Background

Scrooge Global Bank wants to maintain a robust Customer Relationship Management (CRM) system to manage client interactions, streamline processes, and improve profitability. The CRM system must be easy for users to use, secure, scalable, and able to handle a high volume of transactions while ensuring data integrity and compliance with international banking regulations.

This project aims to build a CRM architecture landscape for a global bank, adhering to modern digital principles. The focus is on creating a system that is cloud-native, data-centric, client-focused, secure, modular, automated, accessible, and sustainable.

The system stores the following information:

1. Agent
2. Client and Account
3. Log
4. Transaction

   The system retrieves all the transaction logs from the Secure File Transfer Protocol (SFTP) server. This is the location where the transaction logs from the mainframe are stored. Teams will need to provide a "mock" SFTP server (bank transactions - add money ..)

The system provides the frontend to support the following functionalities:

1. There are two roles:
   a. admin - Account Management for the CRM
   b. agent - The user of the system for creating client profiles and accounts. They have access only to their own clients.
2. A simple frontend application that exposes back-office operations and information to be entered and used with ease. The layout and the design doesn’t need to be fancy. The focus is on functionality and ease of usability. For the purposes of the project, it should include these pages for Feature 1 to 4 (elaborated below).

Features

Feature 1: CRM User Management

Ensuring secure and seamless access to the CRM system is critical. This feature will handle user (admin/agent) enrollment and authentication.

There are 2 roles for the CRM system.

1. The admin role is used for account management. The root administrator cannot be deleted. Other admins can be created and deleted as necessary.
2. The agent role is the user of the system. The agent creates client profiles and accounts. They have access only to their own clients.

APIs (endpoints: e.g., POST /api/users):

1. Create User
2. Disable User
3. Update User
4. Authenticate User
5. Reset Password

Details:

1. Use OAuth 2.0 for secure authentication and authorization.
2. Adhering to Zero Trust principle, each service should be accessed with OAuth2-based token and the authenticity and authorization for all requests should be validated
3. (good to have) Implement multi-factor authentication (MFA) for added security
4. Secure password storage using hashing algorithms.

Feature 2: Client Profile Management

It will manage client account information. Refer to Appendix 2 for the fields.

Components:

1. Client Database: Stores all client information, including personal details, account status, and interaction history.
2. Account Database: Stores the account details for a client.

Significant Actions in Client Onboarding

1. Create Client Profile
   API Endpoint: POST /api/clients
   Action: Capture and store the initial client information.
   Steps:
   a. Agent fills out the onboarding form with the client fields.
   b. System validates the entered information.
   c. System assigns a unique Client ID.
   d. Client profile is created and stored in the database.
   e. There should be a log entry (feature 3).

2. Verify Client Identity
   API Endpoint: POST /api/clients/{clientId}/verify
   Action: Ensure the client's identity and details are accurate and meet regulatory requirements.
   Steps:
   a. System sends a verification request to the client.
   b. Client provides necessary identification documents (NRIC).
   c. System validates the documents against the provided information.
   d. Verification status is updated in the client's profile.
   e. There should be a log entry (feature 3).

3. Update Client Information
   API Endpoint: PUT /api/clients/{clientId}
   Action: Modify existing client details.
   Steps:
   a. System receives a request from client to update information
   b. System validates the new information.
   c. System updates the client's profile with the new details.
   d. There should be a log entry (feature 3).

4. Get Client Profile:
   API Endpoint: GET /api/clients/{clientId}
   Action: Retrieve client information.
   Steps:
   a. System receives a request to view client details.
   b. System fetches the client information from the database.
   c. System returns the client information.
   d. There should be a log entry (feature 3).

5. Delete Client Profile:
   API Endpoint: DELETE /api/clients/{clientId}
   Action: Remove client information from the system.
   Steps:
   a. System receives a request to delete a client profile.
   b. System verifies the request.

c. System deletes the client profile from the database.
d. There should be a log entry (feature 3).

6. Create Account
   API Endpoint: POST /api/accounts
   Action: Capture and store the initial account information.
   Steps:
   a. Agent fills out the onboarding form with the account fields.
   b. System validates the entered information.
   c. System assigns a unique Account ID.
   d. Account is created and stored in the database.
   e. There should be a log entry (feature 3).

