Rest Bank, Rest Bank Client, Rest Security, Rest Email System Design

Now it's Time to let your body to get Rest
And
Have a nice Christmas

Team Members:

Revision History

Date	Version	Description	Author
24/12/2016	1.0.0	Initial draft of the design document	Adell Esehak Tamrat Yohanes

REST

Table of Contents

REST

1. Introduction

Our enterprise architecture was mainly focused on banking system that connects four major division of the whole application. The division includes Rest Bank of Security Server, Email Server, main Server and the client. The customer or the client requests some resource from application through main server. Then the server takes a request and checks the customer if he is authenticated by checking the user name and password from the data base. The data base is directly connected with rest bank security server which has all security configurations and it aware who is authorized for certain resource. If the customer is authenticated then the main server will let the customer to use Email server as an option. Then it will return a response back to the client. The client can use any web based technology to access resources from the server.

1.1 Purpose

The purpose of this project is to demonstrate the proof of concept in enterprise applications, the concepts that have been implemented are:-

- How enterprise applications are integrated
- How the REST web service is implemented
- How the hibernate handles the mapping of objects to database tables
- How the security systems and authentications works
- How the aspect oriented programming is implemented
- How the Model view controller pattern works
- N tier model architecture

1.1.2 Technologies used

To achieve the mentioned functionality we applied technologies such as

- SoapUI: for testing the REST_Bank web service and REST_Client
- Spring REST web service: REST_Bank application in controller package
- Spring REST web service RestTemplate client: in REST_Client in rest package
- Hibernate Mapping: in REST_Bank model class we use hibernate mapping
- Spring Enterprise Integration is done on the REST_Bank application to send account transaction change notification to customer.
- AOP used to log account transaction and notify customer crosscutting concerns.
- REST Email uses JMS to listen the incoming email and forward the outgoing email using Spring JavaMailSenderImpl

2. Requirements - Use-Case – Usage Scenarios

Withdraw	
Intent	Withdraw Money
Author	Yohanes

REST

Last Update:	12/23/2016
Primary Actor	Client application
Secondary Actors	Customer
Description	The user can request to Rest Bank by the client side application provided
Preconditions	• The customer need to create account and • Authenticated by the system
Trigger	• When the customer asks for(clicks a withdraw button)
Success Post Condition	The customer will successfully withdraw the amount they ask abased on the bank policy
Failed Post Condition	

MAIN FLOW	
Step	**Action**
1	The customer will initialize the withdrawal action from the web service
2	The web service will maintain the interaction between the core banking the client
3	The web client will make some input field validation and will sent the request to the server
4	The banking server also will make additional authentications and check the operation validity based on the banks policy
5	Then the system will notify for the integration server for the changes made in the current account
6	The integration system will check the customers subscription and passes
7	Return the results for the web client
8	The web client present the result for the customer in appropriate format
	Use case ends

RELATED INFORMATION	
Performance	
Frequency	Every minute
NOTES & ISSUES	PUT: REST_Bank/account/{id}/withdraw
Future Considerations	

Creating Account	
Intent	Creation of new bank account object
Author	Yohanes
Last Update:	12/23/2016
Primary Actor	Web service
Secondary Actors	Bank agent(internal user employed by the bank)
Description	The user can request to Rest Bank by the client side application provided
Preconditions	Authenticated by the system Need the right privileged to create account
Success Post	Account creation page will popup

Condition	
Failed Post Condition	Account create page will be displayed with appropriate error message
MAIN FLOW	
Step	**Action**
1	The bank officer will initialize the create account action from the web service
2	The web service will maintain the interaction between the core banking the client
3	The web client will make some input field validation and will sent the request to the server
4	The banking server also will make additional authentications and check the operation validity based on the banks policy
5	Account will be created and persisted to the database if all the provide information is valid
6	Then the system will notify for the integration server for the changes made in the current account
7	The integration system will check the customers subscription and passes
8	Return the results for the web client
9	The web client present the result for the customer in appropriate format
10	Use case ends
RELATED INFORMATION	
Performance	
Frequency	Every minute
NOTES & ISSUES	POST: REST_Bank/account/add
Future Considerations	

3. High Level Design

This section of the document sets out at high level how the solution fits together and shows different views how the systems fit within the Enterprise.

