Contents

[Tools and software used 2](#_Toc96204199)

[Functional Description: 2](#_Toc96204200)

[List of Restful Services 2](#_Toc96204201)

[Functional Description 4](#_Toc96204202)

[Architecture Diagram 5](#_Toc96204203)

[Code Architecture 5](#_Toc96204204)

[DB Diagram 7](#_Toc96204205)

[Best Practice Followed 8](#_Toc96204206)

[@Transactional 8](#_Toc96204207)

[Scheduler 8](#_Toc96204208)

[Asynchronous 9](#_Toc96204209)

[Validations 9](#_Toc96204210)

[Idempotency 10](#_Toc96204211)

[Used BigDecimal for Currency attributes 10](#_Toc96204212)

[Stored time in UTC format into the database 11](#_Toc96204213)

[Used database query to retrieve monthly data from the database rather than filtering in java code. 11](#_Toc96204214)

[Used Java8 features 11](#_Toc96204215)

[Used Mockito and Junit for Unit testing: 12](#_Toc96204216)

[Suggestions 12](#_Toc96204217)

[Event Driven 13](#_Toc96204218)

[Using Microservices and Message Queues 13](#_Toc96204219)

[Using AWS Services in Cloud 13](#_Toc96204220)

[Improvements needed in existing code 13](#_Toc96204221)

# Tools and software used

I have used the below stack for the given assignment

|  |  |
| --- | --- |
| Web Framework | Spring Boot |
| ORM | Hibernate |
| Database | PostgreSQL |
| Unit Testing | Junit and Mockito |
| Build Tool | Maven |
| Webservice Testing Tool | postman |
| Programming Language | Java 11 |

## Functional Description:

### List of Restful Services

|  |  |
| --- | --- |
| Rest Service URL | Http Method |
| /processAccountOpening | POST |
| Description  User can create a new account using this service  Body:   {         "bsb":"123456"   }  Sample Response:  {      "identification": 3802,      "bsb": 123456  } | |
| /transactions/{Identification} | POST |
| Description  User can create a new transaction for an existing account using this service  Body:   {         "txnAmount":100.00,         "txnType":"CRDT",         "txnRemarks":"Expense "   }  txnType can be “CRDT” /”DBIT” only.  Request Headers:  Idempotency-Key : <Time based UUID > eg: 959f9c81-c8d0-4a90-8af5-216581450610  This header needs to be unique for each transaction  Sample Response:  “Transaction created successfully”  Http Status: 201 CREATED | |
| /processAccountEndOfDayBalances | PUT |
| Description:  This service will calculate daily interest amount for each account based on their current balance, and credit the interest to their account. This happens only on first hit per day.  The interest amount calculation logic can be triggered only once per day, for other successive hits the interest calculation logic will not be executed, only the balance amount details will be listed.  The service response will contain all account details and their balance amount.  The balance amount shown also includes the current days interest.  Response will contain a list of account details like account balance with the Current date.  {      "balanceDate": "2022-02-19",      "AccountDetails": [          {              "bsb": 123456,              "identification": 2152,              "balance": 1900.52          },          {              "bsb": 123456,              "identification": 2452,              "balance": 200.03          },          {              "bsb": 123456,              "identification": 3802,              "balance": 100.01          }      ]  } | |
| calculateMonthlyInterest/{month}/{Year} | PUT |
| Description:  This service will sum up interest amounts for the given month and year which was calculated for each day with respect to each account.  Response:  [      {          "identification": 2152,          "calculatedamt": 0.52      },      {          "identification": 3802,          "calculatedamt": 0.01      },      {          "identification": 2452,          "calculatedamt": 0.03      }  ]  Where identification is the account identification number | |

### Functional Description

1. User/Tester can create an Account using /processAccountOpening
2. User/Tester can create transactions using /transactions/{Identification} for a specific account. They can trigger both Credit and debit transactions.
3. User/Tester will trigger /processAccountEndOfDayBalances service at the end of each day [Note: This service calculates interest amount only for the first trigger and respond with the balance amount, subsequent trigger will not calculate interest]