7. Delete Account
   API Endpoint: DELETE /api/accounts/{clientId}
   Action: Remove account information from the system.
   Steps:
   a. System receives a request to delete an account.
   b. System verifies the request.
   c. System deletes the account from the database.
   d. There should be a log entry (feature 3).

Details:

1. The Client Database will use a relational database to ensure ACID properties for transactions.
2. The CRM system will support various interaction types (CRUD) and store metadata for each interaction.
3. The CRM system will include validation checks and business rules to ensure data integrity.
4. Ensure data is encrypted at rest and in transit.
5. Allow for audit trails to track changes to client data.

Feature 3: Logging of Agent Interactions and Client Communications

Managing and tracking client transactions is a critical aspect of the CRM system.

Components:

1. Transaction Database: Stores all transaction details.
2. Transaction Processing: Handles the creation, update, and deletion of transactions.

Business Transactions: Involving core client information interactions.

1. Client Profile Management (implemented in feature 2):
   Focus: Handling client data like Client ID, name, contact details, and address.
   Examples of Transactions: Creating, updating, or deleting client profiles.

   APIs to Implement: Create, Update, Get Details, Delete Transaction.
   (internal)

2. Client Communication:
   Focus: Using email for client communication.
   Examples of Transactions: Sending email notifications to clients (Feature 2: create/disable/update/ #1)

   APIs to Implement: Create Communication, Get Communication Status.

Details:

1. [Optional] Implement rollback mechanisms to handle transaction failures.
   a. Your team should explain how it is done and the frequency.

2. Ensure high throughput and low latency for transaction processing.
   a. minimum 100 concurrent agents. Your team can perform a stress test to demonstrate how resilient your application is.
   b. For the frontend application, it should not exceed 5 seconds of latency.

Feature 4: Bank Account Transactions Management

In reality, the core banking system is a separate mainframe application. To populate transaction information, the CRM system retrieves transaction information of accounts belonging to the client from an external SFTP server and populates its own table (Transaction).

Your Task:

1. Create your own set of dummy data based on the fields in Appendix 1
2. Implement a external SFTP server from which the CRM system will receive transaction information

Non-functional Requirements

Each team should handle the three mandatory non-functional requirements and optionally the rest of the requirements.

1. **[mandatory] Scalability/Performance**
    a. Implement auto-scaling for both application and database layers.
    b. Use load balancers to distribute traffic evenly across instances.
    c. Use caching mechanisms (if needed) to handle high loads.

2. **[mandatory] Availability**
    a. Ensure high availability with redundant systems and failover mechanisms.
    b. Use multiple availability zones for disaster recovery. Cross region if needed.
    c. Suggest and implement a robust backup and restore strategy.

3. **[mandatory] Maintainability/Extensibility**
    a. Design the system with modular components to allow for easy addition of new features.
    b. Use APIs to enable integration with external systems.
    c. Ensure backward compatibility when introducing new features (e.g. when introducing new features in the APIs).

4. **Consistency**
    a. Implement strong consistency models where required.
    b. Use eventual consistency for read-heavy systems where immediate consistency is not critical.

5. **Resiliency**
    a. Design the system to handle failures gracefully (e.g., circuit breakers, retries).
    a. Implement disaster recovery plans with defined Recovery Time Objective (recommended: 12 hours) and Recovery Point Objective (recommended: 12 hours).
    b. Use fault-tolerant architectures to minimize impact of failures.

6. **Usability**
    a. Ensure the user interface is intuitive and accessible to all users.
    b. Provide comprehensive documentation and user guides.
    c. Conduct user testing to identify and address usability issues..

7. **Observability**
    a. Implement comprehensive monitoring and logging.
    b. Use metrics and dashboards to gain insights into system performance.
    c. Set up alerting mechanisms for proactive issue resolution.

8. **Security**
    a. Implement multi-layer security, including firewalls, encryption, and intrusion detection.
    b. Conduct regular security audits and penetration testing.
    d. Ensure compliance with industry standards (e.g., GDPR, PCI-DSS).
        - Any PII information should be encrypted or filtered (in logs).
        - GDPR (General Data Protection Regulation) allows for an end user to request that their information be destroyed.
    e. All API should be designed based on OpenAPI standards.

9. **Durability**
    a. Use durable storage solutions with replication and backup.
    b. Implement data retention policies to ensure data longevity.

c. Ensure data is not lost during system failures or updates.