Deployment view – Network Perspective

This view shows the deployment scenario for a system which enables any one of the systems to function properly by interacting with other systems. Since the core business of the bank, the mail and security system can be deployed centrally, they can be deployed at the Bank main branch that employs a centralized approach for all application and database servers. Client application can be deployed centrally or at each bank branch and request the web services provided by the Rest Bank application. Employees and customers can connect to the client application (online bank, mobile bank, ATM) via internet.

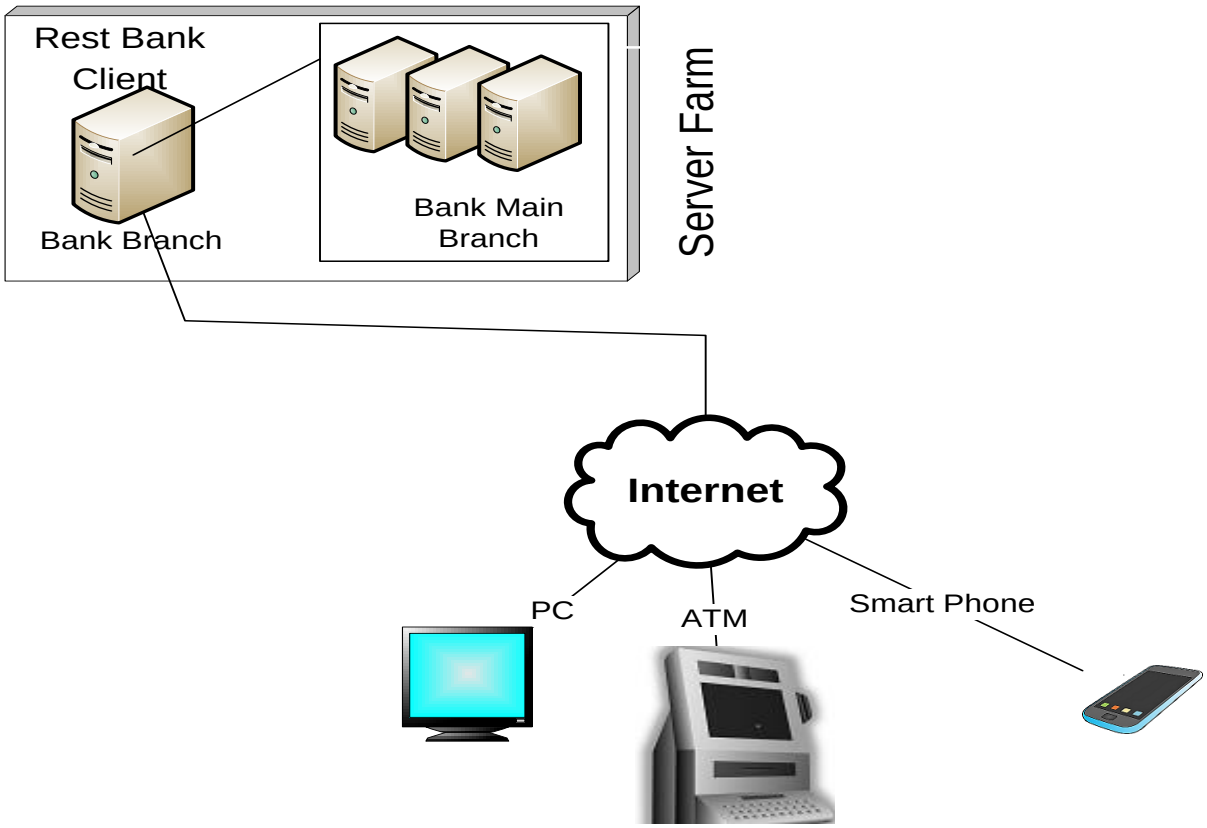
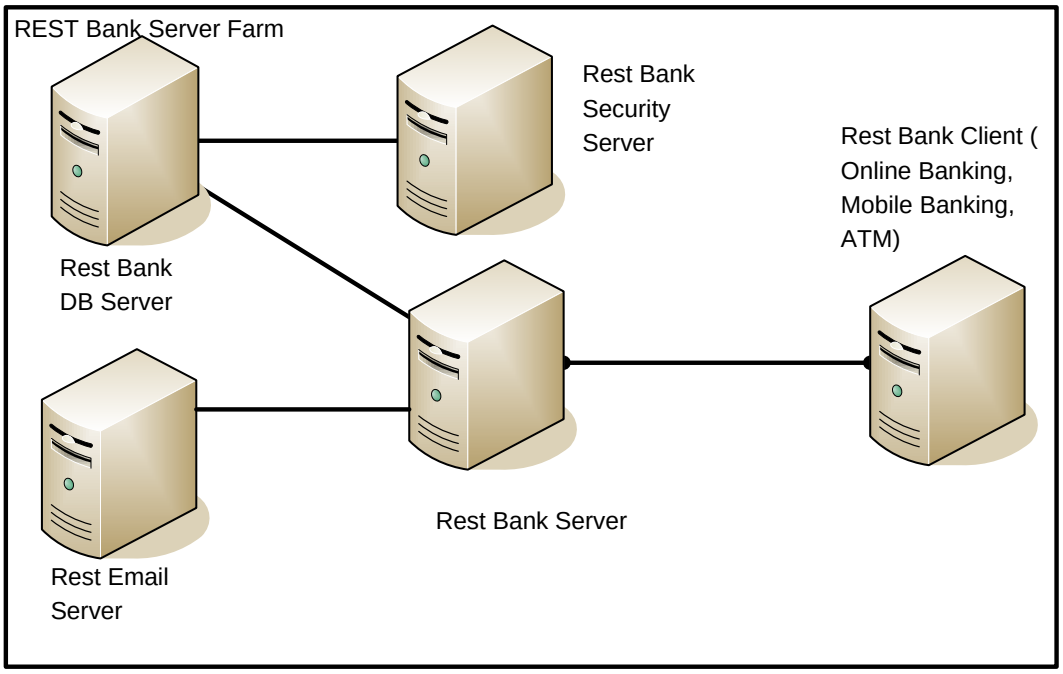


