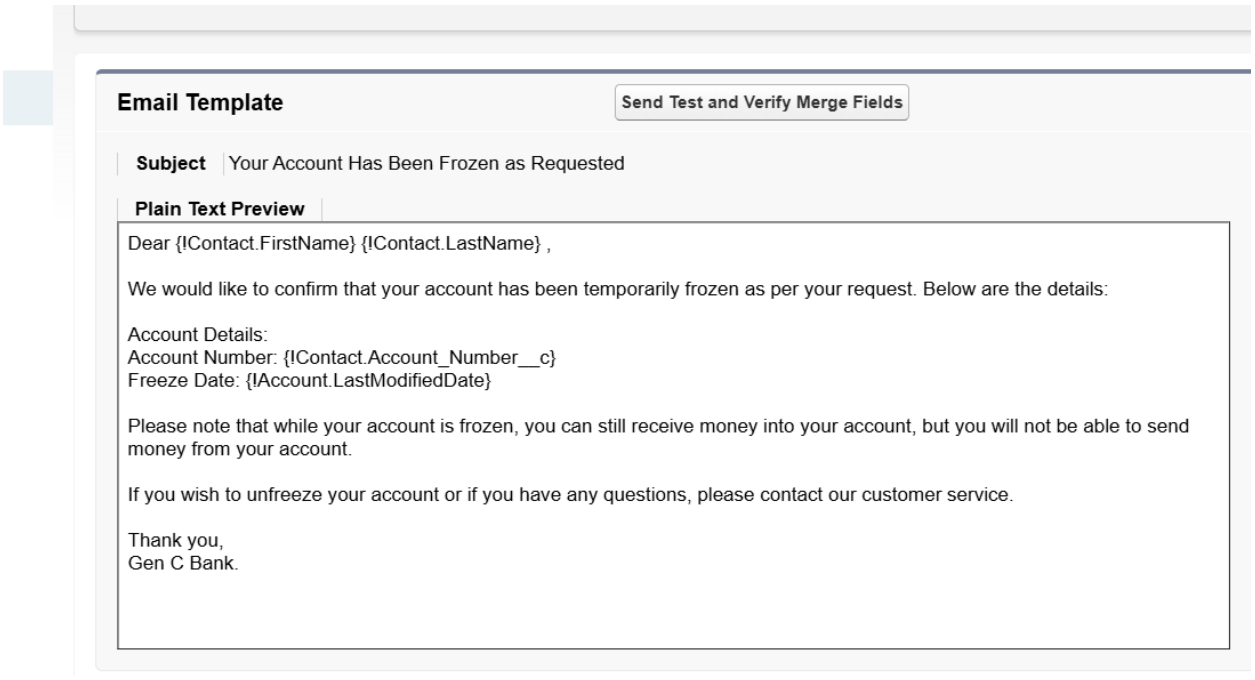
Getting All Transactions
Get Records from Salesforce CRM

All Transactions
Screen

End

❖ Flow Name: Freeze/Unfreeze for Customer

- **Purpose:** This flow is designed to manage the freezing and unfreezing of Account records through a screen interface. It ensures that appropriate notifications and emails are sent, updates the Account record accordingly, and manages user assignments based on the account's status.
- **Email Template: Freeze**



Subject Your Account Has Been Frozen as Requested

Plain Text Preview

Dear {!Contact.FirstName} {!Contact.LastName} ,

We would like to confirm that your account has been temporarily frozen as per your request. Below are the details:

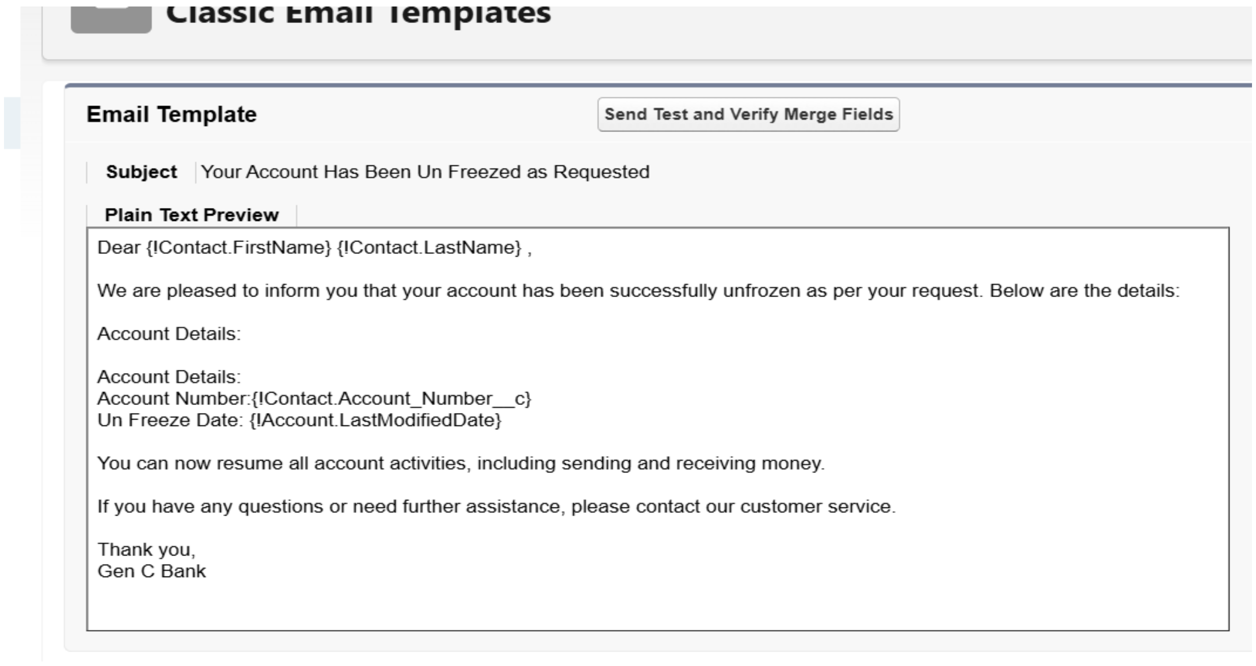
Account Details:
Account Number: {!Contact.Account_Number__c}
Freeze Date: {!Account.LastModifiedDate}

Please note that while your account is frozen, you can still receive money into your account, but you will not be able to send money from your account.

If you wish to unfreeze your account or if you have any questions, please contact our customer service.

Thank you,
Gen C Bank.

- **Email Template: Unfreeze**



Subject: Your Account Has Been Un Freezed as Requested

Plain Text Preview

Dear {!Contact.FirstName} {!Contact.LastName} ,

We are pleased to inform you that your account has been successfully unfrozen as per your request. Below are the details:

Account Details:

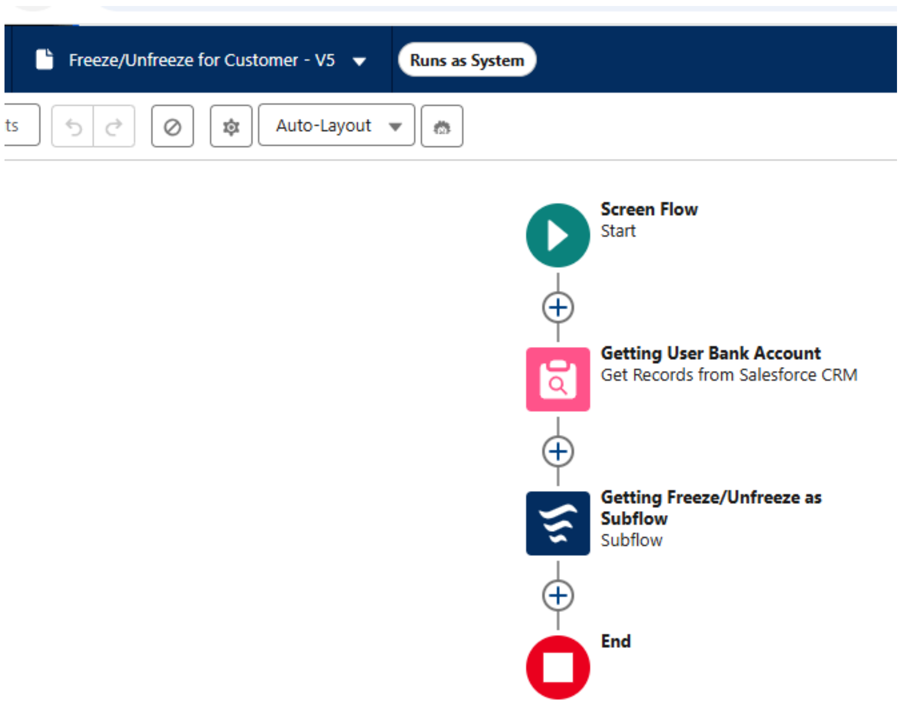
Account Details:
Account Number:{!Contact.Account_Number__c}
Un Freeze Date: {!Account.LastModifiedDate}

You can now resume all account activities, including sending and receiving money.

If you have any questions or need further assistance, please contact our customer service.

Thank you,
Gen C Bank

- **Description:** This is a Screen Flow that activates upon the update of an Account record. The flow performs the following actions based on the value of the Freeze__c field:
  - **Check Freeze Status:**
  - **If Freeze__c is False:**
    - The flow checks the Freeze__c field of the Account record.
  - **If Freeze__c is True:**
    - Sends a frozen notification.
    - Send a frozen email.
    - Updates the Account record to reflect the frozen status. Reloads the page to reflect changes.
    - Assigns users based on the account's frozen status.
  - **If Freeze__c is False:**
    - Sends an unfrozen notification.
    - Send an unfrozen email.
    - Updates the Account record to reflect the unfrozen status.
    - Reloads the page to reflect changes.
    - Unassigns users based on the account's unfrozen status.


Freeze/Unfreeze for Customer - V5
Runs as System
ts
Auto-Layout
Screen Flow
Start
Getting User Bank Account
Get Records from Salesforce CRM
Getting Freeze/Unfreeze as Subflow
Subflow
End

❖ Flow Name: Bank Account Creation

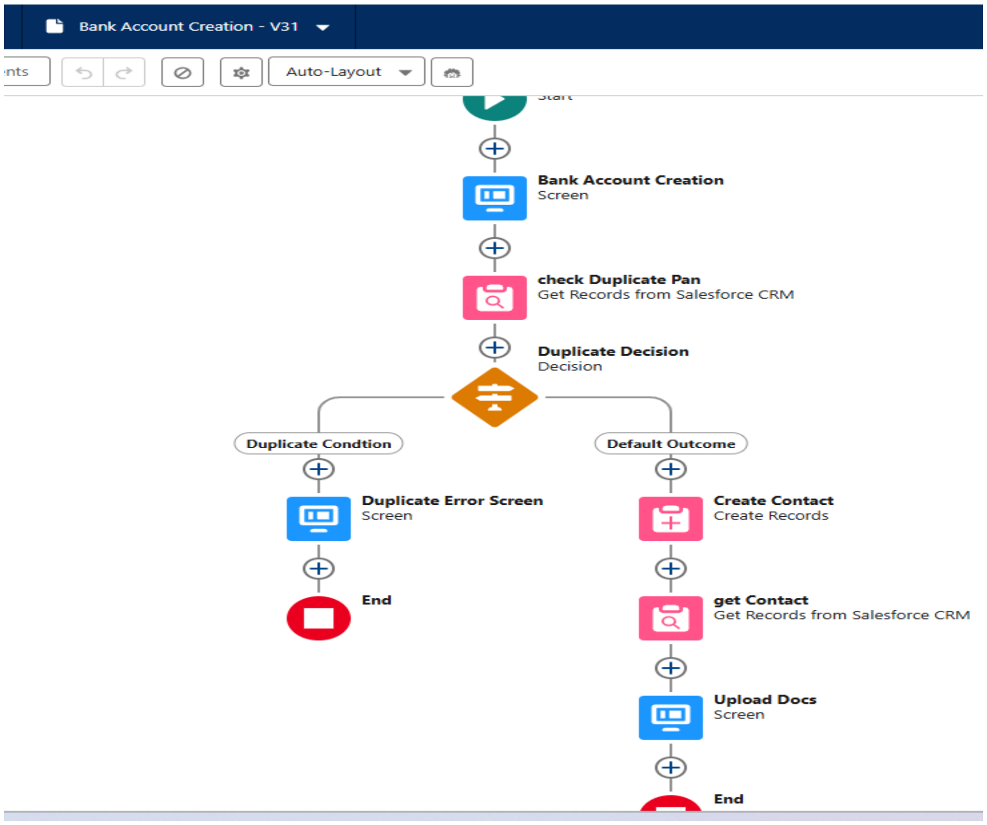
➢ **Purpose:** This flow is designed to facilitate the creation of new bank accounts through a screen interface. It ensures that all necessary details are collected, validated, and processed, and handles duplicate account checks.

➢ **Description:** This is a Screen Flow that performs the following actions:

- **Initiation:**
  - The flow begins by asking the user to select the type of bank account they want to create.
- **Collect User Details:**
  - **Collects various details such as:**
    - Name
    - PAN card number
    - Phone number
    - Email ID
    - Branch selection

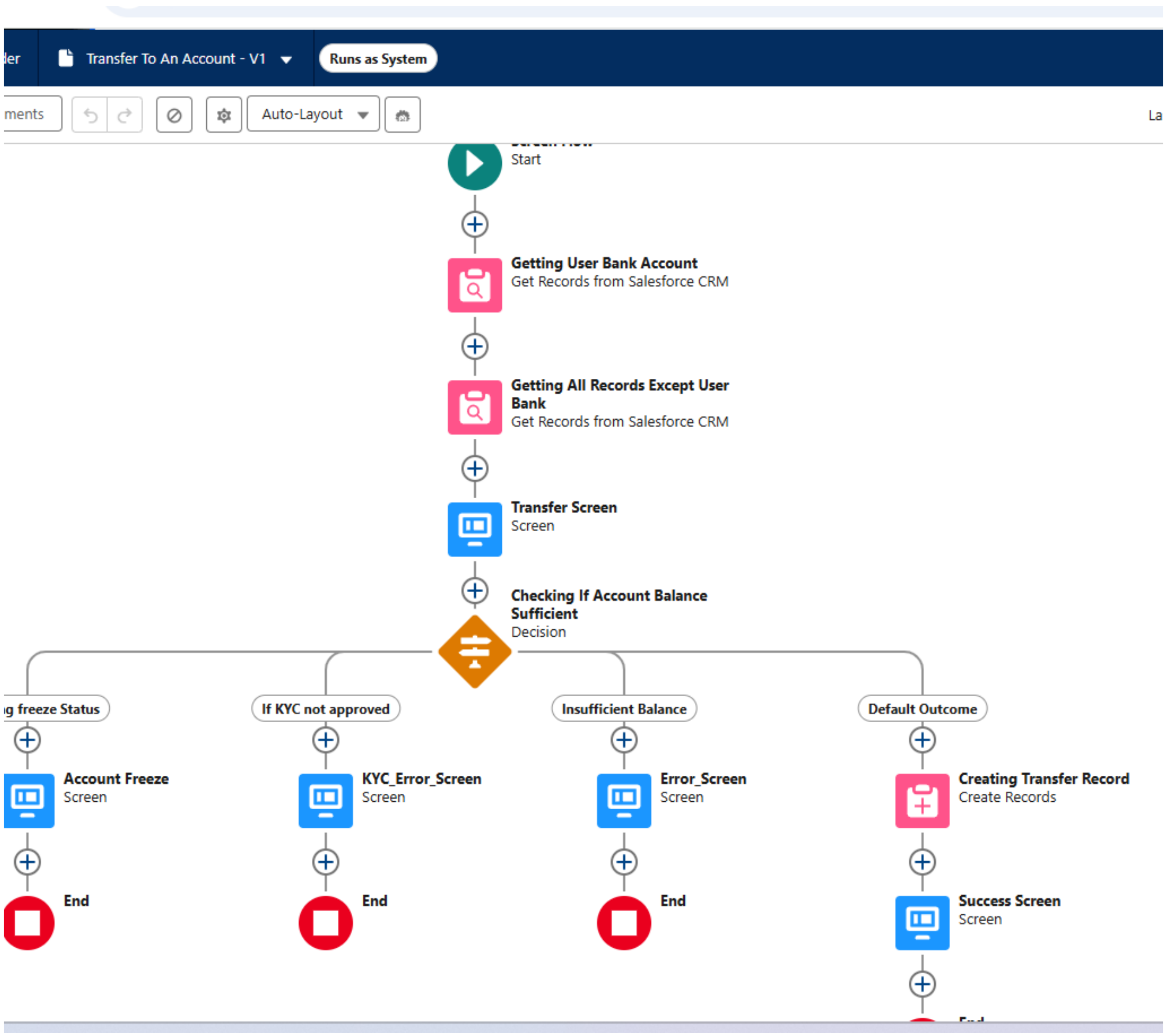
- Initial deposit amount
- Permanent and current addresses