Interest is calculated based on Simple Interest formula

Simple Interest per day = (Balance Amount\* Percentage of interest)

Number of days in current year

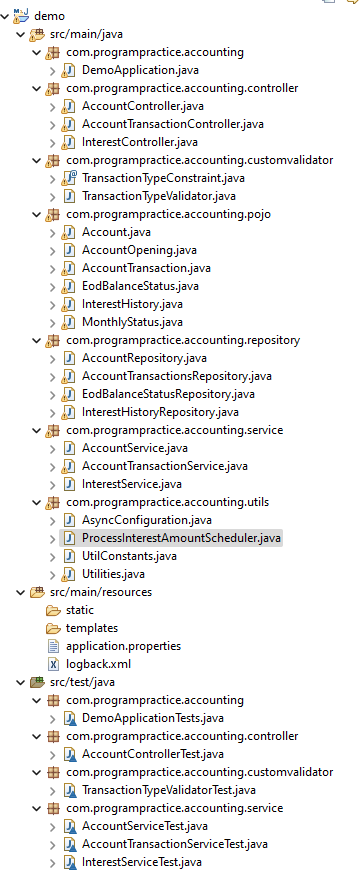
Assumptions: Percentage of interest= 5% =>0.05

The interest calculated will be credited to the account balance and an entry will be made in the transaction table.

1. User/Tester can check the total interest amount credited for a particular amount in a given month/year by retrieving the data from “Interest History “ Table using the “/calculateMonthlyInterest/02/2022” service.

# Architecture Diagram

## Code Architecture



Contain cron job Scheduler

Contain configuration for Thread executor used in Async operation

Junit Test cases

Contains Utility Functions and Constants used in the project

Service classes for implementing business logic for restful service

Repository classes for interacting with the databases.

Bean classes which define the properties used for database schemas, Restful apis request -body and response json structure

Validator classes for custom validation of properties in beans

Controller classes – to define the definition of RestFul Apis

Main java Class to Start the spring boot application

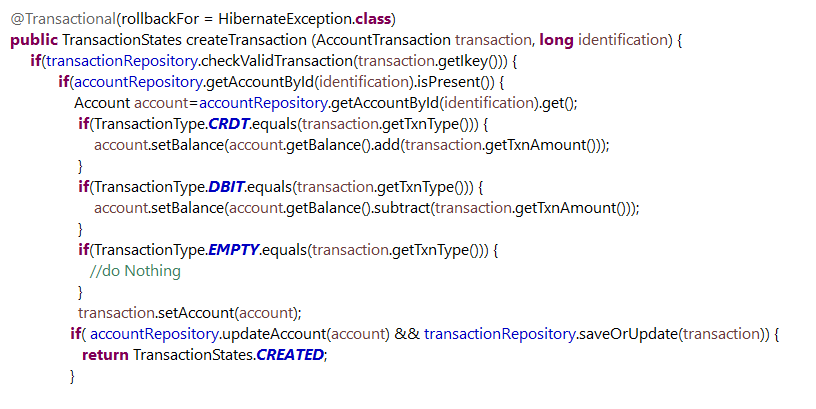
## DB Diagram

# Best Practice Followed

## @Transactional

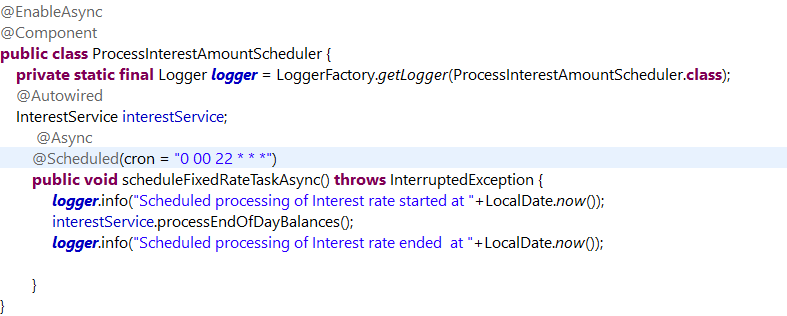
Used @Transactional annotation to process updation of multiple tables in a single transaction

Eg: While creating a record in Account transaction table. we will check for existing Idempotency-Key from “Account transaction” table and check for the account details from “account” table. Then based on the transaction, we update the account balance and insert the transaction details in their respective tables.