Figure 1 Deployment diagram

The Server Farm

The Diagram below shows the proposed organization of the servers for deployment of Rest Bank system server farm. A number of servers may be employed to provide availability and fail over



No	Element	Description
1.	DB Server	The Database Servers serve as data store for the entire enterprise applications. **Rest** Enterprise systems, MySQL is installed in the server.
2.	Rest Bank Sever	The main application server that hosts the **Rest Bank** applications. Since Rest Bank system will have many client applications (Online Banking, ATM, Mobile Banking), the application expected load will be high. Performance tuning in the application should be done, clustering and a good server computer will be used to provide high performance.
3.	Rest Security Server	Rest Security Server host **Rest Security** system. The security system is concerned with Authentication and Authorization of access to specific resource using group based security. Then **Rest Bank** uses the data saved by the Rest Security system to allow or deny a client using group based security configuration.
4.	Rest Email Server	Rest Email Server hosts mail application that receives incoming e-

		mail from **Rest Bank** application and forwards outgoing e-mail for delivery.
5.	Rest Bank Client	Rest Bank Client server could be Online Banking, ATM or Mobile Banking System that uses the **Rest Bank** System. Whenever a client application requests the **Rest Bank**, It has to be authenticated to verify whether that the client has the appropriate permission to perform that particular operation on the **Rest Bank.**

4. Detailed Design

Rest Bank maintain quite wide area and the systems runs in 4 different machines, we have domain classes for the core banking application, a replica domain object in the REST client and for the integration and messaging system we also have some domain classes. The below diagram shows the model class interaction and the level of associations between them.