- **Validate Inputs:**
  - Validates the inputs provided by the user.
- **Check for Duplicate Accounts:**
  - Checks if an account with the provided PAN number already exists using the
  - Contact - PAN_Card_Number__c field.
- **Create New Contact Record:**
  - If no duplicate is found, creates a new contact record with the provided details.
  - Proceeds to the document upload screen.
- **Handle Duplicate Accounts:**
  - If a duplicate is found, it displays an error message.
  - Allows the user to go back and correct their input.



❖ Flow Name: Transfer To An Account

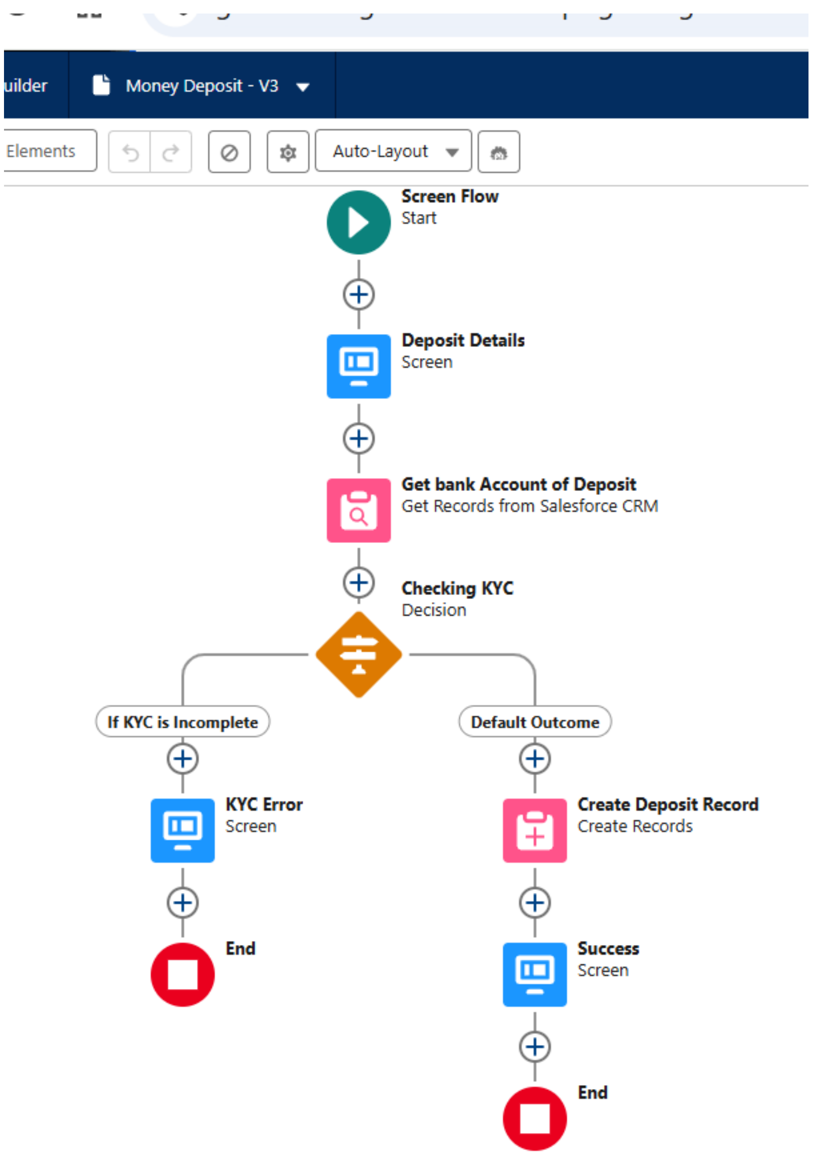
- **Purpose:** This flow is designed to facilitate the transfer of funds between bank accounts through a screen interface. It ensures that all necessary checks are performed before creating a transfer record and provides appropriate feedback to the user.
- **Description:** This is a Screen Flow that performs the following actions:
  - **Initiation:**
    - The flow retrieves the user's bank account details and all other bank accounts except the users.
  - **Present Transfer Screen:**
    - Presents a transfer screen where the user can select a receiver account and enter the transfer amount.
  - **Perform Checks:**
    - **Balance Check:** Verifies if the sender's account balance is sufficient for the transfer.
    - **KYC Approval Check:** Verifies if the sender's KYC is approved.
    - **Frozen Account Check:** Verifies if the sender's account is frozen.
  - **Create Transfer Record**:
    - If all conditions are met, a new transfer record is created.
    - Displays a success message to the user.
  - **Handle Errors:**
    - If any condition fails, it displays appropriate error messages.



❖ Flow Name: Money Deposit

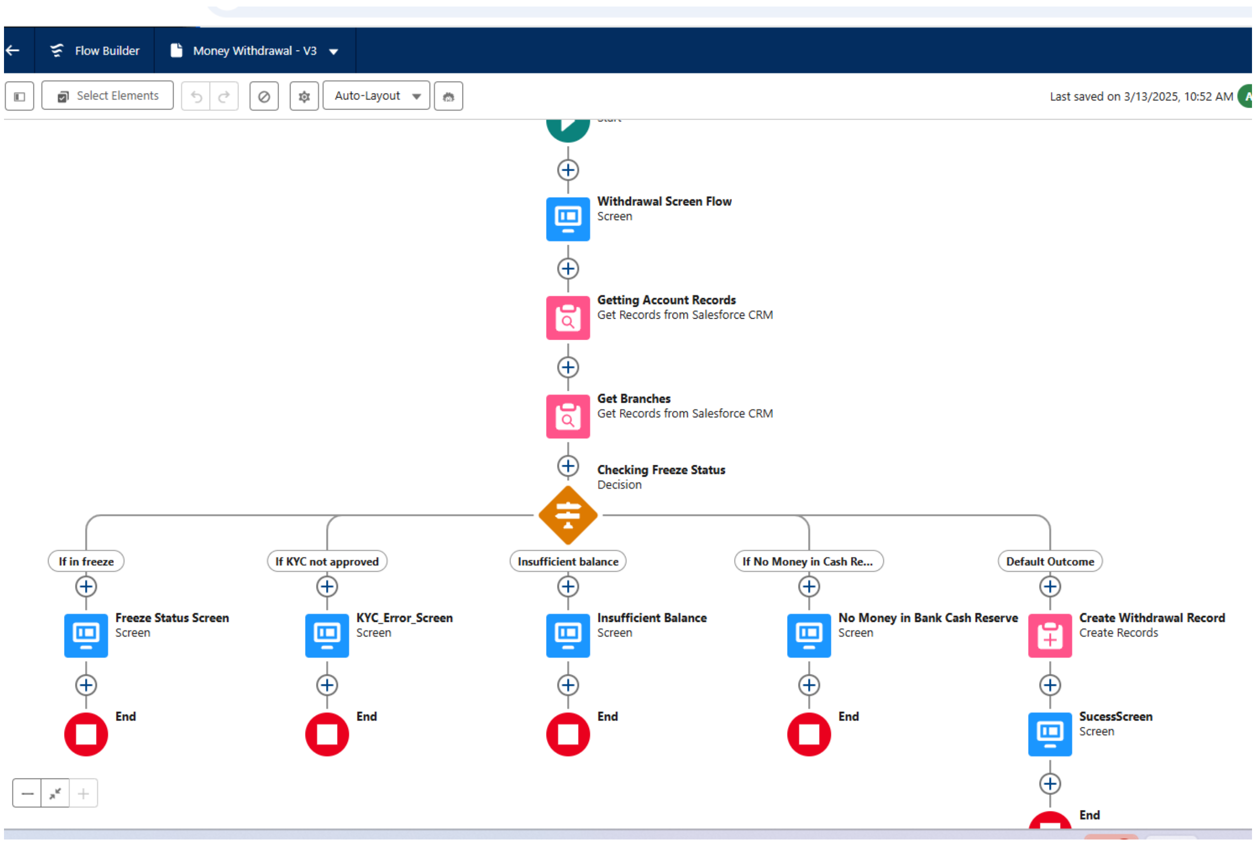
- **Purpose:** This flow is designed to facilitate the deposit of funds into a bank account through a screen interface. It ensures that the KYC status is verified before creating a deposit record and provides appropriate feedback to the user.
- **Description:** This is a Screen Flow that performs the following actions:
  - **Initiation:**
    - The flow prompts users to enter deposit details including account ID, amount, and description.

- **Check KYC Status:**
  - Check the KYC status of the provided account using the Contact - Id field.
  - If KYC is incomplete, an error message is displayed.
- **Create Deposit Record:**
  - If KYC is complete, create a deposit record with the provided details.
  - Displays a success message to the user.



❖ Flow Name: Money Withdrawal

- **Purpose:** This flow is designed to facilitate the withdrawal of funds from a bank account through a screen interface. It ensures that all necessary checks are performed before creating a withdrawal record and provides appropriate feedback to the user.
- **Description:** This is a Screen Flow that performs the following actions:
  - **Initiation:**
    - The flow begins with a withdrawal screen where users input their account number and withdrawal amount.
  - **Perform Checks:**
    - **Freeze Status Check:** Verifies if the account is frozen using the Contact - Freeze__c field.
    - **KYC Approval Check:** Verifies if the KYC is approved using the Contact - KYC__c field.
    - **Balance Sufficiency Check:** Verifies if the account balance is sufficient using the Contact - Account_Balance__c field.
    - **Cash Reserve Availability Check:** Verifies if the cash reserve is available using the Account - Cash_Reserve__c field.
  - **Handle Errors:**
    - If any check fails, it displays appropriate error messages.
  - **Create Withdrawal Record:**
    - If all checks pass, creates a withdrawal record using the Withdrawal__c object and fields: Bank_Account__c and Withdrawal_Amount__c.
    - Displays a success message to the user.

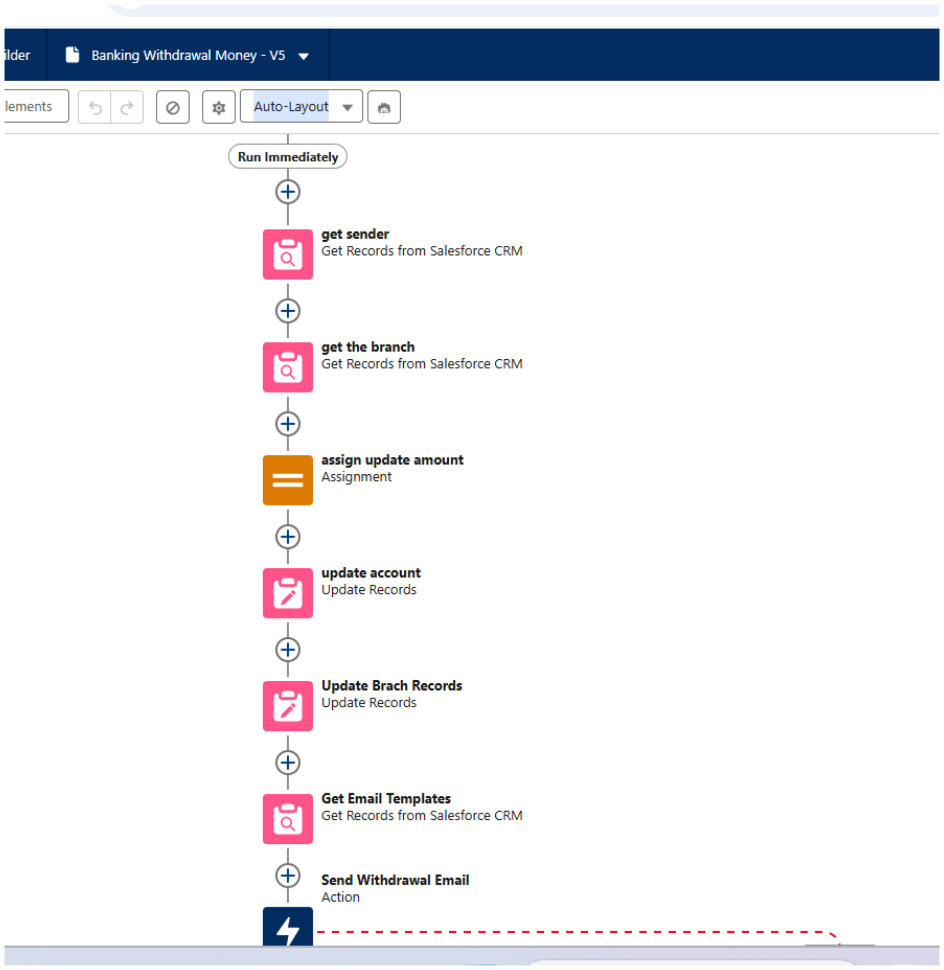


Record-Trigger Flows

❖ Flow Name: Banking Money Withdrawal

- **Purpose:** This flow is designed to manage the processes triggered by the creation of a Withdrawal__c record. It ensures that account balances and cash reserves are updated, and that notifications and emails are sent to the relevant parties.
- **Description:** This is a Record-Triggered Flow that performs the following actions upon the creation of a Withdrawal__c record:
  - **Retrieve Associated Contact and Account:**
    - The flow retrieves the associated Contact (sender) and Account (branch) using the Contact - Id and Contact - AccountId fields.
  - **Update Account Balance and Cash Reserve:**

