2018

***Team: ME2***

***Harmony exists in Diversity!***

***12/19/2018***

Rainbow System Design



|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 12/19/2018 | 1.0.0 | System Design Document | * Ehab Tarik * Elsabeth Melaku * Melat Hadgu * Merih Gebreyohannes |

Table of Contents

[1. Introduction 2](#_Toc533024365)

[1.1 Problem statement 2](#_Toc533024366)

[1.2 Purpose 2](#_Toc533024367)

[1.3 Scope 2](#_Toc533024368)

[1.4 Technologies 2](#_Toc533024369)

[2. Use Cases 3](#_Toc533024370)

[2.1 List All Clients 3](#_Toc533024371)

[2.2 List Client Transaction by Client No 4](#_Toc533024372)

[2.3 Add Client 5](#_Toc533024373)

[2.4 Achieving/Daily Batch 6](#_Toc533024374)

[2.5 Generate Monthly Statement 7](#_Toc533024375)

[3. High Level Design 8](#_Toc533024376)

[3.1 Overview 8](#_Toc533024377)

[3.2 Main Components 8](#_Toc533024378)

[4. Detailed Design 9](#_Toc533024379)

[4.1 Class Diagram 9](#_Toc533024380)[9](#_Toc533024381)

[4.2 Sequence Diagram 10](#_Toc533024382)

[5. Issues/ assumption and future releases 11](#_Toc533024383)

# Introduction

## Problem statement

Banking system is good example of enterprise application. It deals with huge amount of daily transactions data, which should be maintained for business analysis. It integrates many products and services together to achieve business goals, for example handling, adding new client, adding daily transaction, achieve them, generate reports, route short messages, provide services added values, …etc.

## Purpose

Rainbow is a proof of concept project that demonstrates the capabilities of technologies and how to use them to address real-life enterprise issues.

It illustrates how to deal with huge amount of transactional data through archiving and make it available for adding transaction, reporting and data analysis purposes. It also provides a highly available and scalable integration solution which guarantee data consistency across different applications. And provide high level of abstraction to integrate a loosely coupled services plus handling the interactions between these services with minimum effort using orchestration.

## Scope

Rainbow project introduces a service layer exposed through RESTful services to be invoked by external applications and integrated with Reporting systems through Spring Integration. It has 2 batch jobs one for archiving daily transactions and generate aggregates and the other is responsible for generating monthly statements.

## Technologies

To achieve the mentioned functionality, we applied technologies such as

* ORM
* AOP
* Spring Rest
* Spring Batch
* Spring Security
* Spring Integration (AMQP – JMS)
* Maven

# Use Cases

## List All Clients

|  |  |  |
| --- | --- | --- |
| **Intent** | | Listing all the clients in the system |
| **Description** | | The user will list all the clients in the system |
| **Preconditions** | | None |
| **Success Post Condition** | | Clients retrieved successfully |
| **Failed Post Condition** | | Return a message to indicate failure causes. |
| **MAIN FLOW** | | |
| **Step** | **Action** | |
|  | The user pick to list all the clients | |
|  | A Rainbow Rest web service will be called in order to list all the clients | |
|  | If the clients were retrieved successfully, then they will be displayed | |
|  | If there is a failure and error message will be displayed | |

## List Client Transaction by Client No

|  |  |  |
| --- | --- | --- |
| **Intent** | | Listing all the transaction by clientNo |
| **Description** | | Retrieving all the transaction based on a clientNo |
| **Preconditions** | | Transaction should be there for a client |
| **Success Post Condition** | | List of transactions are retrieved successfully |
| **Failed Post Condition** | | Return a message to indicate failure causes. |
| **MAIN FLOW** | | |
| **Step** | **Action** | |
| 1 | The user will select to retrieve transactions by clientNo | |
| 2 | The user will enter the clientNo | |
| 3 | A Rainbow Rest web service will be called in order to retrieve all the transactions for this client | |
| 4 | If the transactions were retrieved successfully, then they will be displayed | |
|  | If there is a failure and error message will be displayed | |

## Add Client

|  |  |  |
| --- | --- | --- |
| **Intent** | | Adding new client for Rainbow System |
| **Description** | | Create new client with the following information: Client number, First Name, Last Name, Email, etc. |
| **Preconditions** | | None |
| **Success Post Condition** | | A new Client is created. |
| **Failed Post Condition** | | Return a message to indicate failure causes. |
| **MAIN FLOW** | | |
| **Step** | **Action** | |
|  | Enter the information for the new client (FirstName, Last Name, Client No and Email) | |
|  | Then Submit | |
|  | A Rainbow Rest web service will be called in order to create the client | |
|  | If the client is created successfully, then a successful message will be shown | |
|  | If the client is not created, then a failure message will be shown (there’s validation here try to leave the clientNo for example empty) | |