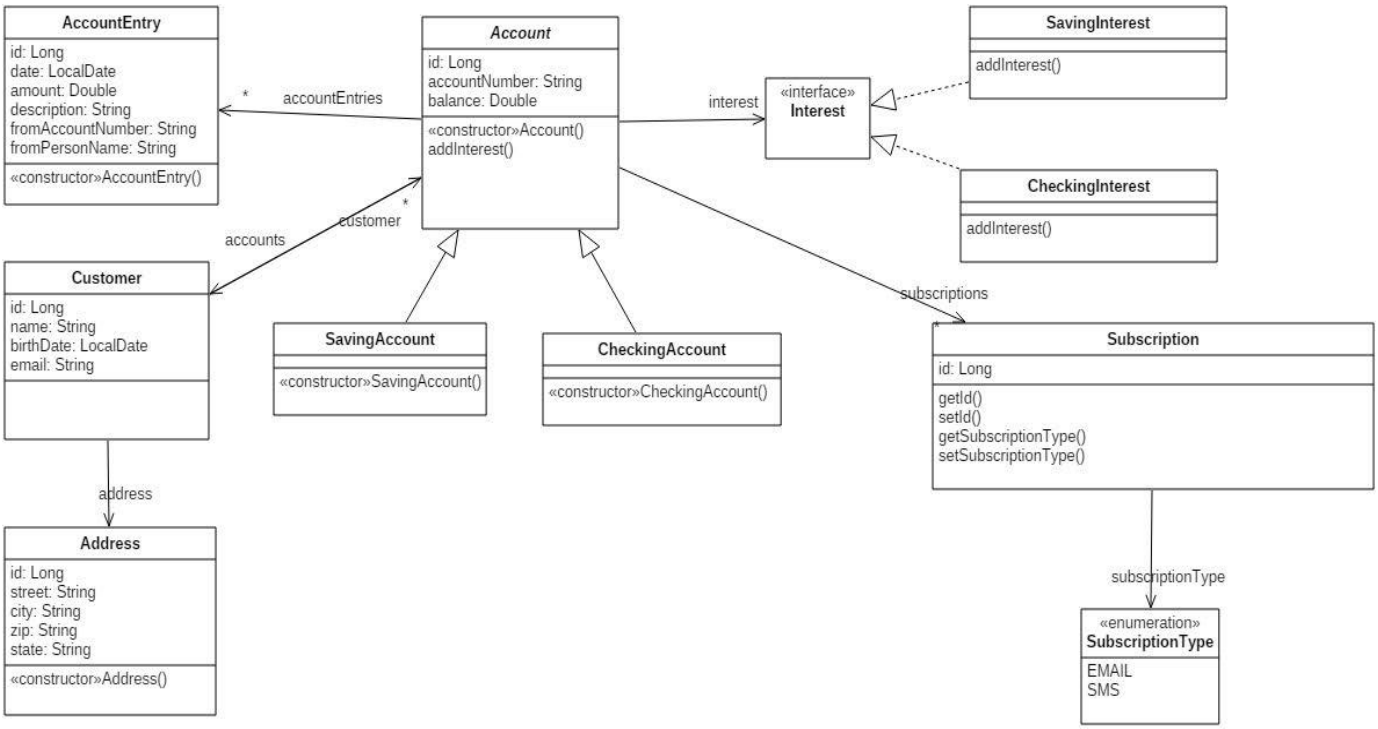


Figure 2 Domain objects and interaction

NOTE: here we didn't add the AbsractEntitty object which is the common for all the classes that provide the versioning profiles and it doesn't need to be persisted in the database.

We also have DateConverter class which is like a date converter utility for the localDate that java 8 provides, it need to be changed to database Date and vice versa.