## Scheduler

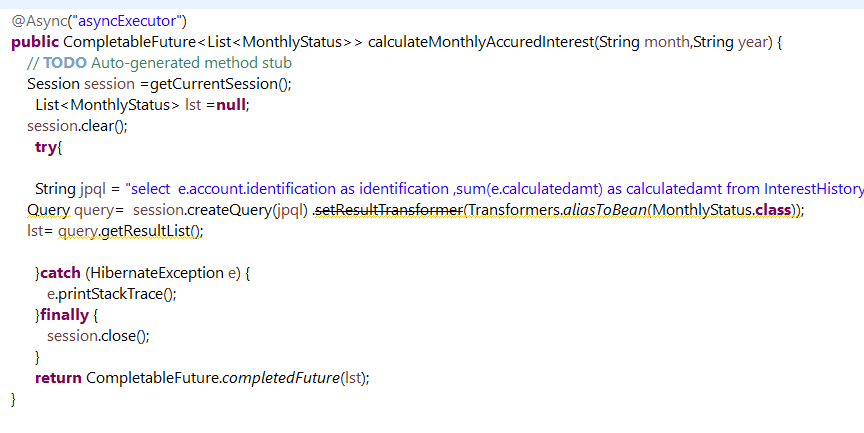
Created a Cron job scheduler to execute the code for calculating the interest amount at the end of each day at 10 30 pm and credit the amount into each respective account.



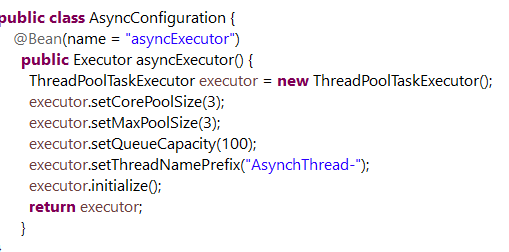
If the interest amount is already calculated and credited to the account for that particular day, this job will run but will skip the interest calculation and updation logics.

## Asynchronous

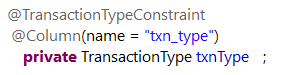
Used @Async annotation to execute the dB query in an asynchronous way.



Async Configuration



## Validations

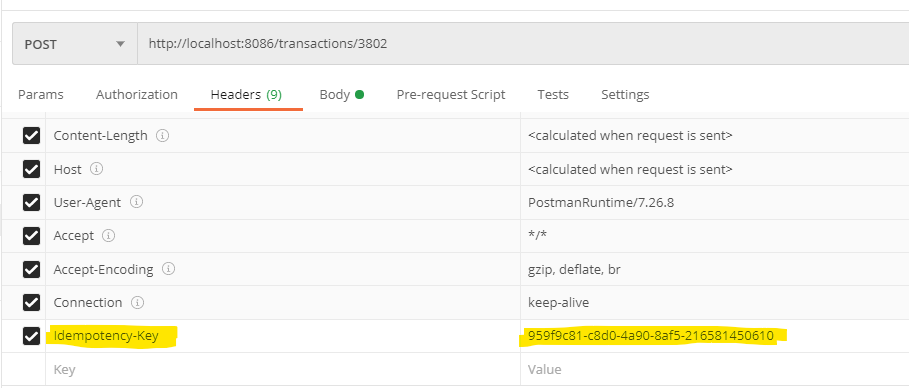


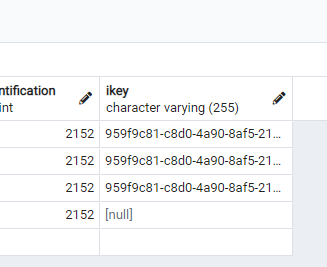
Used custom validators to validate Enums for the transaction types



## Idempotency

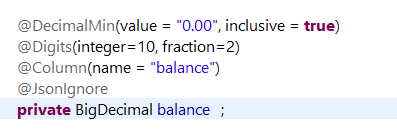
Time based UUID Key needs to be sent as request header for the “/transactions/{Identification}” service. If the transaction is successful, the key will be inserted into the transaction record. So that a duplicate transaction can be avoided, if the user tries to repeat the transaction with same key.