        - The flow reads the Withdrawal__c - Withdrawal_Amount__c field to calculate the updated account balance and cash reserve.
        - Updates the account balance and cash reserve in the Account - Cash_Reserve__c field.
    - **Send Email to Sender:**
        - Fetches the appropriate email template using the EmailTemplate - Id and EmailTemplate - Name fields.
        - Sends an email to the sender (Contact).
    - **Send Custom Notifications:**
        - Retrieves the relevant CustomNotificationType and associated users.
        - Sends custom notifications to those users.



get sender
Get Records from Salesforce CRM

get the branch
Get Records from Salesforce CRM

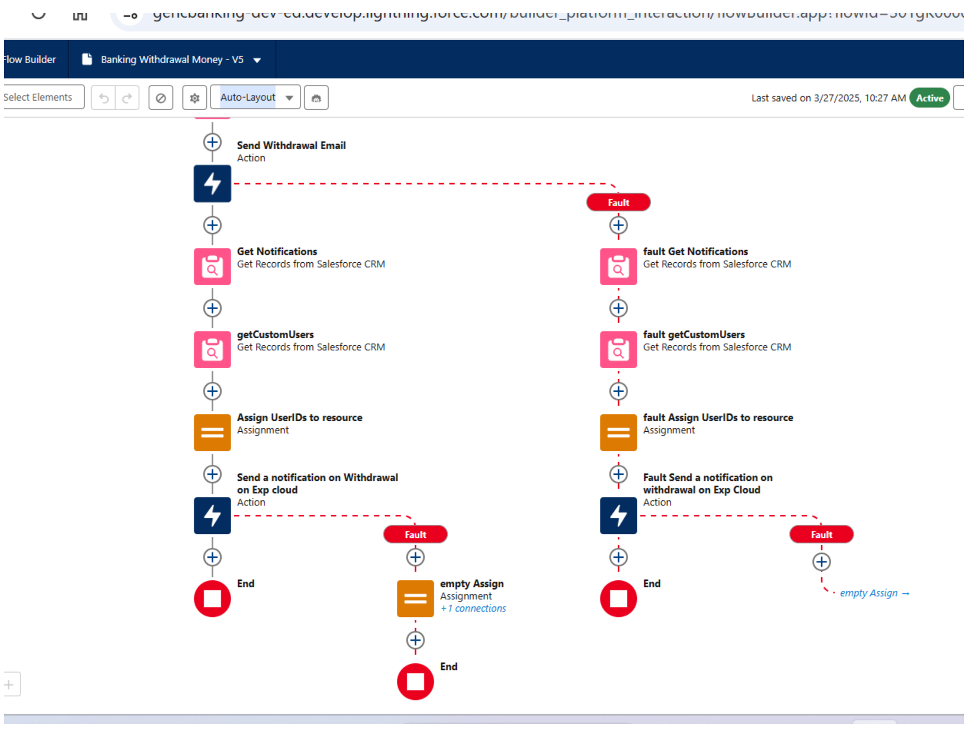
assign update amount
Assignment

update account
Update Records

Update Brach Records
Update Records

Get Email Templates
Get Records from Salesforce CRM

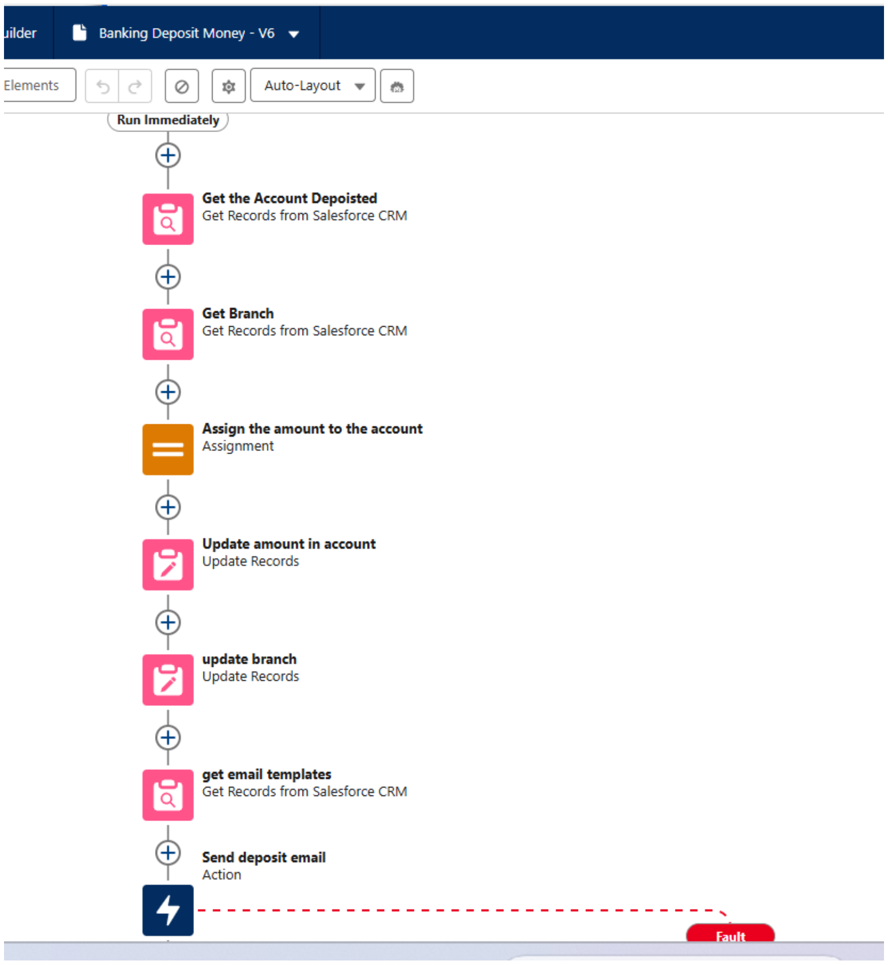
Send Withdrawal Email
Action


Flow Builder
Banking Withdrawal Money - V5
Select Elements
Auto-Layout
Last saved on 3/27/2025, 10:27 AM
Active
Send Withdrawal Email
Action
Fault
Get Notifications
Get Records from Salesforce CRM
fault Get Notifications
Get Records from Salesforce CRM
getCustomUsers
Get Records from Salesforce CRM
fault getCustomUsers
Get Records from Salesforce CRM
Assign UserIDs to resource
Assignment
fault Assign UserIDs to resource
Assignment
Send a notification on Withdrawal on Exp cloud
Action
Fault Send a notification on withdrawal on Exp Cloud
Action
Fault
Fault
End
empty Assign
Assignment
+1 connections
End
empty Assign →
End

❖ Flow Name: Banking Deposit Money

- **Purpose:** This flow is designed to manage the processes triggered by the creation of a Deposit__c record. It ensures that account balances are updated, and that notifications and emails are sent to the relevant parties.
- **Description:** This is a Record-Triggered Flow that performs the following actions upon the creation of a Deposit__c record:
  - **Retrieve Associated Contact and Account:**
    - The flow retrieves the associated contact using the Deposit__c - Bank_Account__c field.
    - Retrieves the associated account using the Contact - AccountId field.
  - **Update Balances:**
    - The flow reads the Deposit__c - Amount__c field to update the balances of both the contact and their account.

        - Updates the contact's account balance. Updates the account's cash reserve in the Account - Cash_Reserve__c field.
    - **Send Email Notification:**
        - Sends an email to the contact using a predefined template ('Amount Deposit Email').
    - **Send Custom Notifications:**
        - Triggers custom notifications to specific users based on the 'Deposit' notification type.



Get the Account Depoisted
Get Records from Salesforce CRM

Get Branch
Get Records from Salesforce CRM

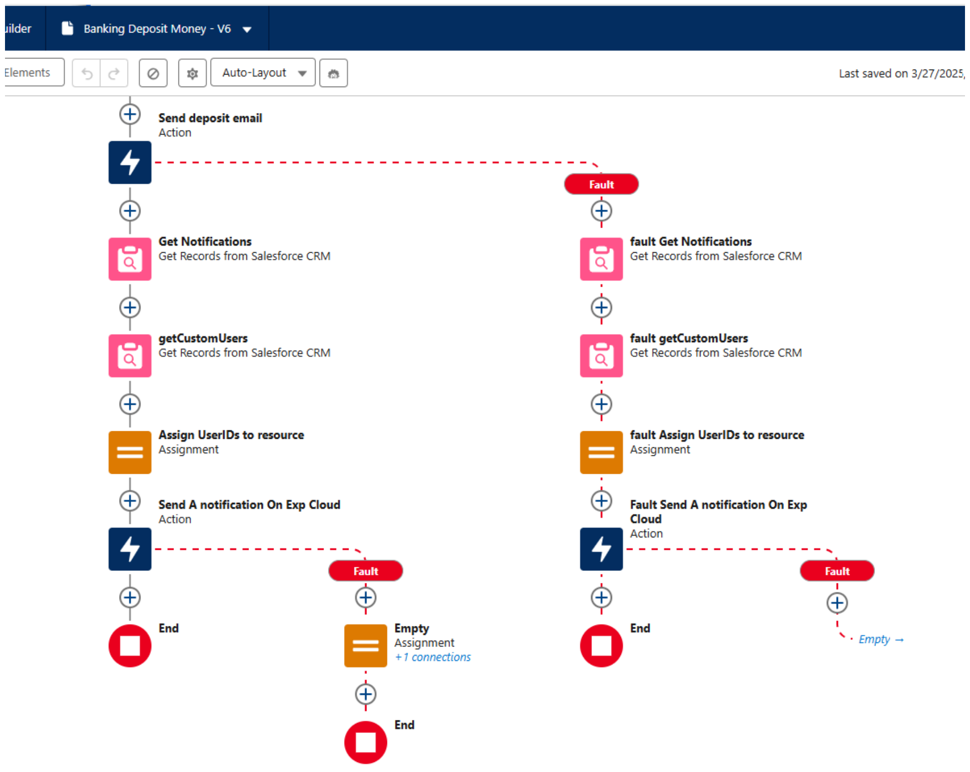
Assign the amount to the account
Assignment

Update amount in account
Update Records

update branch
Update Records

get email templates
Get Records from Salesforce CRM

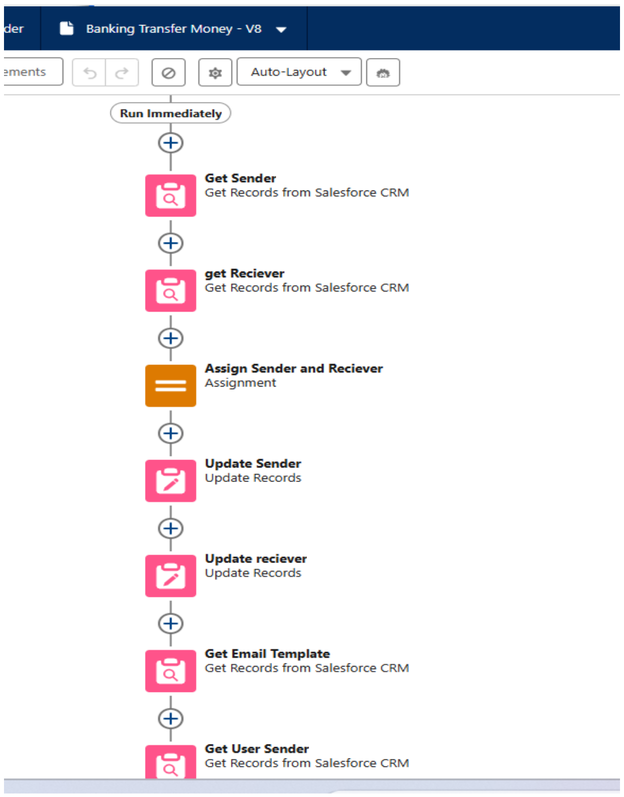
Send deposit email
Action

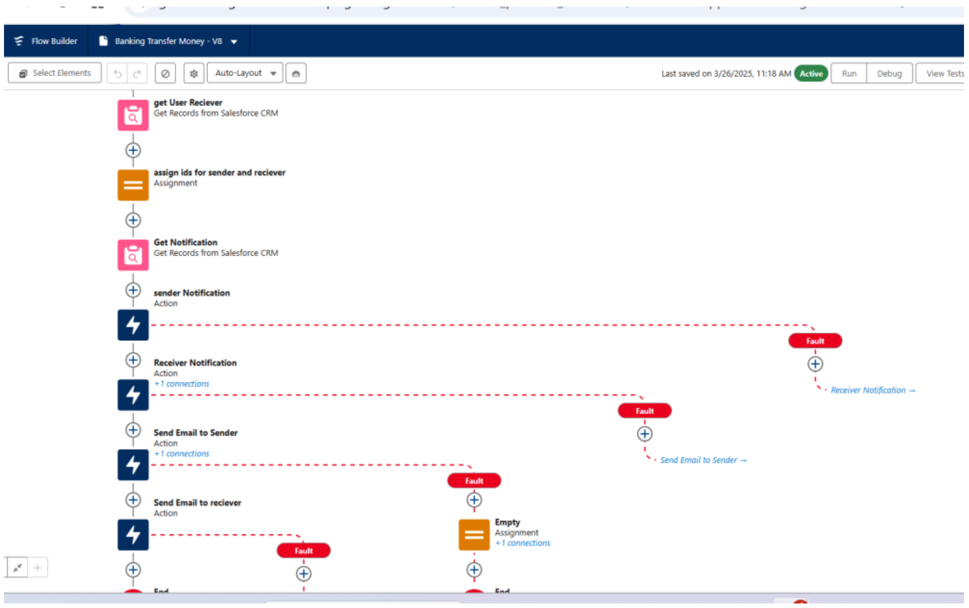

uilder
Banking Deposit Money - V6
Elements
Auto-Layout
Last saved on 3/27/2025,
Send deposit email
Action
Fault
Get Notifications
Get Records from Salesforce CRM
fault Get Notifications
Get Records from Salesforce CRM
getCustomUsers
Get Records from Salesforce CRM
fault getCustomUsers
Get Records from Salesforce CRM
Assign UserIDs to resource
Assignment
fault Assign UserIDs to resource
Assignment
Send A notification On Exp Cloud
Action
Fault Send A notification On Exp Cloud
Action
Fault
Fault
End
Empty
Assignment
+1 connections
End
Empty →
End

❖ Banking Transfer Money

- **Purpose:** This flow is designed to manage the processes triggered by the creation of a Transfer__c record. It ensures that account balances are updated, and that notifications and emails are sent to both the sender and receiver.
- **Description:** This is a Record-Triggered Flow that performs the following actions upon the creation of a Transfer__c record:
  - **Retrieve Sender and Receiver Contacts:**
    - The flow retrieves the sender and receiver contacts based on the related accounts using the Transfer__c - Benefactor_Account__c and Transfer__c - Beneficiary_Account__c fields.
  - **Fetch Associated Users:**
    - Retrieves the respective users associated with these contacts using the User - ContactId field. Retrieve Email Template and Notification