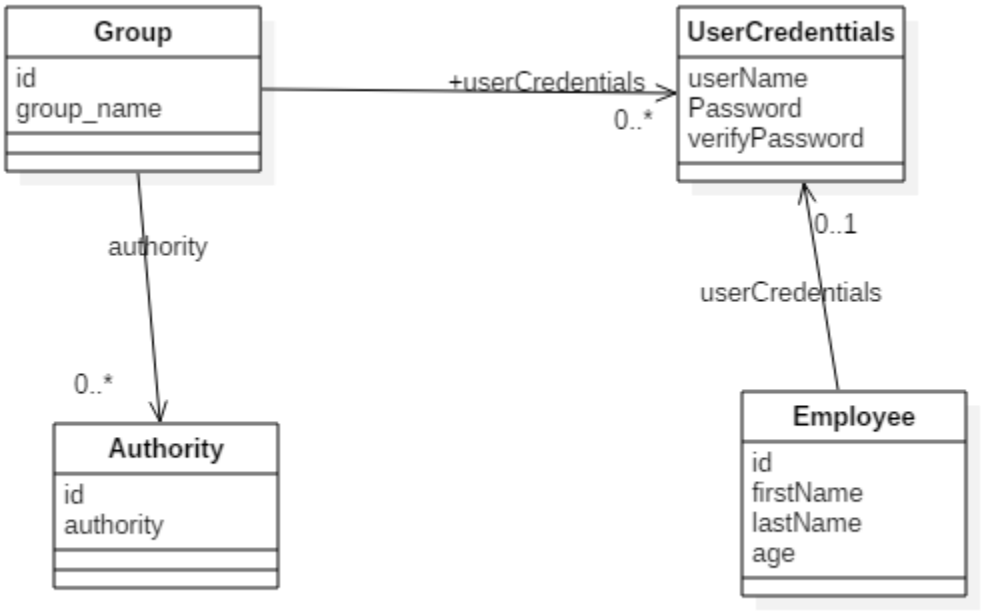


Figure 3 Security system domain class Diagram

Our Rest Bank Security includes Group, UserCredentials, Authority and Employee. we create UserCredential and Group domain object and insert in to Group; and we created Employee and assign to UserCredentials. In our above diagram we have relations in which Group domain have many UserCredentials and Authority. Our Employee Object has at most One userCredentials.

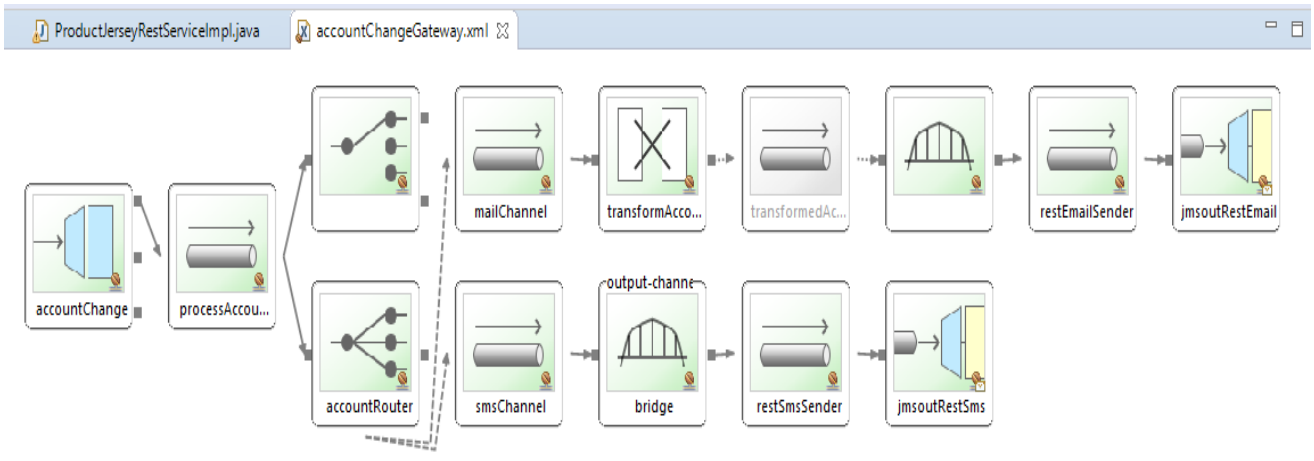


Figure 4

REST

Use Case: Notify account change to Customer via Email and/or SMS Rest Bank receives account Transaction (deposit, withdraw or transfer) via webservice Rest Bank generates an Account It broadcasts the Account to accountChangedGateway The Gateway encapsulates the message and pass it to the API Then the message pass through processAccountChange channel. At this point, the system notifies the Customer depending on the Customer Notification subscription (Email, SMS) The AccountChangeRouter dynamically decides Email and or SMS channel(s) should receive the Message Then the Transformer transforms the Account message to EmailText message from mailChannel to transformedAccountToEmail channel for JMS And then the bridge broadcasts the message from transformedAccountToEmail to restEmailSender channel. The Rest Email app will listen to for EmailText on JMS restEmailQueue The Rest Email accept the email and forwards outgoing e-mail for delivery