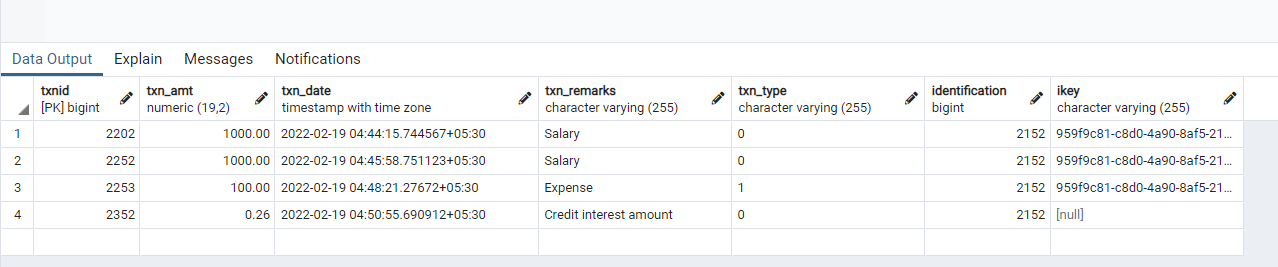
## Used BigDecimal for Currency attributes

Used BigDecimal Datatype for storing financial data like balance amount and interest amount.



## Stored time in UTC format into the database

Time stamp is stored in UTC format in the database.



## Used database query to retrieve monthly data from the database rather than filtering in java code.

String jpql = "select e.account.identification as identification ,sum(e.calculatedamt) as calculatedamt from InterestHistory e where date\_part('month',e.calculatedDate) = :month and date\_part('year',e.calculatedDate) = :year GROUP BY e.account.identification";

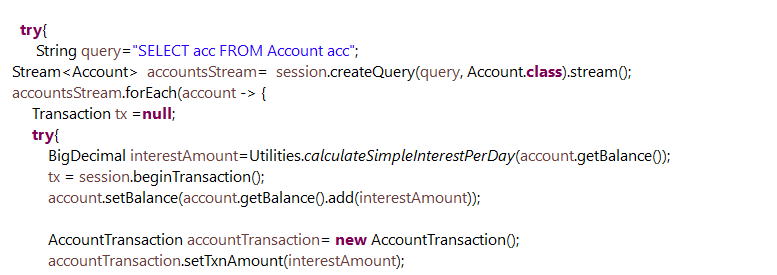
Query query= session.createQuery(jpql).setParameter(":month",month).setParameter(":year",year)

By this way, the computation happens in database itself and send only the required data to the spring boot application thus reducing the latency.