- **Type:**
  - Fetches an email template using the EmailTemplate - Id and EmailTemplate - Name fields.
  - Retrieves a notification type using the CustomNotificationType - Id and CustomNotifTypeName fields.
- **Update Account Balances:**
  - Updates the account balances of the sender and receiver using the Contact - Account_Balance__c field based on the Transfer__c - Transfer_Amount__c field.
- **Send Custom Notifications:**
  - Sends custom notifications to both the sender and receiver using the retrieved notification type.
- **Send Emails:**
  - Sends emails to the sender and receiver using the fetched email template.


der
Banking Transfer Money - V8
ements
Auto-Layout
Run Immediately
Get Sender
Get Records from Salesforce CRM
get Reciever
Get Records from Salesforce CRM
Assign Sender and Reciever
Assignment
Update Sender
Update Records
Update reciever
Update Records
Get Email Template
Get Records from Salesforce CRM
Get User Sender
Get Records from Salesforce CRM



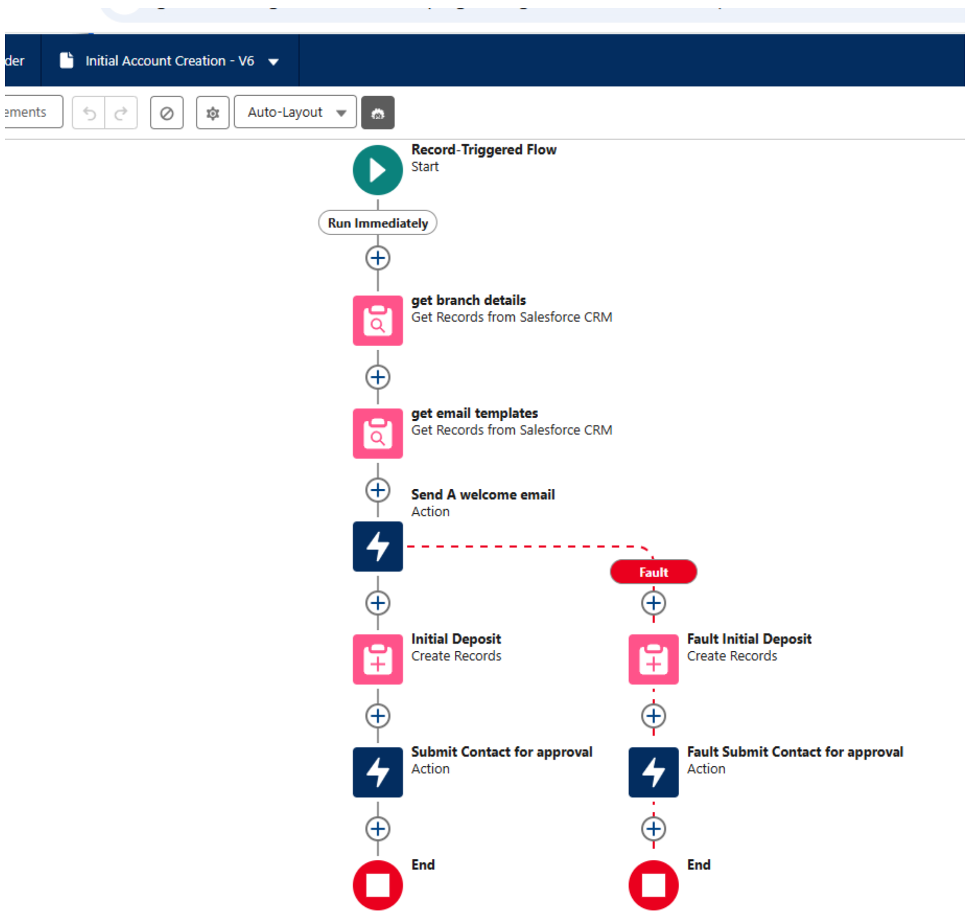
❖ Flow Name: Initial Account Creation

- **Purpose:** This flow is designed to manage the processes triggered by the creation of a new Contact record. It ensures that a welcome email is sent, an initial deposit record is created, and the contact is submitted for KYC approval.
- **Description:** This is a Record-Triggered Flow that performs the following actions upon the creation of a Contact record:
  - **Retrieve Associated Account Details:**
    - The flow retrieves the associated account details using the Contact - AccountId field.
  - **Fetch Email Template:**
    - Fetches the appropriate email template using the EmailTemplate - Name field with the template name 'New Account Creation'.
  - **Send Welcome Email:**
    - Sends a welcome email to the contact using the retrieved email template.
  - **Create Initial Deposit Record:**

- If the email is sent successfully, the flow creates an initial deposit record linked to the contact using the Contact - Id field.

▪ **Submit for KYC Approval:**

- Submits the contact for approval in a process named 'Know Your Customer'.

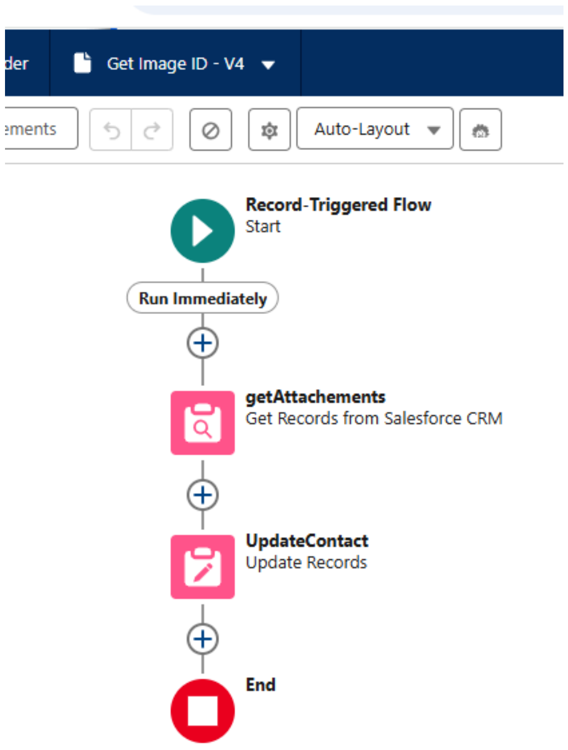


❖ Flow Name: Get Image ID

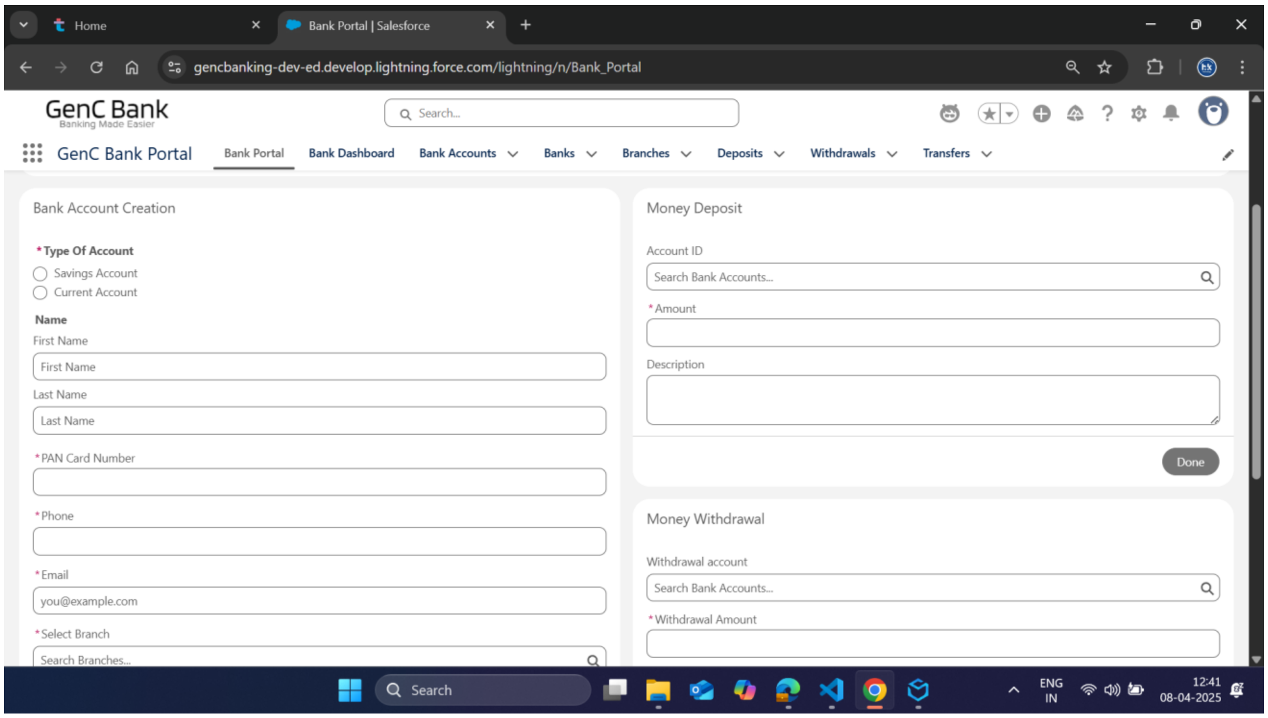
➢ **Purpose:** To automatically update a Contact record with the Attachment ID from related ContentDocumentLink records whenever a Contact record is saved.

➢ **Description:**This flow is triggered when a Contact record is saved. It performs the following actions:

- **Query ContentDocumentLink Records:** The flow queries for ContentDocumentLink records where the LinkedEntityId matches the Id of the triggering Contact record.
- **Update Contact Record:** The flow updates the Contact record with the Attachment ID from the queried ContentDocumentLink record.



Lightning Application



Home | Bank Portal | Salesforce

gencbanking-dev-ed.develop.lightning.force.com/lightning/n/Bank_Portal

GenC Bank — Banking Made Easier

GenC Bank Portal: Bank Portal | Bank Dashboard | Bank Accounts | Banks | Branches | Deposits | Withdrawals | Transfers

Bank Account Creation

*Type Of Account
- Savings Account
- Current Account

Name
First Name: First Name
Last Name: Last Name
*PAN Card Number
*Phone
*Email: you@example.com
*Select Branch: Search Branches...

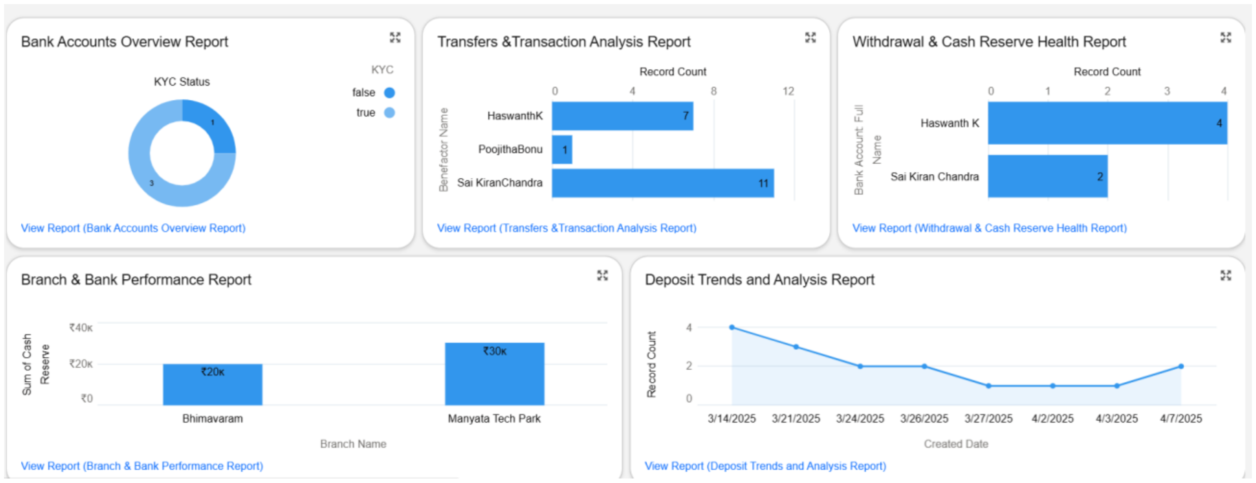
Money Deposit

Account ID: Search Bank Accounts...
*Amount
Description
Done

Money Withdrawal

Withdrawal account: Search Bank Accounts...
*Withdrawal Amount

Reports and Dashboards



Report Overviews Each report includes:

➢ **Report Name & Purpose:** Summary of the report’s goal and target audience.

- **Data Source & Objects:** Primary Salesforce objects used.
- **Key Fields & Formula Fields:** Essential fields for calculations and grouping.
- **Filters & Grouping:** Criteria to limit and organize data.
- **Visualizations:** Suggested charts or graphs.
- **Notes & Best Practices:** Additional tips and dashboard integration details.

2.1 Bank Accounts Overview Report Purpose:

Summarizes account balances, KYC status, account freezes, and provides links to PAN card images.

- **Data Source & Objects:**
    - **Bank Account (Std Contact) Key Fields:**
        - Name, Account_Number__c, Account_Balance__c, KYC__c, Freeze__c, Pan_Card_Image__c, Pan_Card_Number__c
        - **Filters & Grouping:**Date ranges using Created Date. Group by KYC status. \
        - **Visualizations:**Pie chart for KYC status. Bar chart for account balances.
        - **Notes:** Ensure PAN Card links are clickable. Set appropriate permissions for managers. 2.2 Branch & Bank Performance Report Purpose: Compares performance across branches and **banks.**
- **Data Source & Objects:**
    - **Branch (Std Account) Bank (via lookup) Key Fields:**Name, Branch_Address__c, Cash_Reserve__c, Bank_Address__c, Name from Bank
    - **Filters & Grouping:** Group by Bank Name and Branch Name.
    - **Visualizations:** Grouped bar charts. Heat maps.
    - **Notes:**Use join capabilities for multiple objects.

2.3 Deposit Trends and Analysis Report Purpose:

Monitors deposit activity trends over time.

- ❖ **Data Source & Objects:**
    - ➢ **Deposit (Custom) Key Fields:**Name, Amount__c,Bank_Account__c
    - ➢ **Filters & Grouping:**Time-based filters using Created Date. Group by Date.
    - ➢ **Visualizations:**Line charts for deposits over time. Trend lines for average deposit amounts.
    - ➢ **Notes:**Validate deposit amounts meet minimum requirements. Schedule periodic dashboard refreshes.

2.4 Transfers and Transaction Analysis Report Purpose:

Details the flow of funds through transfers.

- ❖ **Data Source & Objects:**
    - ➢ **Transfer (Custom) Key Fields:**Name, Transfer_Amount__c, Benefactor and Beneficiary details
    - ➢ **Filters & Grouping:**Group by Benefactor Name or Beneficiary Name.
    - ➢ **Visualizations:**Bubble or scatter charts. Tables with drill-down links.
    - ➢ **Notes:**Ensure correct concatenation of names. Add filters for high-value transfers.

2.5 Withdrawal & Cash Reserve Health Report Purpose:

Assesses withdrawal transactions against account balances and cash reserves.

- ❖ **Data Source & Objects:**
    - ➢ **Withdrawal (Custom) Key Fields:**Name, Withdrawal_Amount__c, Bank_Account__c, Account_Balance__c
    - ➢ **Filters & Grouping:**Filter for transactions with positive withdrawal amounts. Highlight accounts with high withdrawal amounts.
    - ➢ **Visualizations:**Dual-axis charts for withdrawals and balances. Alerts for at-risk records.