4.1 Interaction Diagram

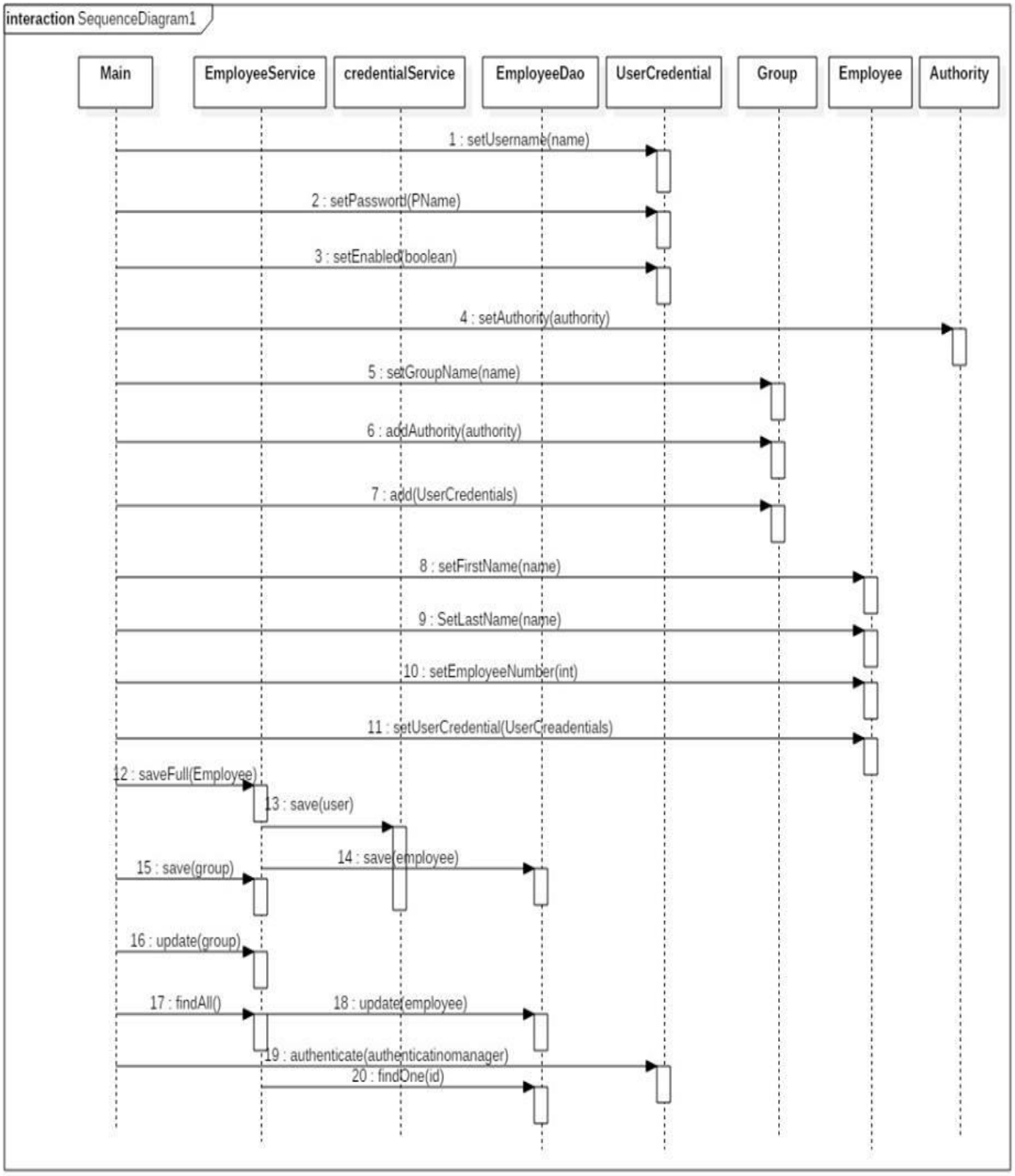

interaction SequenceDiagram1
Main
EmployeeService
credentialService
EmployeeDao
UserCredential
Group
Employee
Authority
1 : setUsername(name)
2 : setPassword(PName)
3 : setEnabled(boolean)
4 : setAuthority(authority)
5 : setGroupName(name)
6 : addAuthority(authority)
7 : add(UserCredentials)
8 : setFirstName(name)
9 : SetLastName(name)
10 : setEmployeeNumber(int)
11 : setUserCredential(UserCreadentials)
12 : saveFull(Employee)
13 : save(user)
14 : save(employee)
15 : save(group)
16 : update(group)
17 : findAll()
18 : update(employee)
19 : authenticate(authenticatinomanager)
20 : findOne(id)

Figure 5 REST Security sequence diagram

We pretty much defined the interaction on the sequence diagram, to more clarify we started with the main class in which all our four domain objects relation with their service method was written. And then saves their data to data base. The flow of persistence starts from the main class and passes the data to service and finally to Data access object which will connects with our data base.