## Achieving/Daily Batch

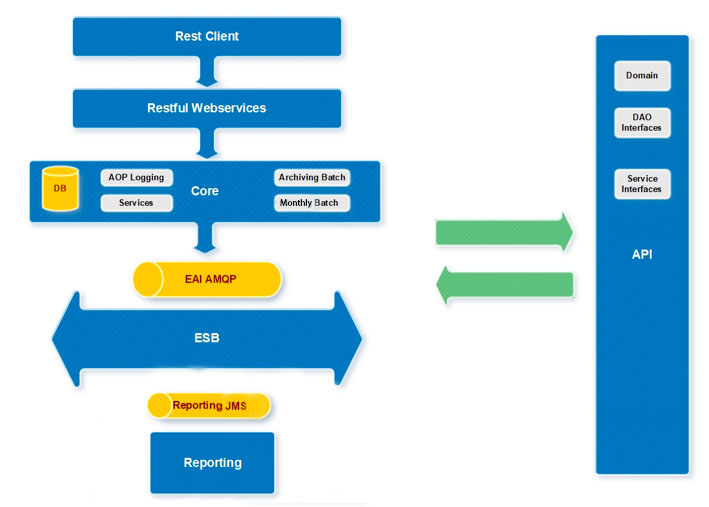
|  |  |  |
| --- | --- | --- |
| **Intent** | | Adding new client for Rainbow System |
| **Description** | | A scheduled Daily Batch Job reads all the daily transaction then transfer them to the transaction table (Historical Table) then mark the it as archived. |
| **Preconditions** | | There should be a daily transaction to be processed. |
| **Success Post Condition** | | A new Client is created. |
| **Failed Post Condition** | | Transaction (Historical) record is created and the daily transaction is marked as achieved |
| **MAIN FLOW** | | |
| **Step** | **Action** | |
|  | Define and Schedule Batch Job | |
|  | Batch Reader fetches all the daily transaction | |
|  | For each daily transaction a new transaction (Historical)one is created | |
|  | Marking each daily transaction as archived. | |

## Generate Monthly Statement

|  |  |  |
| --- | --- | --- |
| **Intent** | | Adding new client for Rainbow System |
| **Description** | | A scheduled Monthly Batch Job read all the transaction and generate a statement for each client. |
| **Preconditions** | | A Client should have at least one monthly transaction record over the affected period of the scheduled batch. |
| **Success Post Condition** | | A message to the reporting system that a statement has been generated |
| **Failed Post Condition** | | Monthly statement is created |
| **MAIN FLOW** | | |
| **Step** | **Action** | |
| 1 | Define and Schedule Batch Job | |
| 2 | Batch Reader fetches all the clients that have transactions | |
| 3 | For each client get all his transactions | |
| 4 | Calculate the total for all these transitions and generate a statement for this client | |

# High Level Design

## Overview



## Main Components

**API**

Abstraction Layer holds the domain model, DOA Layer Interfaces, and Service Layer interfaces.

**Rest-Client**

Rest-Client is a console Based Application which calls restful web services to simulate listing clients, transactions per client, create client and create transaction.

**Restful Services**

Restful services are exposed for client.

**Core**

This layer is the implementation of Service and DAO interfaces. Provides logging on DAO and Service Calls using AOP.

Contains Archive/Daily Batch as well as Monthly Batch.

**Archiving/Daily Batch**

Spring Batch that runs daily to archive transactions from daily transaction table to transaction table (Historical).

**Monthly Batch**

Scheduled to run monthly and responsible for getting the transaction for the client and create a monthly statement for it.

**ESB**

ESB layer contains many adapters to communicate with various transports and includes routing and transformation logic for messaging from rainbow-core to rainbow-eai and facilitate the integration of any future system (rainbow-email).

**Reporting**

This is stand along application which simulate how to publish and listen for a generated message.

# Detailed Design

## Class Diagram

## 

**Security Modeling Part:**

These our domain objects. Authorities are assigned to groups. UserCredentials are assigned to groups. Client has one credentials. There is ManyToMany relationship between Authorities and Groups, also between UserCredentials and Groups.