- **Notes:** Review validation rule error messages.

Development

Triggers

- **Trigger Name: ContactTrigger.Trigger**
  - **Description:** The ContactTrigger manages Contact records before insert, before update, and after update events. It calls ContactTriggerHelper.checkSameAddress to synchronize addresses before insert and update. After update, it calls ContactTriggerHelper.CheckMinBalance to check account balances and send email alerts if the balance is below the minimum required amount. This ensures address synchronization and balance checks are handled appropriately.

```
trigger ContactTrigger on Contact (before insert,Before Update,after Update) {
    if(Trigger.isBefore&&Trigger.isInsert){
        ContactTriggerHelper.checkSameAddress(Trigger.New);
    }
    if(Trigger.isBefore&&Trigger.isUpdate){
        ContactTriggerHelper.checkSameAddress(Trigger.New);
    }
    if(Trigger.isAfter&&Trigger.isUpdate){
        ContactTriggerHelper.CheckMinBalance(Trigger.New);
    }


}
```

- **Trigger Name: DepositTrigger.Trigger**
  - **Description:** The DepositTrigger is an Apex trigger that runs after a new Deposit__c record is inserted. When the trigger is activated, it calls the newDeposit method from the DepositHelper class, passing the newly inserted records (Trigger.New) as a parameter. This setup ensures that

any necessary post-insert logic for deposits is handled by the DepositHelper class.

```
trigger DepositTrigger on Deposit__c (after insert) {
    if (Trigger.isAfter && Trigger.isInsert) {
        DepositHelper.newDeposit(Trigger.New);
    }
}
```

- **Trigger Name:TransferTrigger.Trigger**
  - **Description:** The TransferTrigger is an Apex trigger that executes after a new Transfer__c record is inserted. When the trigger is fired, it checks if the operation is an "after insert" event. If true, it calls the newTransfer method from the TransferTriggerHelper class, passing the newly inserted records (Trigger.New) as a parameter. This ensures that any necessary post-insert logic for transfers is handled by the TransferTriggerHelper class.

```
trigger TransferTrigger on Transfer__c (after insert) {
    if (Trigger.isAfter && Trigger.isInsert) {
        TransferTriggerHelper.newTransfer(Trigger.New);
    }
}
```

- **Trigger Name:WithdrawalTrigger.Trigger**
  - **Description:** The WithdrawalTrigger is an Apex trigger that runs after a new Withdrawal__c record is inserted. When the trigger is activated, it checks if the operation is an "after insert" event. If true, it calls the newWithdrawal method from the WithdrawalHelper class, passing the newly inserted records (Trigger.New) as a parameter. This ensures that any necessary post-insert logic for withdrawals is handled by the WithdrawalHelper class.

```
trigger WithdrawalTrigger on Withdrawal__c (after insert) {
    if (Trigger.isAfter && Trigger.isInsert) {
        WithdrawalHelper.newWithdrawal(Trigger.New);
    }
}
```

Trigger Handlers

- ❖ **Handler Name: ContactTriggerHelper.cls**
  - ➢ **Description:** The ContactTriggerHelper class is designed to manage specific business logic for Contact records in Salesforce. It includes two main methods: checkSameAddress and CheckMinBalance. The checkSameAddress method ensures that if a contact's current address is marked as the same as their permanent address, the current address field is updated accordingly. The CheckMinBalance method checks if a contact's account balance falls below a specified minimum (500) and if the contact's KYC is complete. If both conditions are met, it sends an email alert to the contact, notifying them of their low balance. This method also includes error handling to log any issues that occur during the email sending process.

```
public class ContactTriggerHelper {
    private static final Decimal MIN_BALANCE = 500;
    public static void checkSameAddress(list<contact>conList){
        for(contact con:conList){
            if(con.Current_Address_Same_as_Permanent__c==True){
                con.Current_Address__c=con.Permanent_Address__c;
            }
        }
    }
    public static void CheckMinBalance(list<contact>conList){
       List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();
        for(Contact con:conList){
            if (con.Email != null && con.Account_Balance__c <
MIN_BALANCE&&con.KYC__c==True) {
            Messaging.SingleEmailMessage email = new
Messaging.SingleEmailMessage();
            email.setToAddresses(new List<String>{con.Email});
            email.setSubject('Low Account Balance Alert');
            email.setPlainTextBody('Dear ' + con.FirstName + ',\n\n' +
                                   'Your account balance
(Rs.'+con.Account_Balance__c+') has dropped below the minimum required balance of
500.\n' +
                                   'Please ensure to maintain the required
balance to avoid any inconvenience.\n\n' +
                                   'Best regards,\n' +
                                   'GenC Bank');
            emails.add(email);
        }

        }
         try {
             if (!emails.isEmpty()) {
                 Messaging.sendEmail(emails);
             }
         }
        catch (Exception e) {

        System.debug('Error sending emails: ' + e.getMessage());
    }
    }


}
```

- ❖ **Handler Name: DepositHandler.cls**
  - ➢ **Description:** The DepositHelper class handles the creation of new transaction records based on deposit records. The newDeposit method processes a list of Deposit__c records, validating that each deposit has a non-null amount and associated bank account. For each valid deposit, it creates a corresponding Transaction__c record, setting the name, amount, and receiver account fields. These transaction records are collected in a list and inserted into the database if the list is not empty. The method includes error handling to log any DML exceptions that occur during the insertion process.

```
public class DepositHelper {
    public static void newDeposit(List<Deposit__c> depositList) {
        List<Transaction__c> newTransList = new List<Transaction__c>();

        for (Deposit__c deposit : depositList) {
            // Validate required fields
            if (deposit.Amount__c != null && deposit.Bank_Account__c != null) {
                Transaction__c newTrans = new Transaction__c();
                newTrans.Name = deposit.Name;
                newTrans.Amount__c = deposit.Amount__c;
                newTrans.Receiver_Account__c = deposit.Bank_Account__c;
                newTransList.add(newTrans);
            }
        }

        // Insert only if the list is not empty
        if (!newTransList.isEmpty()) {
            try {
                insert newTransList;
            } catch (DmlException e) {
                // Log error and handle gracefully
                System.debug('DML Exception occurred: ' + e.getMessage());
            }
        }
    }
}
```

- ❖ **Handler Name: TransferTriggerHelper.Cls**
  - ➢ **Description:** The TransferTriggerHelper class manages the creation of transaction records based on transfer records. The newTransfer method processes a list of Transfer__c records, validating that each transfer has a non-null transfer amount, benefactor account, and beneficiary account. For each valid transfer, it creates a corresponding Transaction__c record, setting the name, amount, sender account, and receiver account fields. These transaction records are collected in a list and inserted into the database if the list is not empty. The method includes error handling to log any DML exceptions that occur during the insertion process.

```
public class TransferTriggerHelper {
    public static void newTransfer(List<Transfer__c> transferList) {
        List<Transaction__c> newTransList = new List<Transaction__c>();
        for (Transfer__c transfer : transferList) {
            if (transfer.Transfer_Amount__c != null &&
                transfer.Benefactor_Account__c != null &&
                transfer.Beneficiary_Account__c != null) {
```

```
                Transaction__c newTrans = new Transaction__c();
                newTrans.name=transfer.name;
                newTrans.Amount__c = transfer.Transfer_Amount__c;
                newTrans.Sender_Account__c = transfer.Benefactor_Account__c;
                newTrans.Receiver_Account__c = transfer.Beneficiary_Account__c;
                newTransList.add(newTrans);
            }
        }
        if (!newTransList.isEmpty()) {
            try {
                insert newTransList;
            } catch (DmlException e) {

                System.debug('DML Exception: ' + e.getMessage());
            }
        }
    }
}
```

❖ **Handler Name: WithdrawalHelper.cls**

➢ **Description:** The WithdrawalHelper class handles the creation of transaction records based on withdrawal records. The newWithdrawal method processes a list of Withdrawal__c records, validating that each withdrawal has a non-null bank account and withdrawal amount. For each valid withdrawal, it creates a corresponding Transaction__c record, setting the name, amount, and sender account fields. These transaction records are collected in a list and inserted into the database if the list is not empty. The method includes error handling to log any DML exceptions that occur during the insertion process.

```
public class WithdrawalHelper {
    public static void newWithdrawal(List<Withdrawal__c> withdrawalList) {
        List<Transaction__c> newTransList = new List<Transaction__c>();

        for (Withdrawal__c withdrawal : withdrawalList) {
            // Validate required fields
            if (withdrawal.Bank_Account__c != null &&
withdrawal.Withdrawal_Amount__c != null) {
```

```
                Transaction__c newTrans = new Transaction__c();
                newTrans.Name=Withdrawal.Name;
                newTrans.Amount__c = withdrawal.Withdrawal_Amount__c;
                newTrans.Sender_Account__c = withdrawal.Bank_Account__c;
                newTransList.add(newTrans);
            }
        }

        // Insert only if list is not empty
        if (!newTransList.isEmpty()) {
            try {
                insert newTransList;
            } catch (DmlException e) {
                // Log error and handle gracefully
                System.debug('Error inserting transactions: ' + e.getMessage());
            }
        }
    }
}
```

Lightning Web Components

- ❖ **LWC Name: Account Balance**
    - ➢ **AccountBalance.html:**
    - ➢ **Description:** This template is designed for displaying an account balance in a user interface. It includes:
        - ▪ **Heading:** Displays "Account Balance" as the title.
        - ▪ **Account Balance Value:** Shows the account balance if available, formatted with the currency symbol ₹.
        - ▪ **Error Message:** Displays an error message if there is an issue retrieving the account balance.
        - ▪ **Refresh Button:** A button to refresh the account balance, with an SVG icon indicating the refresh action.
    - ➢ The template uses conditional rendering to display the account balance or error message based on the presence of accountBalance and error variables. The refresh button triggers the handleRefresh function when clicked.

```
<template>
    <div class="account-balance-container">
        <!-- Heading -->
        <h2 class="account-balance-heading">Account Balance</h2>

        <!-- Account Balance Value -->
        <template if:true={accountBalance}>
            <p class="account-balance-value">₹{accountBalance}</p>
        </template>

        <!-- Error Message -->
        <template if:true={error}>
            <p class="error-message">{error}</p>
        </template>

        <!-- Refresh Button -->
        <button class="refresh-button" onclick={handleRefresh}>
    <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="#ffffff"
viewBox="0 0 24 24">
        <path d="M17.65 6.35A7.958 7.958 0 0 0 12 4c-4.41 0-8 3.59-8 8h2.05A5.978
5.978 0 0 1 12 6c1.63 0 3.12.66 4.24 1.76L13 11h7V4l-2.35 2.35zM18 12a6.002 6.002
0 0 1-6 6c-1.63 0-3.12-.66-4.24-1.76L11 13H4v7l2.35-2.35A7.958 7.958 0 0 0 12
20c4.41 0 8-3.59 8-8h-2z"/>
    </svg>
</button>
    </div>

</template>
```

- **AccountBalance.Css:**
- **Description:** This CSS code styles the account balance container and its elements:
    - **Container:** Centers the container with a white background, light grey border, rounded corners, and a subtle shadow. It uses Arial font and centers text.
    - **Heading:** Styles the heading with a bold, dark grey font.

    - Account Balance: Displays the account balance in a large, bold, medium grey font.
    - **Error Message:** Shows error messages in a bold, muted red font.
    - **Refresh Button:** Styles the button with a dark grey background, white text, rounded shape, and shadow. It includes hover and active states for visual feedback and an optional spinning effect for the SVG icon on hover.

- This setup ensures a clean, user-friendly interface for displaying account balances and handling errors

```css
/* Container Styles */
.account-balance-container {
    max-width: 400px;
    margin: auto;
    background-color: #ffffff; /* White background */
    border: 1px solid #dcdcdc; /* Light grey border */
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    padding: 20px;
    position: relative;
    text-align: center;
    font-family: 'Arial', sans-serif;
}

/* Heading Styles */
.account-balance-heading {
    font-size: 24px;
    font-weight: bold;
    color: #333333; /* Dark grey text */
    margin-bottom: 10px;
}

/* Account Balance Styles */
.account-balance-value {
    font-size: 36px;
    font-weight: bold;
    color: #666666; /* Medium grey text */
}

/* Error Message Styles */
```

```css
.error-message {
    font-size: 18px;
    color: #cc3333; /* Muted red for errors */
    font-weight: bold;
}

/* Button Styles */
.refresh-button {
    background-color: #333333; /* Dark grey button */
    color: #ffffff; /* White text */
    font-size: 16px;
    font-weight: bold;
    border: none;
    border-radius: 50%;
    padding: 12px;
    cursor: pointer;
    position: absolute;
    bottom: 10px;
    right: 10px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
    display: flex;
    align-items: center;
    justify-content: center;
    transition: all 0.3s ease-in-out;
}

/* Button Hover and Active States */
.refresh-button:hover {
    background-color: #555555; /* Medium grey button on hover */
}

.refresh-button:active {
    transform: scale(0.9);
    background-color: #222222; /* Slightly darker grey on click */
    transition: transform 0.1s ease-in-out, background-color 0.1s ease-in-out;
}

/* Icon Animation (Optional) */
.refresh-button svg {
    transition: transform 0.3s ease-in-out;
}
```

```
.refresh-button:hover svg {
    transform: rotate(360deg); /* Adds spinning effect */
}
```

- **AccountBalance.js:**
- **Description:** The AccountBalance Lightning Web Component (LWC) fetches and displays the account balance, handles errors, and publishes the balance to a message channel. Here's a brief description:
    - **Imports:** Includes necessary modules for LWC, Apex method, and Lightning Message Service.
    - **Properties:** Tracks accountBalance and error.
    - **Message Context:** Wires the message context for publishing.
    - **connectedCallback:** Calls refreshBalance when the component is initialized.
    - **refreshBalance:** Fetches the account balance using the Apex method, sets the balance, clears errors, and publishes the balance. Handles errors by setting the error message and logging it.
    - **handleRefresh:** Refreshes the account balance when the refresh button is clicked.
- This setup ensures the component dynamically updates and communicates the account balance while handling any errors gracefully.

```
import { LightningElement,track,wire } from 'lwc';
import getAccountBalance from
'@salesforce/apex/AccountBalanceController.getAccountBalance';
import {publish,MessageContext} from 'lightning/messageService';
import ContactForAccBal from '@salesforce/messageChannel/ContactForAccBal__c';

export default class AccountBalance extends LightningElement {
    @track accountBalance;
    @track error;
    @wire(MessageContext)
    messageContext;

    connectedCallback(){
        this.refreshBalance();
    }

    refreshBalance(){

      getAccountBalance()
        .then(result => {
            this.accountBalance = result; //set the fetched account balance
            this.error=undefined;//clear any errors
            const payload={accBalToSend:this.accountBalance};
            publish(this.messageContext,ContactForAccBal,payload);
        })
        .catch(error => {
            this.error = error;
            this.accountBalance = undefined;
            this.error=error.body.message||'An error occured while fetching the
account balance';
            console.log('Error',error);

        });

    }
    handleRefresh(){
        this.refreshBalance();
    }
}
```