10. Agility

a. Use agile development methodologies to iterate and deliver features quickly.
b. Implement CI/CD pipelines for automated testing and deployment. (demo a CI/CD)
c. Foster a culture of continuous improvement and innovation.

11. Architecture

a. UBS adopts a layered architecture approach by segregating the Presentation and Domain layer.
b. Application should be cloud native.
c. Applications can adapt modularization by using Microservice or Micro Frontend approach (optional) and separating each service based on its domain context.

12. Data Segregation

a. Each country/legal entity will have its own data protection policy and data should not be shared with other legal entities.
b. Application will be hosted separately based on the legal entity requirement and this instance will only process that particular legal entity data.

13. Automated Testing

a. Application should be capable of automated testing.

APPENDIX 1: Digital Principles

1. **Cloud Native First**
   The CRM system will be designed and built using cloud-native technologies to leverage scalability, flexibility, and efficiency. This includes utilizing microservices architecture, containerization (e.g., Docker), and orchestration (e.g., Kubernetes).

2. **Do Less - Reuse, Build**
   The project will prioritize the reuse of existing solutions and components wherever possible. This includes leveraging existing APIs, libraries, and frameworks to reduce development time and improve reliability.

3. **Data Centric, Client Focused, and Predictive**
   The CRM system will be centered around data, with a strong focus on client needs. Advanced analytics and machine learning algorithms will be used to predict client behavior, personalize interactions, and enhance client satisfaction.

4. **Zero Trust Security**
   Implementing a zero trust security model will ensure that every request is authenticated and authorized, regardless of its origin. This includes multi-factor authentication (MFA), encryption, and continuous monitoring for threats.

5. **Open, Modular, and Unbundled**
   The system will be built using open standards and modular components, allowing for easy integration, scalability, and flexibility. This unbundled approach ensures that individual components can be updated or replaced without affecting the entire system.

6. **Automated By Design**
   Automation will be integral to the CRM system, from deployment and scaling to monitoring and maintenance. Automated testing, CI/CD pipelines, and infrastructure as code (IaC) will ensure reliability and efficiency.

7. **All Services Any_Channel**
   The CRM system will support multi-channel access, allowing clients to interact with the bank through various channels (web, mobile, chat, etc.). This ensures a seamless and consistent client experience.

8. **All Services, Always Available**
   High availability and resilience will be key priorities, ensuring that all services are accessible 24/7. Redundancy, load balancing, and failover mechanisms will be implemented to minimize downtime.

9. **Simple, Reliable and Accessible, for Everyone**
   The CRM system will be designed to be user-friendly and accessible to all clients, including those with disabilities. Simplicity and reliability will be at the forefront of the design process.

10. **Ethical, Explainable, and Sustainable**
    The system will adhere to ethical guidelines, ensuring transparency and fairness in all interactions.

APPENDIX 2: Attributes

User (Feature 1)

1. id: Unique User ID
2. First Name: The User's first name
3. Last Name: The User's last name
4. Email: The User's email
5. Role: Admin, or Agent

Client (Feature 2)

1. Client ID: Unique identifier for each client (auto-generated).
2. First Name: Client's first name.
3. Last Name: Client's last name.
4. Date of Birth: Client's date of birth.
5. Gender: Client's gender.
6. Email Address: Client's email for contact and communication.
7. Phone Number: Client's contact number.
8. Address: Client's residential address.
9. City: Client's city of residence.
10. State: Client's state of residence.
11. Country: Client's country of residence (Singapore).
12. Postal Code: Postal code of the client's address.

Validations

Field	**Type**	**Validation**	**Business Rule**
Client ID	String	System-generated, unique, non-editable.	Must be unique across the system.
First Name	String	Required, minimum length 2 characters, maximum length 50 characters.	Must contain only alphabetic characters and spaces.
Last Name	String	Required, minimum length 2 characters, maximum length 50 characters.	Must contain only alphabetic characters and spaces.
Date of Birth	Date	Required, valid date format, must be in the past.	Age must be between 18 and 100 years.
Gender	String	Required, must be one of predefined options (Male, Female, Non-binary, Prefer not to say).	
Email Address	String	Required, valid email format.	Must be unique across the system.
Phone Number	String	Required, valid phone number format (e.g., +1234567890), minimum length 10 digits, maximum length 15 digits.	Must be unique across the system.
Address	String	Required, minimum length 5 characters, maximum length 100 characters.	