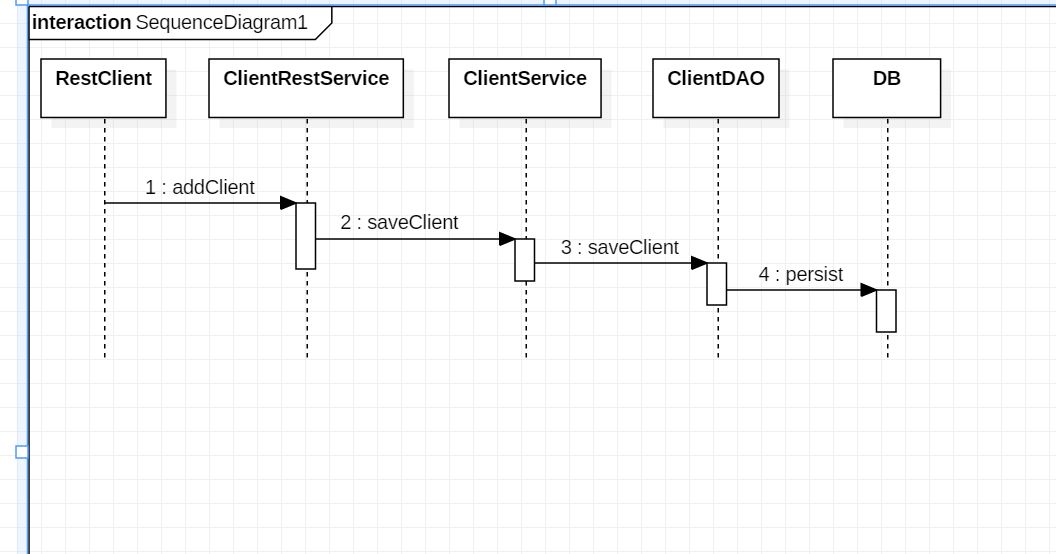
**Core Modeling Part:**

Client has many accounts.

The daily transaction has relation with the client and the account. The transaction (historical one) just has a simple attribute. Monthly statement reflects the total for the transactions