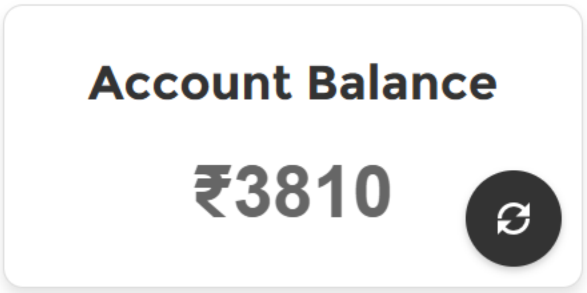
- **AccountBalance.js-meta.xml:**
- **Description:** This XML configuration file defines the metadata for a Lightning Web Component (LWC) bundle.
    - **apiVersion:** Specifies the API version (63.0) for the component.
    - **isExposed:** Indicates that the component is exposed and can be used in various contexts.
    - **targets:** Lists the different pages and layouts where the component can be used, including community pages, record pages, app pages, and home pages.
- This setup ensures the component is accessible and usable across multiple Salesforce environments.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>63.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightningCommunity__Page</target>
        <target>lightningCommunity__Page_Layout</target>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

- **AccountBalanceController.Cls:**
- **Description:** The AccountBalanceController class provides a method to retrieve the account balance for the current user.
    - **Class Declaration:** The class is declared as public with sharing to enforce sharing rules.
    - **Method:** getAccountBalance is an @AuraEnabled method, allowing it to be called from Lightning components.
    - **Logic:** Queries the User object to get the contactId for the current user. Extracts the contactId values into a list. Uses this list to query the Contact object for the Account_Balance__c field.
- Returns the account balance if found, or 0 if no contact is associated. This setup ensures that the account balance is retrieved securely and efficiently for the logged-in user.

```apex
public with sharing class AccountBalanceController {
    @AuraEnabled(cacheable=false)
    public static Decimal getAccountBalance() {
        // Query the User object to retrieve the contactId field
        List<User> users = [SELECT contactId FROM User WHERE Id =
:UserInfo.getUserId()];

        // Extract the contactId values into a List<Id>
        List<Id> conIds = new List<Id>();
        for (User user : users) {
            conIds.add(user.contactId);
        }

        // Use the List<Id> to query the Contact object
        List<Contact> conts = [SELECT Account_Balance__c FROM Contact WHERE Id IN
:conIds];

        // Return the Account_Balance__c value
        return (conts.isEmpty()) ? 0 : conts[0].Account_Balance__c;
    }
}
```

➢ **Output:**



❖ **LWC Name: Currency Converter**

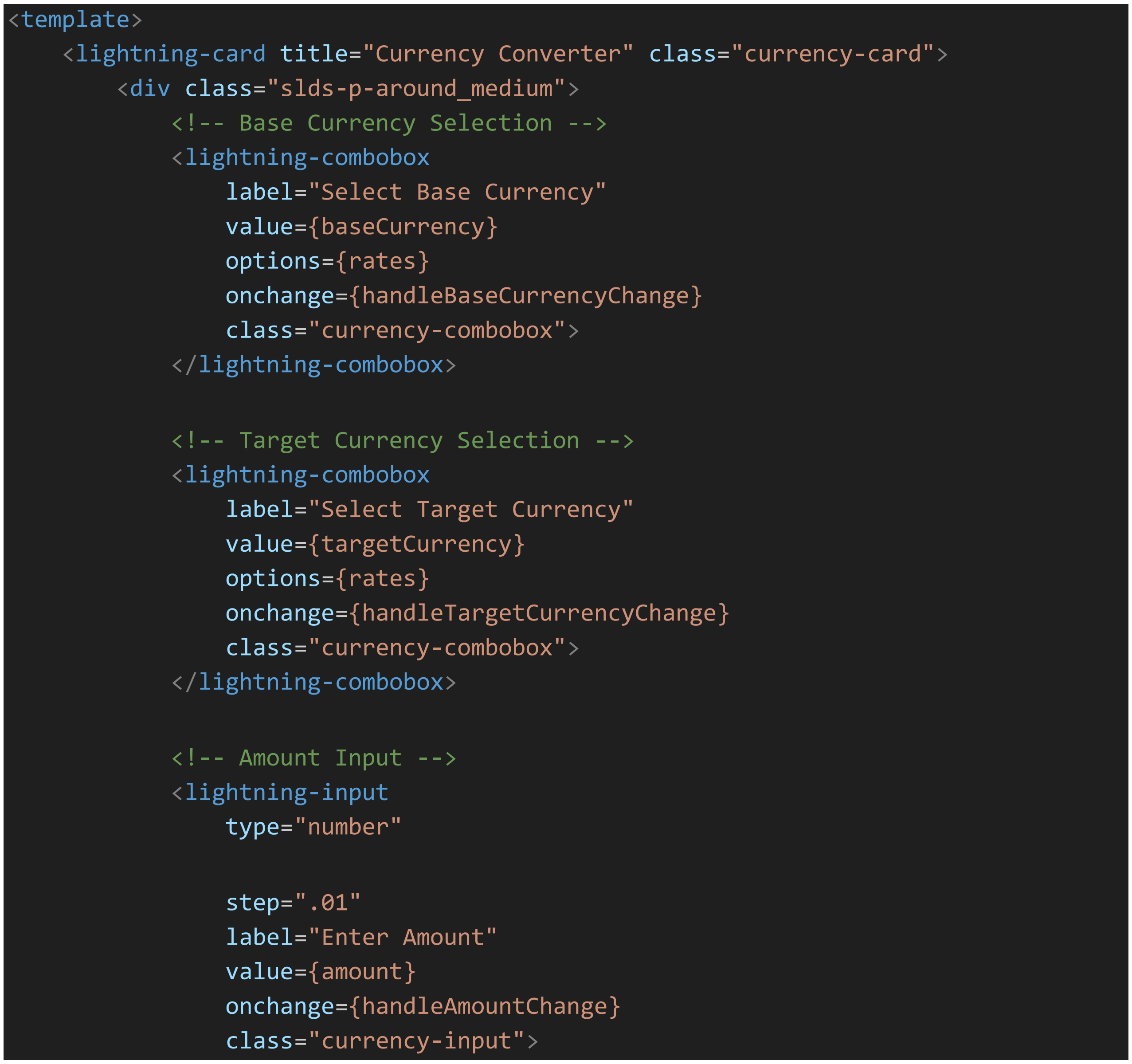
➢ **CurrencyConverter.html:**

➢ **Description:** This template is designed for a currency converter component in Salesforce Lightning. It includes:

- **Lightning Card:** A container with the title "Currency Converter."
- **Base Currency Selection:** A combobox for selecting the base currency, with options populated from rates and an onchange event handler.
- **Target Currency Selection:** A combobox for selecting the target currency, also with options from rates and an onchange event handler.

- **Amount Input:** An input field for entering the amount to be converted, with a step value of 0.01 and an onchange event handler.
- **Convert Button:** A button to trigger the conversion process, with an onclick event handler.
- **Result Box:** Displays the converted amount and target currency if convertedAmount is available.

➢ This setup provides a user-friendly interface for converting currencies, handling user input, and displaying the results.



```
<template>
    <lightning-card title="Currency Converter" class="currency-card">
        <div class="slds-p-around_medium">
            <!-- Base Currency Selection -->
            <lightning-combobox
                label="Select Base Currency"
                value={baseCurrency}
                options={rates}
                onchange={handleBaseCurrencyChange}
                class="currency-combobox">
            </lightning-combobox>

            <!-- Target Currency Selection -->
            <lightning-combobox
                label="Select Target Currency"
                value={targetCurrency}
                options={rates}
                onchange={handleTargetCurrencyChange}
                class="currency-combobox">
            </lightning-combobox>

            <!-- Amount Input -->
            <lightning-input
                type="number"

                step=".01"
                label="Enter Amount"
                value={amount}
                onchange={handleAmountChange}
                class="currency-input">
            </lightning-input>

            <!-- Convert Button -->
            <lightning-button
                label="Convert"
                variant="brand"
                class="convert-button"
                onclick={handleConvert}>
            </lightning-button>

            <!-- Result Box for Converted Amount -->
            <template if:true={convertedAmount}>
                <div class="result-box">
                    <p class="result-text"><strong>{convertedAmount}
{targetCurrency}</strong></p>
                </div>
            </template>
        </div>
    </lightning-card>
</template>
```

- **CurrencyConverter.css:**
- **Description:** This CSS code styles the currency converter component, ensuring a clean and responsive design:
    - **General Card Styling:** Centers the card with a white background, rounded corners, and a light grey border. It includes a subtle shadow and text alignment.
    - **Spacing and Layout Adjustments:** Adds padding around the card content.
    - **Combobox and Input Fields:** Sets full width, margin, and rounded corners for comboboxes and input fields.
    - **Convert Button Styling:** Styles the button with a grey background, white text, rounded corners, and a hover effect that changes the background to black.
    - **Result Box Styling:** Styles the result box with padding, a light background, rounded corners, and a shadow. The result text is bold and centered.

- **Responsive Adjustments:** Ensures the card and converted amount text adjust for smaller screens.

➢ This setup provides a user-friendly and visually appealing interface for the currency converter.

```css
/* General Card Styling */
.currency-card {
    background-color: #ffffff;
    border-radius: 15px;
    padding: 0px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0);
    max-width: 380px;
    margin: auto;
    text-align: center;
    border: 1px solid #e4e7ea;
}

/* Spacing and Layout adjustments */
.currency-card .slds-p-around_medium {
    padding: 20px;
}

/* Combobox and Input Fields */
lightning-combobox, lightning-input {
    width: 100%;
    margin-bottom: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0);
}

lightning-combobox .slds-select {
    padding: 10px;
}

/* Convert Button Styling */
.convert-button {
    width: 100%;
    padding: 0px;
    background-color: grey;
    color: white;
    font-weight: 600;
    border-radius: 8px;
    text-align: center;
    border: none;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.convert-button:hover {
    background-color:black;

}

/* Result Box Styling */
.result-box {
    margin-top: 20px;
    padding: 20px;
    background-color: #f4f7fa;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
    border: 1px solid #e4e7ea;
}

.result-text {
    font-size: 22px;
    color: #333;
    font-weight: 600;
    text-align: center;
}

/* Responsive adjustments */
@media (max-width: 480px) {
    .currency-card {
        padding: 0px;
    }

    .converted-amount {
        font-size: 18px;
        padding: 12px;
    }
}
```

- **CurrencyConverter.js:**
- **Description:** The CurrencyConverter Lightning Web Component (LWC) allows users to convert amounts between different currencies.
    - **Imports:** Includes necessary modules for LWC and Apex method.
    - **Properties:** Tracks base currency, target currency, amount, converted amount, available rates, and conversion rates.
    - **connectedCallback:** Fetches currency rates when the component is initialized.
    - **fetchRates:** Calls the Apex method to get conversion rates based on the selected base currency, and updates the rates and conversion rates properties.
    - **handleBaseCurrencyChange:** Updates the base currency and refreshes rates when the base currency changes.
    - **handleTargetCurrencyChange:** Updates the target currency.
    - **handleAmountChange:** Updates the amount to be converted.
    - **handleConvert:** Calculates the converted amount using the conversion rates and updates the converted amount property.
- This setup ensures the component dynamically updates currency rates and performs conversions based on user input.

```
import { LightningElement, track } from 'lwc';
import getRates from '@salesforce/apex/CurrencyConverter.getRates';

export default class CurrencyConverter extends LightningElement {
    @track baseCurrency = 'INR';
    @track targetCurrency = 'USD';
    @track amount = 1;
    @track convertedAmount;
    @track rates = [];
    @track conversionRates = {};

    connectedCallback() {
        this.fetchRates();
    }

    fetchRates() {
        getRates({ baseCurrency: this.baseCurrency })
```

```
            .then(result => {
                if (result) {
                    this.rates = Object.keys(result).map(currency => ({ label:
currency, value: currency }));
                    this.conversionRates = result;
                }
            })
            .catch(error => {
                console.error('Error fetching rates:', JSON.stringify(error));
            });
    }

    handleBaseCurrencyChange(event) {
        this.baseCurrency = event.target.value;
        this.fetchRates(); // Refresh rates when base currency changes
    }

    handleTargetCurrencyChange(event) {
        this.targetCurrency = event.target.value;
    }

    handleAmountChange(event) {
        this.amount = event.target.value;
    }

    handleConvert() {
        if (this.conversionRates[this.targetCurrency]) {
            this.convertedAmount = (this.amount *
this.conversionRates[this.targetCurrency]).toFixed(2);
        } else {
            this.convertedAmount = 'Error';
        }
    }
}
```

- **CurrencyConverter.js-meta.xml:**
- **Description:** This XML configuration file defines the metadata for a Lightning Web Component (LWC) bundle.
  - **apiVersion:** Specifies the API version (63.0) for the component.

- **isExposed:** Indicates that the component is exposed and can be used in various contexts.
- **targets:** Lists the different pages and layouts where the component can be used, including community pages, record pages, app pages, and home pages.

➢ This setup ensures the component is accessible and usable across multiple Salesforce environments.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>63.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightningCommunity__Page</target>
        <target>lightningCommunity__Page_Layout</target>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

➢ **CurrencyConverter.cls:**

➢ **Description: The CurrencyConverter class provides a method to fetch currency conversion rates from an external API. Here's a brief description:**

➢

➢ Class Declaration: The class is declared as public without sharing to bypass sharing rules.

➢ Method: getRates is an @AuraEnabled method, allowing it to be called from Lightning components and caching the results.

➢ Logic:

➢ Constructs the API endpoint URL using the provided base currency.

➢ Sends an HTTP GET request to the API.

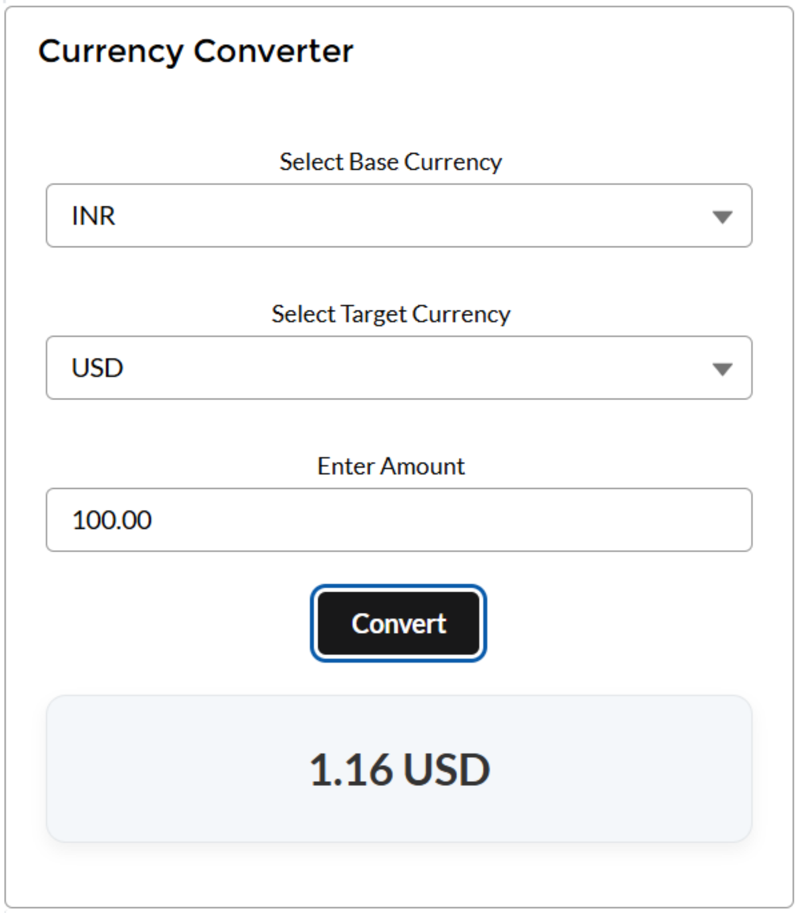
➢ Parses the JSON response to extract conversion rates.