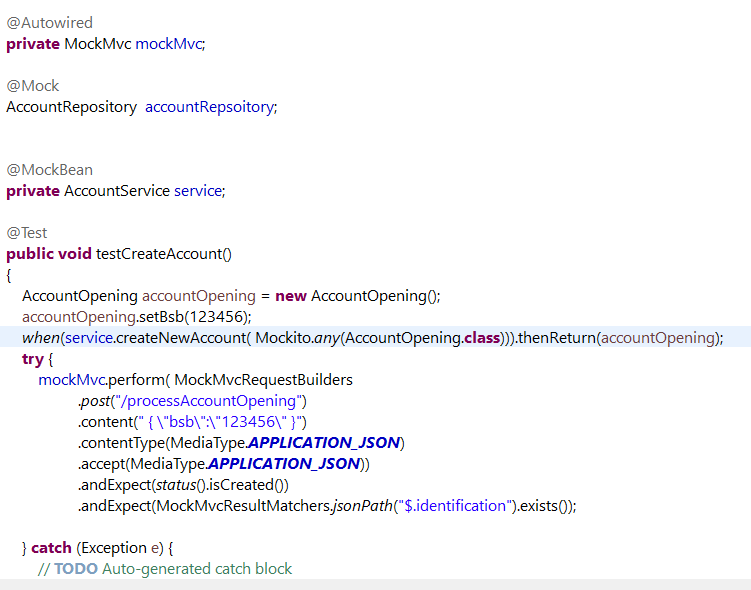
## 

## Used Java8 features

1. Streams



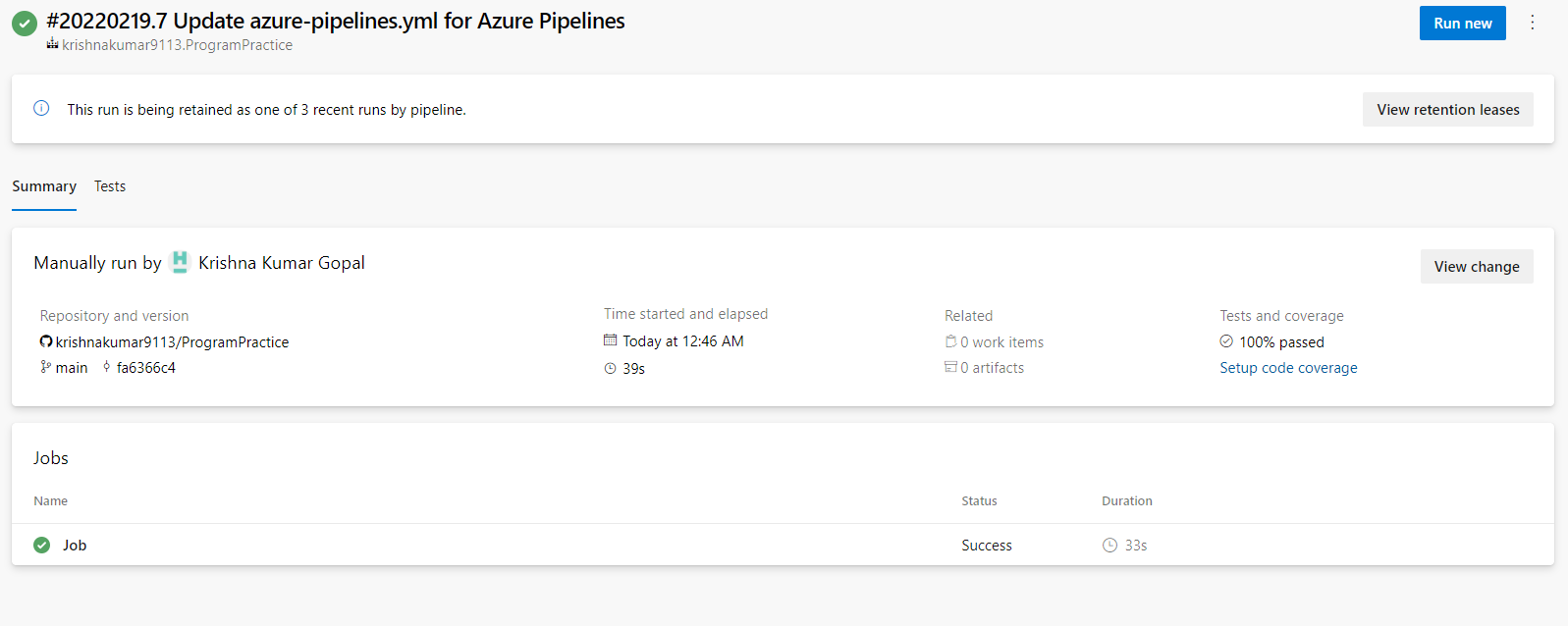
## Used Mockito and Junit for Unit testing:



## Built the code using Azure Devops Pipeline from GIT Repository:

I have built the code using Maven from Azure Devops CI/CD Pipeline along with the unit test cases.

Please find the attached screenshot for reference.



# Suggestions

## Event Driven

## Using Microservices and Message Queues

## Using AWS Services in Cloud

## Improvements needed in existing code

1. More Junit test cases can be added
2. Idempotency can be achieved in /processAccountOpening by having a column for “EmailId” as unique column in Account table. So that the user cannot submit the form multiple times with same emailID.