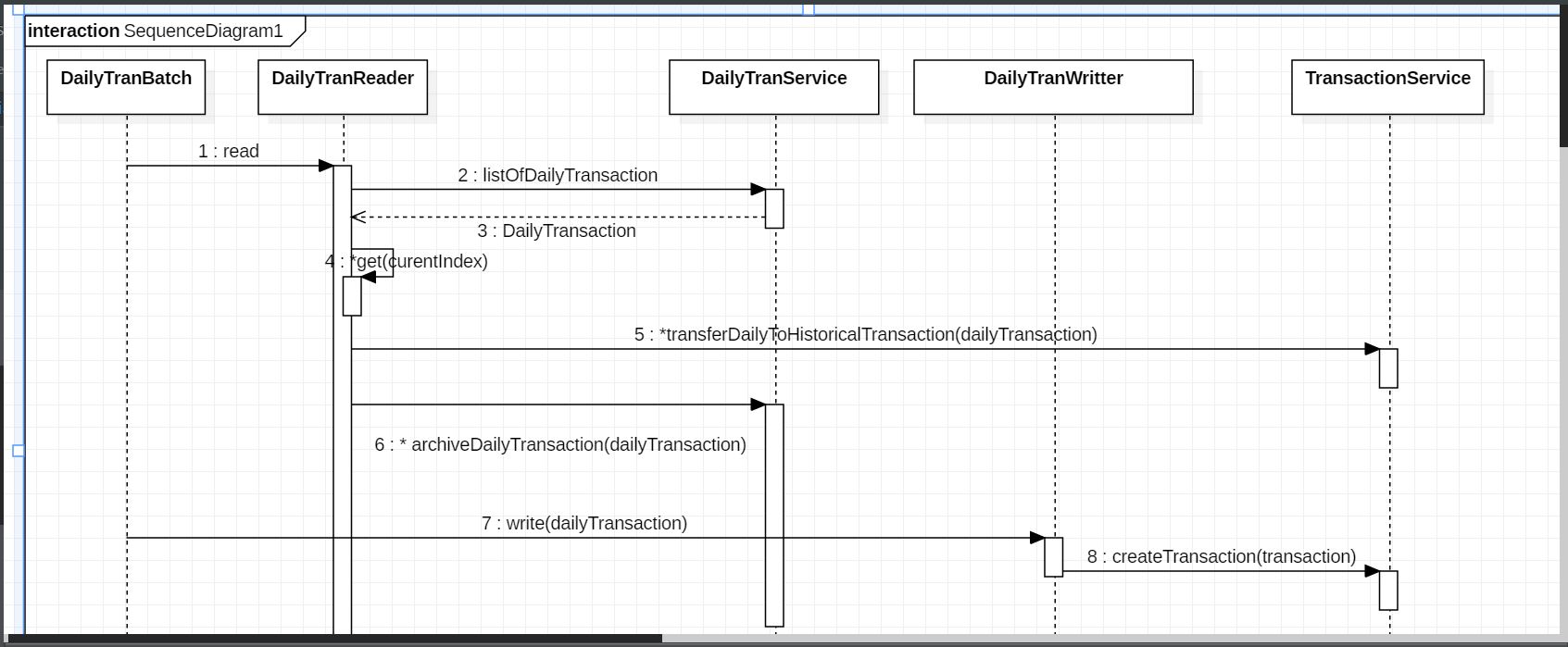
## Sequence Diagram

**Create Client Sequence Diagram**

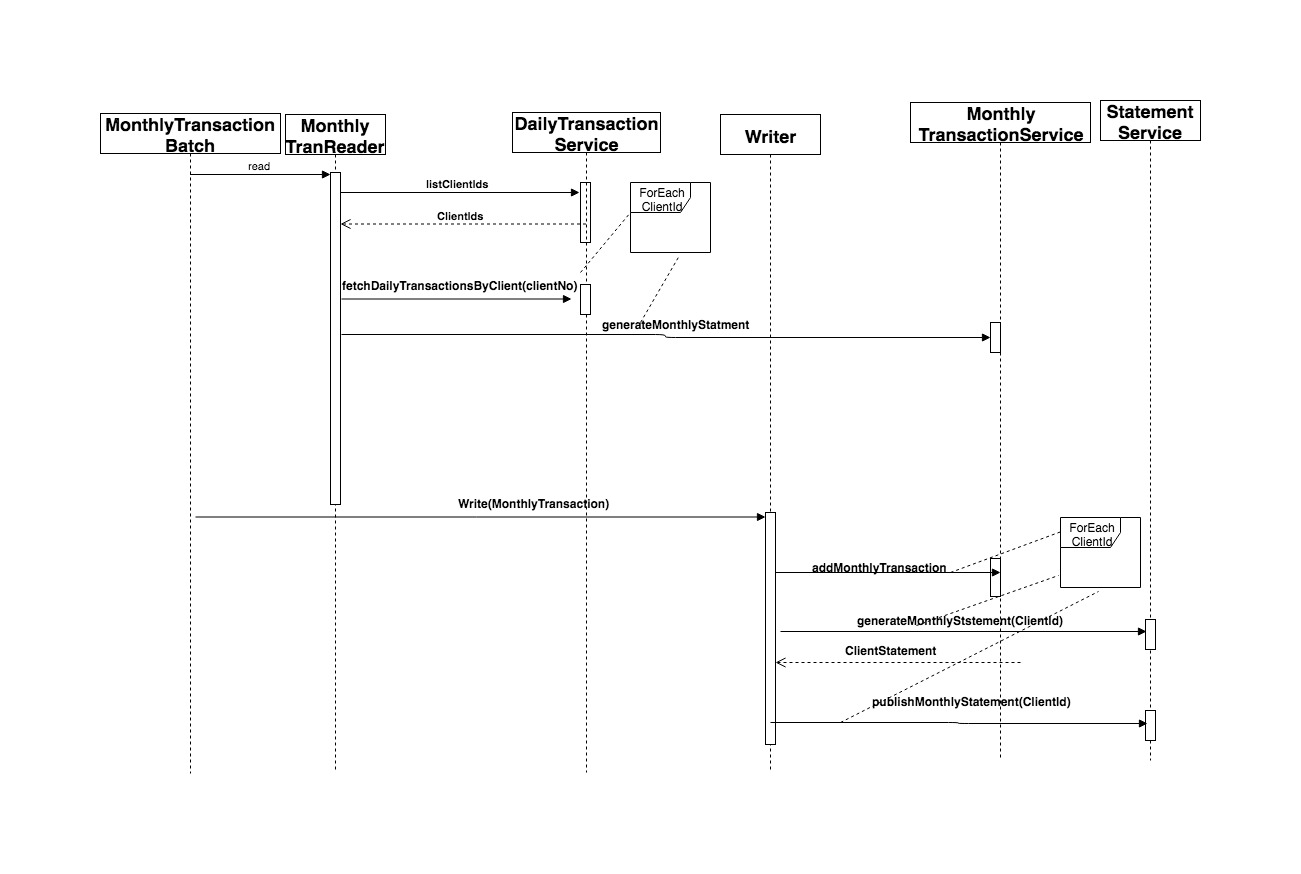
****

We want to insert new record for new client. There is Rest web service which accepts new client and passes it to clientService which passes it to clientDao for persistence into DB

**Daily Transaction Batch (Archiving) Sequence Diagram**

****

**Daily Transaction Batch (Archiving) Sequence Diagram**

****

# Issues/ assumption and future releases

* For simplicity while developing and testing wee assumed that the client will have just one account.
* For simplicity while testing creating the transaction we picked creating the transaction even this is a business violation as users are not allowed to create Historical transaction just daily. We go for this because the transaction table has simple attributes compared to the daily which has two relation which may make the population a little bit complex
* We didn’t have time to test the application with the security, but it’s implemented!!
* We didn’t have time to integrate different module like the email so when the statement is generated an email would be sent to the client (even the stand-alone application was there)
* While working with we found that the read is called more than one time which cause the process to fail (work around has been done).
* Because of the different mentalities and background we faced some issues but we managed to deliver (win-win situation 😊 )

For more references check the demos and lectures