City	String	Required, minimum length 2 characters, maximum length 50 characters.	
State	String	Required, minimum length 2 characters, maximum length 50 characters.	
Country	String	Required, minimum length 2 characters, maximum length 50 characters.	
Postal Code	String	Required, valid postal code format, minimum length 4 characters, maximum length 10 characters.	Must match the country's postal code format.

Account - (Feature 2)

1. Account ID: Unique identifier for each account (auto-generated).
2. Client ID: Unique identifier for the associated client.
3. Account Type: Type of account (e.g., Savings, Checking, Business).
4. Account Status: Status of the account (e.g., Active, Inactive, Pending).
5. Opening Date: Date when the account was opened.
6. Initial Deposit: Amount of the initial deposit made by the client.
7. Currency: Currency in which the account operates (SGD for Singapore Dollar).
8. Branch ID: Identifier for the branch where the account is opened.

Log - (feature 3)

1. CRUD Create, Read, Update, Delete
2. Attribute name "First Name|Address"
   For Create, Read, Delete, store Client ID. For Update, store the attribute.
3. Before Value e.g. "LEE|ABC"
4. After Value e.g. "TAN|XX"
5. Agent ID The agent ID
6. Client ID: The client ID
7. DateTime The datetime using ISO 8601 format.

Transaction - (feature 4)

1. ID The transaction ID
2. Client ID The client ID
3. Transaction (D)eposit, (W)ithdrawal
4. Amount The deposit or withdrawal amount
5. Date The transaction date
6. Status: The transaction's status. Possible values are Completed, Pending, Failed

Appendix 3: UI Mockup

Login Page

Login Page:

markdown Copy code

```
+----------------------------------+
|          CRM Login               |
+----------------------------------+
| Username: [____________________] |
| Password: [____________________] |
|                                  |
| [ Login ]                        |
+----------------------------------+
```

Admin Dashboard

Admin Dashboard:

sql Copy code

```
+----------------------------------+
|          Admin Dashboard         |
+----------------------------------+
| [Create New Account] [Manage Accounts] [View Transactions] [Settings] |
+----------------------------------+
| Recent Activities:               |
| - User1 created an account       |
| - User2 updated profile details  |
|                                  |
+----------------------------------+
```

Note:

1. Agents can only see their own client activities.
2. Admins can see the activities of all agents.

Admin Manage Account

Manage Accounts:

yaml Copy code

```
+-----------------------------------+
|          Manage Accounts          |
+-----------------------------------+
| Search: [_____________________]   |
|                                   |
| [Search]  [Add New Account]       |
+-----------------------------------+
| Account List:                     |
| - Account ID: 1234, Client: John Doe |
| - Account ID: 5678, Client: Jane Smith |
| - Account ID: 9101, Client: Alan Turing |
|                                   |
| [Edit] [Delete]                   |
+-----------------------------------+
```

Agent Dashboard

Agent Dashboard:

sql Copy code

```
+-----------------------------------+
|          Agent Dashboard          |
+-----------------------------------+
| [Create Client Profile] [Manage Profiles] [View Transactions] |
+-----------------------------------+
| My Recent Activities:             |
| - Created profile for Client1     |
| - Updated profile for Client2     |
|                                   |
+-----------------------------------+
```

Agent create Client Profile

Create Client Profile:

markdown Copy code

```
+----------------------------------+
|        Create Client Profile     |
+----------------------------------+
| First Name:   [_______________] |
| Last Name:    [_______________] |
| Date of Birth:[_______________] |
| Gender:       [_______________] |
| Email:        [_______________] |
| Phone:        [_______________] |
| Address:      [_______________] |
| City:         [_______________] |
| State:        [_______________] |
| Country:      [_______________] |
| Postal Code:  [_______________] |
|                                  |
| [ Save ]    [ Cancel ]           |
+----------------------------------+
```

Agent View Transactions

View Transactions:

yaml Copy code

```
+----------------------------------+
|          View Transactions       |
+----------------------------------+
| Search: [_____________________]    |
|                                  |
| [Search]                         |
+----------------------------------+
| Transaction List:                |
| - Transaction ID: 1001, Amount: $500, Status: Completed |
| - Transaction ID: 1002, Amount: $750, Status: Pending   |
| - Transaction ID: 1003, Amount: $200, Status: Failed    |
|                                  |
| [View Details] [Retry]           |
+----------------------------------+
```