5. Design Alternatives

We used spring group based authentication system but as an alternative we can use ACL attribute based authentication.

The messaging system can be change by Rabbit MQ instead of JMS because JMS is java dependent.

6. Issues, Risk and Dependencies

Issued by Yohanes:

- Persist LocalDate and LocalDateTime with JPA has some difficulties of this two technologies.
  Solved by:
  Applying an auxiliary class that converts from java LocalDate to database Date by accessing @Convert annotation.
  http://www.thoughts-on-java.org/persist-localdate-localdatetime-jpa/

- Problem when JSON creates Account class which is Abstract and had a variety of it. So here we need to have dynamic creation for the child's.

  Solved by applying:
  JacksonPolymorphicDeserialization Jackson JSON:
  https://github.com/FasterXML/jackson-docs/wiki/JacksonPolymorphicDeserialization

- Deserialize a list of objects of subclasses of an abstract class
  http://www.davismol.net/2015/03/05/jackson-json-deserialize-a-list-of-objects-of-subclasses-of-an-abstract-class/

- The class AbstractEntity run us into problem because of we don't need it to be mapped in database, hence I didn't specify the database mapping and all the classes inherit from it it runs in to error.
  Solved (by Tamrat): Adding the @MappedSuperclass annotation

While Adding @cacheble() and speciy ehcache.xml I run to error but not solved

Issued by Esehak:

The issues that I had was, getting information to implement ACL there are only few small piece of codes which makes a bit challenging to implement it within two days, some of the generally challenges while I am doing the project was to configure security configuration and connect it with application context, which I finally figured out after I read from the internet.

Issued by Tamrat:

- When running rabbitmq and activemq on the same machine, port conflict problem occurs in windows10

  Exception java.net.BindException: Address already in use: JVM_Bind will be thrown.

  The problem is both brokers use the 5672 port (amqp default port).

  Soln, just change one of the broker's port.

  RabbitMQ uses port 5672 for RABBITMQ_NODE_PORT to configure cluster the node to run multiple RabbitMQ nodes

  ActiveMQ uses port 5672 to support AMQP.

  It is enabled in the default ActiveMQ server configuration.

  you should look for the file name activemq.xml in the conf folder located in the ActiveMQ server installation directory.

  Eg :- C:\Program Files\activemq\conf

  So, change 5672 to any port that is not used by the OS will solve the problem.

  - Reference

    https://www.rabbitmq.com/configure.html
    http://activemq.apache.org/amqp.html

  Soln

  @MappedSuperclass

  Use the @MappedSuperclass annotation on the superclass of the embeddable class you're mapping just like you would for an entity. Subclasses will inherit the properties of this class as persistent properties. The ORM will Map the properties to the subclasses.

  But the superclass will not have a database table representation

- To get "Gateway" reference from ApplicationContext in web application.

  Soln
  I think, the problem was Spring integration xml (Gateway, broker, queue...) configuration file using import in the applicationContext.xml was not correct.

  ```
  <import resource="classpath:context/integration/common.xml"/>
  <import resource="classpath:context/integration/accountChangeGateway.xml"/>
  ```

  Now the configuration is done in web.xml and modified to

  ```
  <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
  classpath*:**/applicationContext.xml
  classpath*:**/securityContext.xml
  classpath*:**/integration/common.xml
  classpath*:**/integration/accountChangeGateway.xml
  </param-value>
  </context-param>
  ```

  Finally, I added the below code snippet to the AccountChangeNotifyAspect

  ```
  @Autowired
  ApplicationContext applicationContext;

  AccountChangeGateway accountChangeGateway = (AccountChangeGateway)
  applicationContext.getBean("accountChange");
  ```

  In Spring message integration, jmsoutRestEmail adapter doesn't receive the message. Definately, I forgot something in the configuration step.

7. Future Considerations

REST security system can be enhanced by using RBAC

8. References

- Text book: Spring Security 3

- Spring Security tutorial
  http://krams915.blogspot.com/2011/01/spring-security-3-full-acl-tutorial.html
- Text book: Spring REST
  By Balaji and Sudha
- Java persistence with hibernate Second Edition 2015
  By: Chirstian Baur, Gavin King, Gary Gregory
- Spring Framework Reference Documentation
  http://docs.spring.io/spring/docs/5.0.0.M3/spring-framework-reference/htmlsingle/