➢ Converts the rates to a Map<String, Decimal> and returns it.

- Handles errors by throwing CalloutException with appropriate messages.
- This setup ensures that the component can retrieve and use up-to-date currency conversion rates.

```
public without sharing class CurrencyConverter {
    @AuraEnabled(cacheable=true)
    public static Map<String, Decimal> getRates(String baseCurrency) {
        String endpoint = 'https://v6.exchangerate-
api.com/v6/4c2436ae479598d735d50db4/latest/' + baseCurrency;

        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(endpoint);
        request.setMethod('GET');
        request.setTimeout(10000);

        HttpResponse response = http.send(request);
        System.debug('API Response: ' + response.getBody());

        if (response.getStatusCode() == 200) {
            try {
                Map<String, Object> jsonResponse = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());

                if (jsonResponse.containsKey('conversion_rates')) {
                    Map<String, Object> rawRates = (Map<String, Object>)
jsonResponse.get('conversion_rates');
                    Map<String, Decimal> conversionRates = new Map<String,
Decimal>();

                    for (String key : rawRates.keySet()) {
                        conversionRates.put(key,
Decimal.valueOf(String.valueOf(rawRates.get(key))));
                    }

                    return conversionRates;
                } else {
                    throw new CalloutException('Error: Missing conversion_rates
in API response.');
                }
            } catch (Exception e) {
                throw new CalloutException('Error parsing response: ' +
e.getMessage());
            }
        } else {
            throw new CalloutException('API Error: ' + response.getStatus());
        }
    }
}
```

Output:


Currency Converter

Select Base Currency: INR

Select Target Currency: USD

Enter Amount: 100.00

1.16 USD

❖ LWC Name: Savings Goal Tracker

- ➢ **SavingsGoalTracker.html**
- ➢ **Description:** This template is designed for a savings goal tracker component in Salesforce Lightning. It includes:

- **Lightning Card:** A container with the title "Savings Goal Tracker."
- **Current Balance and Savings Goal Section:** Displays the current balance and savings goal, with an edit icon to open a modal for changing the goal.
- **Horizontal Progress Bar:** Shows the progress towards the savings goal with a dynamic progress bar and percentage.
- **Modal for Changing Goal:** A modal dialog for setting a new savings goal, including input fields and buttons for saving or canceling the changes.

➢ This setup provides a user-friendly interface for tracking savings goals and updating them as needed.

```
<template>
    <lightning-card title="Savings Goal Tracker">
        <div class="slds-p-around_medium">

            <!-- Current Balance and Savings Goal Section -->
            <div class="slds-grid slds-grid_align-spread slds-p-bottom_small">
                <!-- Current Balance -->
                <p><strong>Current Balance: ₹{PresentAccBal}</strong></p>

                <!-- Savings Goal Section with Edit Icon -->
                <p>
                    <strong>Goal: ₹{savingsGoal}</strong>
                    <lightning-icon
                        icon-name="utility:edit"
                        alternative-text="Edit Savings Goal"
                        class="edit-icon"
                        onclick={openModal}
                    ></lightning-icon>
                </p>
            </div>

            <!-- Horizontal Progress Bar -->

            <div class="progress-container">
                <div class="progress-bar" style={progressStyle}></div>
                <span class="progress-percent">{progress}%</span>
            </div>



        </div>
    </lightning-card>

    <!-- Modal for Changing Goal -->
    <template if:true={isModalOpen}>
        <section role="dialog" class="slds-modal slds-fade-in-open">
            <div class="slds-modal__container">
                <header class="slds-modal__header">
                    <h2 class="slds-text-heading_medium">Set Savings Goal</h2>
                </header>
                <div class="slds-modal__content slds-p-around_medium">
                    <lightning-input
                        type="number"
                        label="Enter New Goal"
                        value={tempGoal}
                        onchange={handleGoalChange}
                    ></lightning-input>
                </div>
                <footer class="slds-modal__footer">
                    <lightning-button label="Cancel"
onclick={closeModal}></lightning-button>
                    <lightning-button label="Save" variant="brand"
onclick={saveGoal}></lightning-button>
                </footer>
            </div>
        </section>
        <div class="slds-backdrop slds-backdrop_open"></div>
    </template>
</template>
```

- **SavingsGoalTracker.css:**
- **Description:** This CSS code styles the savings goal tracker component, ensuring a clean and responsive design:
    - **Global Styles:** Sets the default font family and color.
    - **Card Height Adjustment:** Reduces padding for a more compact card.

- **Progress Bar Styles:** Styles the progress bar container and bar, including hover effects to display the percentage.
- **Modal Container Styling:** Styles the modal container with rounded corners and a shadow.
- **Button Styles:** Styles buttons with dark grey background, white text, rounded corners, and hover/active states for visual feedback.
- **Edit Icon Styles:** Styles the edit icon with color transitions and hover/active effects for a smooth user experience.

➢ This setup provides a user-friendly and visually appealing interface for tracking and updating savings goals.

```css
/* Global Styles */
:host {
  font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;
  color: #333;
}

/* Card Height Adjustment */
.reduced-height-card {
    padding: 8px !important; /* Smaller padding */
}



/* Progress Bar Styles */
.progress-container {
    width: 100%;
    background-color: #f0f0f0;
    border-radius: 10px;
    height: 15px; /* Adjust height */
    margin: 8px 0; /* Reduced margin */
    overflow: hidden;
    position: relative;
}

.progress-bar {
    height: 100%;
    background-color: #28a745;
    border-radius: 10px;
    transition: width 0.5s ease-in-out;
    position: relative;
}

.progress-percent {
    display: none; /* Initially hidden */
    position: absolute;
  font-size: smaller;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    color: #ffffff;
    font-weight: bold;
}

/* Show percentage on hover */
.progress-container:hover .progress-percent {
    display: block;
}

/* Modal Container Styling */
.slds-modal__container {
  max-width: 400px;
  border-radius: 8px;
  overflow: hidden;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0);
}

/* Button Styles */
.button {
  background-color: #333333; /* Dark grey for normal state */
  color: #ffffff; /* White text */
  font-size: 16px;
  font-weight: bold;
  padding: 12px 24px;
  border: none;
  border-radius: 4px; /* Subtle rounding for edges */
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); /* Soft shadow for depth */
}

/* Hover State */
```

```css
.button:hover {
  background-color: #555555; /* Medium grey for hover */
}

/* Active State */
.button:active {
  background-color: #222222; /* Slightly darker grey for click */
  transform: scale(0.95); /* Subtle press effect */
}

/* Edit Icon Styles */
.edit-icon {
  margin-left: 4px;
  color: #28a745;
  cursor: pointer;
  width: 12px;  /* Adjust width as needed */
  height: 12px; /* Adjust height as needed */
  transition: transform 0.5s cubic-bezier(0.4, 0.0, 0.2, 1), color 0.5s ease;
  transform: scale(0.5); /* Slightly larger initial size for smoother transitions
*/
}

.edit-icon:hover {
  color: #555555; /* Grey color for hover state */
  transform: scale(0.7); /* Subtle zoom effect */
}

.edit-icon:active {
  color: #222222; /* Slightly darker grey for click state */
  transform: scale(0.60); /* Gentle press effect */
  transition: transform 0.2s cubic-bezier(0.6, -0.28, 0.735, 0.045), color 0.2s
ease-out; /* Snappier motion on press */
}
```

- **SavingsGoalTracker.js:**
- **Description:** The SavingsGoalTracker Lightning Web Component (LWC) helps users track their savings goals.
  - **Imports:** Includes necessary modules for LWC and Lightning Message Service.
  - **Properties:** Tracks the current account balance (PresentAccBal), savings goal (savingsGoal), progress towards the goal (progress),

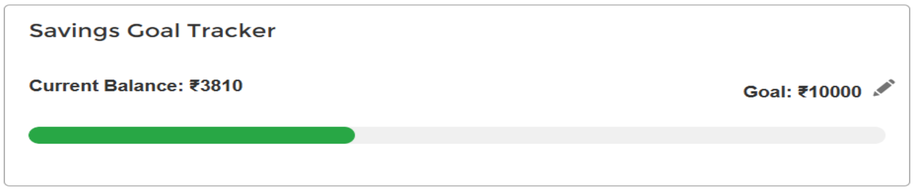
and modal state (isModalOpen). Also includes a temporary goal (tempGoal) for modal input.

- **Message Context:** Wires the message context for subscribing to messages.
- **connectedCallback:** Subscribes to the ContactForAccBal message channel to receive account balance updates and loads the saved goal from localStorage.
- **handleMessage:** Updates the current account balance and recalculates progress when a message is received.
- **calculateProgress:** Calculates the progress towards the savings goal as a percentage.
- **progressStyle:** Returns the width style for the progress bar based on the progress percentage.
- **openModal:** Opens the modal for setting a new savings goal.
- **closeModal:** Closes the modal.
- **handleGoalChange:** Updates the temporary goal based on user input.
- **saveGoal:** Saves the new goal to localStorage, updates the savings goal, recalculates progress, and closes the modal.

➢ This setup ensures the component dynamically updates and tracks savings goals, providing a user-friendly interface for managing financial targets.

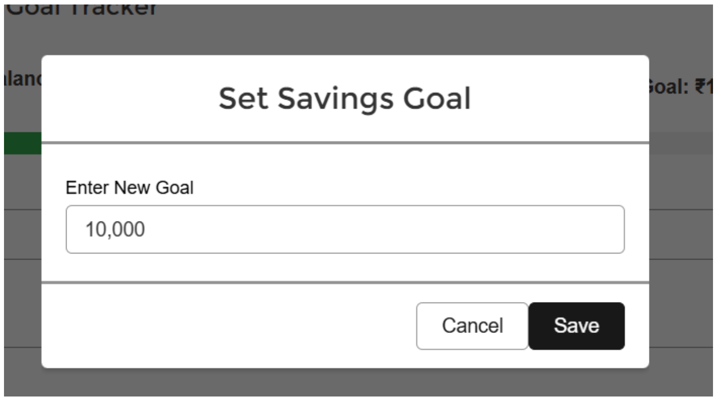
```
import { LightningElement, track, wire } from 'lwc';
import { subscribe, MessageContext } from 'lightning/messageService';
import ContactForAccBal from '@salesforce/messageChannel/ContactForAccBal__c';

export default class SavingsGoalTracker extends LightningElement {
    PresentAccBal = 0;
    @track savingsGoal = 100000; // Default goal
    @track progress = 0;
    @track isModalOpen = false;
    tempGoal;

    @wire(MessageContext)
    messageContext;
    connectedCallback() {
        this.subscription = subscribe(
            this.messageContext,
            ContactForAccBal,
            (message) => {
                this.handleMessage(message);
            }
        );

        // Load saved goal from localStorage
        const savedGoal = localStorage.getItem('savingsGoal');
        if (savedGoal) {
            this.savingsGoal = parseInt(savedGoal, 10);
        }
        this.calculateProgress();
    }

    handleMessage(message) {
        this.PresentAccBal = message.accBalToSend;
        this.calculateProgress();
    }

    calculateProgress() {
        this.progress = Math.min((this.PresentAccBal / this.savingsGoal) * 100,
100);
    }

    get progressStyle() {
        return `width: ${this.progress}%`;
    }

    openModal() {
        this.tempGoal = this.savingsGoal;
        this.isModalOpen = true;
    }

    closeModal() {
        this.isModalOpen = false;
    }

    handleGoalChange(event) {
        this.tempGoal = event.target.value;
    }

    saveGoal() {
        this.savingsGoal = parseInt(this.tempGoal, 10);
        localStorage.setItem('savingsGoal', this.savingsGoal);
        this.calculateProgress();
        this.closeModal();
    }
}
```

➢ **Output:**


Savings Goal Tracker

Current Balance: ₹3810

Goal: ₹10000



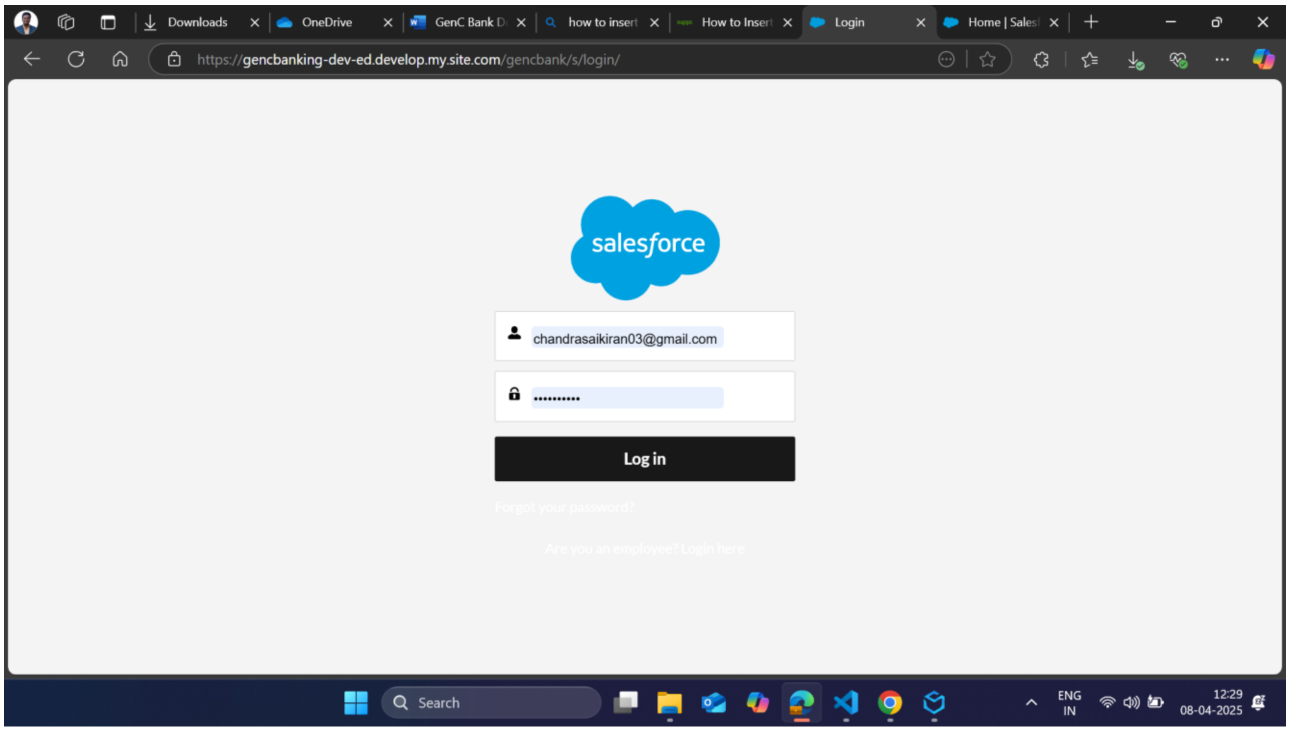
Lightning Messaging Service:

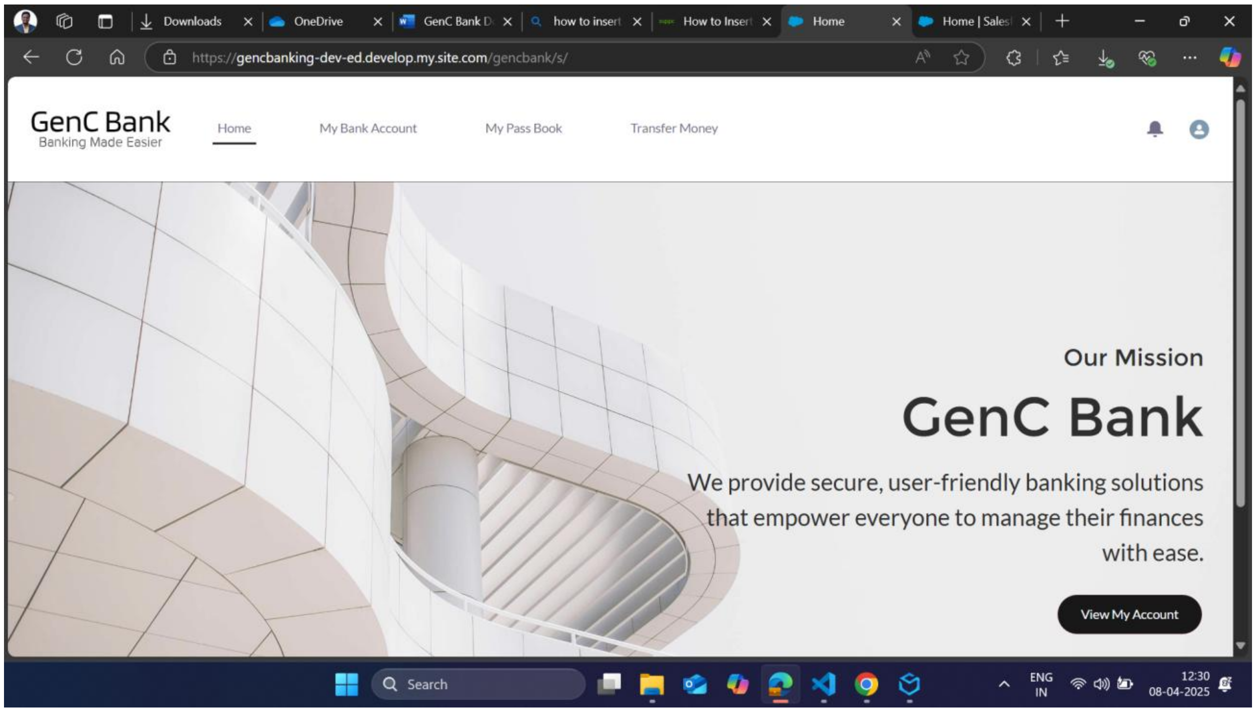
❖ **MessageChannel:**

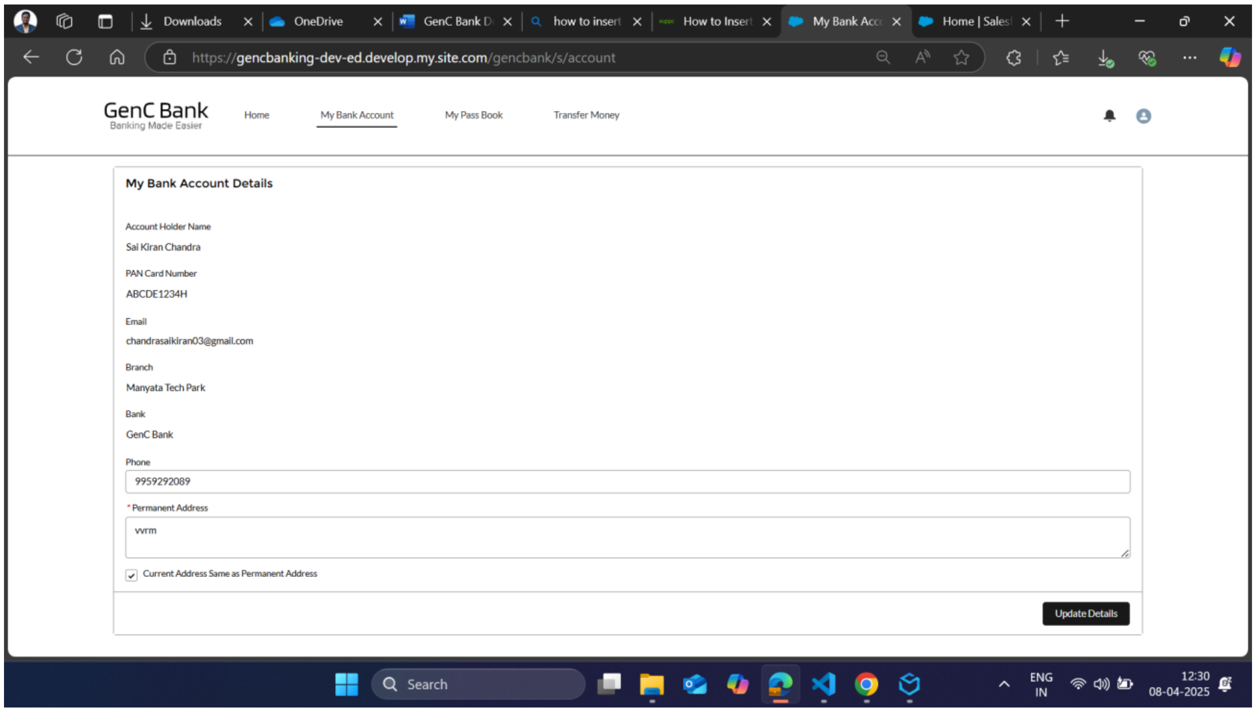
```
<?xml version="1.0" encoding="UTF-8"?>
<LightningMessageChannel xmlns="http://soap.sforce.com/2006/04/metadata">
    <masterLabel>ContactForAccBal</masterLabel>
    <isExposed>true</isExposed>
    <lightningMessageFields>
```

```xml
        <fieldName>accBalToSend</fieldName>
        <description>account balance needed to be sent</description>
    </lightningMessageFields>
</LightningMessageChannel>
```

Experience Cloud:


https://gencbanking-dev-ed.develop.my.site.com/gencbank/s/login/
salesforce
chandrasaikiran03@gmail.com
Log in
Forgot your password?
Are you an employee? Login here


https://gencbanking-dev-ed.develop.my.site.com/gencbank/s/
GenC Bank
Banking Made Easier
Home
My Bank Account
My Pass Book
Transfer Money
Our Mission
GenC Bank
We provide secure, user-friendly banking solutions that empower everyone to manage their finances with ease.
View My Account


https://gencbanking-dev-ed.develop.my.site.com/gencbank/s/account

GenC Bank
Banking Made Easier

Home | My Bank Account | My Pass Book | Transfer Money

My Bank Account Details

Account Holder Name
Sai Kiran Chandra

PAN Card Number
ABCDE1234H

Email
chandrasaikiran03@gmail.com

Branch
Manyata Tech Park

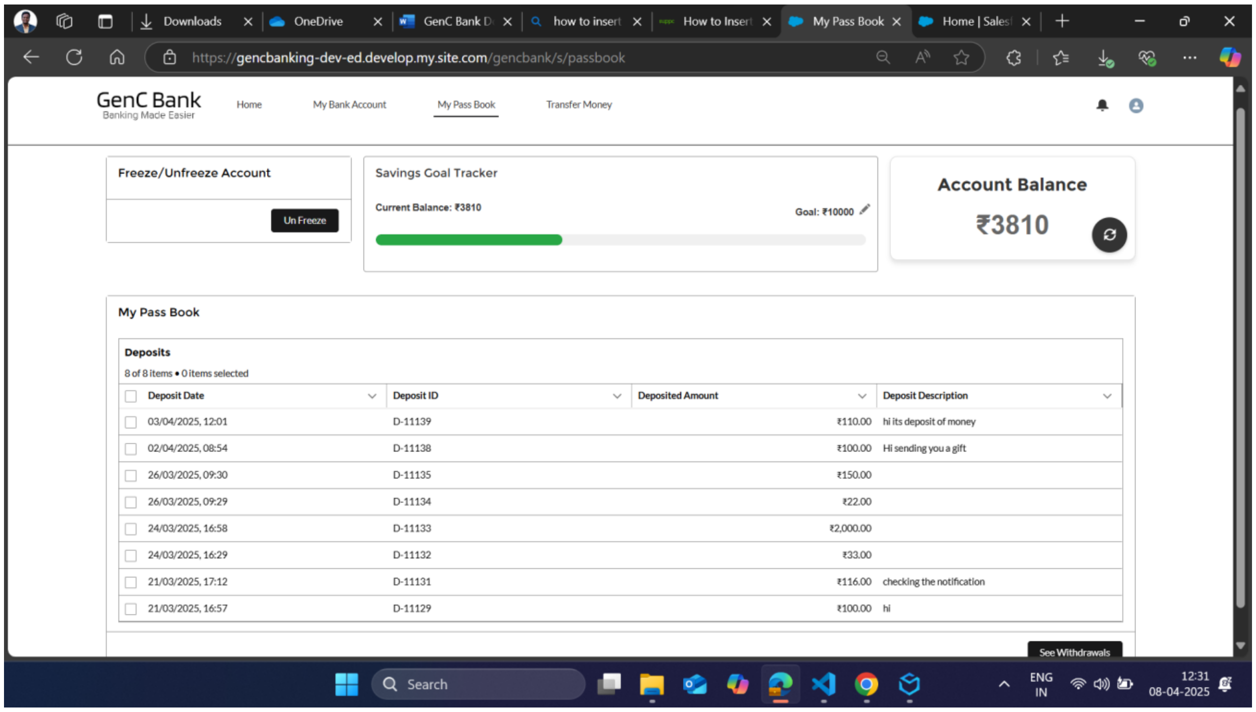
Bank
GenC Bank

Phone
9959292089

* Permanent Address
vvrm

✓ Current Address Same as Permanent Address

Update Details


https://gencbanking-dev-ed.develop.my.site.com/gencbank/s/passbook

GenC Bank
Banking Made Easier

Home | My Bank Account | My Pass Book | Transfer Money

Freeze/Unfreeze Account

Un Freeze

Savings Goal Tracker

Current Balance: ₹3810 Goal: ₹10000

Account Balance

₹3810

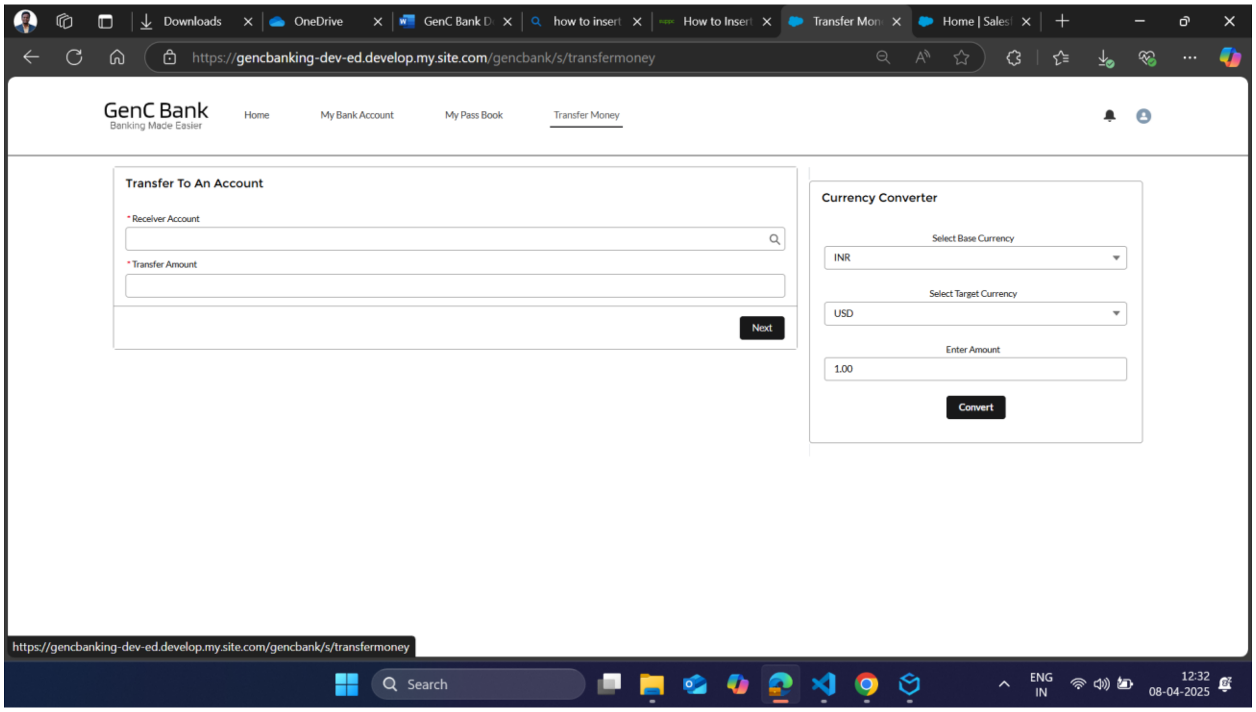
My Pass Book

Deposits

8 of 8 items • 0 items selected

Deposit Date	Deposit ID	Deposited Amount	Deposit Description
03/04/2025, 12:01	D-11139	₹110.00	hi its deposit of money
02/04/2025, 08:54	D-11138	₹100.00	Hi sending you a gift
26/03/2025, 09:30	D-11135	₹150.00	
26/03/2025, 09:29	D-11134	₹22.00	
24/03/2025, 16:58	D-11133	₹2,000.00	
24/03/2025, 16:29	D-11132	₹33.00	
21/03/2025, 17:12	D-11131	₹116.00	checking the notification
21/03/2025, 16:57	D-11129	₹100.00	hi

See Withdrawals



GenC Bank
Banking Made Easier

Home | My Bank Account | My Pass Book | Transfer Money

Transfer To An Account

* Receiver Account

* Transfer Amount

Next

Currency Converter

Select Base Currency: INR

Select Target Currency: USD

Enter Amount: 1.00

Convert

https://gencbanking-dev-ed.develop.my.site.com/gencbank/s/transfermoney