PROJECT REPORT ON

A WEB MODULE FOR CLEARING SYSTEM TO MANAGE PLEDGE PROCESS

Submitted in partial fulfilment of the requirements for
The award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
OF
SASTRA UNIVERSITY

Submitted by

MOHAMEED MUSTHAFA - 117003121

PATEL VAMSHI KRISHNA - 117003137

Under the Guidance of

VIJAYPRAKASH SODISETTI
INAUTIX TECHNOLOGIES Pvt. Ltd, CHENNAI

SCHOOL OF COMPUTING

SHANMUGHA
ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY
(SASTRA UNIVERSITY)
(A University Established under section 3 of the UGC Act, 1956)
TIRUMALAISAMUDRAM
THANJAVUR – 613 401
April 2017

SCHOOL OF COMPUTING

SHANMUGHA
ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY
(SASTRA UNIVERSITY)
(A University Established under section 3 of the UGC Act, 1956)
TIRUMALAISAMUDRAM, THANJAVUR – 613401

BONAFIDE CERTIFICATE

Certified that this project work entitled "**A WEB MODULE FOR CLEARING SYSTEM TO MANAGE PLEDGE AND RELEASE PROCESS**" submitted to the Shanmugha Arts, Science, Technology & Research Academy (SASTRA University), Tirumalaisamudram-613401 by **MOHAMEED MUSTHAFA** with Reg no.**117003121**, **PATEL VAMSHI KRISHNA** with Reg no.**117003137** in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** is the original and independent work carried out under my guidance, during the period December 2016 - April 2017**.**

EXTERNAL GUIDE	**ASSOCIATE DEAN**
VIJAYPRAKASH SODISETTI	**Dr. A. UMAMAKESWARI**
INAUTIX TECHNOLOGIES Pvt Ltd, CHENNAI	**SCHOOL OF COMPUTING**

Submitted for University Examination held on_____

EXAMINER - I **EXAMINER - II**

SCHOOL OF COMPUTING

SHANMUGHA
ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY
(SASTRA UNIVERSITY)
(A University Established under section 3 of the UGC Act, 1956)
TIRUMALAISAMUDRAM, THANJAVUR – 613401

DECLARATION

We submit this project work entitled "**A WEB MODULE FOR CLEARING SYSTEM TO MANAGE PLEDGE PROCESS**" to the Shanmugha Arts, Science, Technology & Research Academy (SASTRA) University, Tirumalaisamudram–613 401, in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** and declare that it is our original and independent work carried out under the guidance of **VIJAYPRAKASH SODISETTI**, iNautix Technologies, Chennai.

Date :
Place :

Name :MOHAMEED MUSTHAFA
Reg. No:117003121

Signature:

Name :VAMSHI
Reg. No:117003137

Signature:

ACKNOWLEDGEMENT

First and Foremost, we take pride in thanking the almighty to gave us strength for the successful completion of this project.

We have immense pleasure in expressing my heartfelt thanks to our Vice-Chancellor **prof.R.Sethuraman** for the benevolent advice and guidance during my tenure in the college.

We would like to express our gratitude to **Dr.Balachandran, Registrar,** SASTRA UNIVERSITY, for his benevolent guidance through-out my college life.

We wish to express our thanks to **Dr.S.Vaidhyasubramanian, Dean of Planning and Development** for his encouragement and providing us with all the needed amenities.

We would like to express our gratitude to **Dr.P.Swaminathan,** Dean **,** School of Computing, SASTRA UNIVERSITY, for his benevolent guidance through-out my college life.

We would like to express our gratitude to **Dr.A.Umamakeshwari,** Associate Dean **,** Computer Science and Engineering, School of Computing, SASTRA UNIVERSITY, for her benevolent guidance through-out my college life.

We wish to express our thanks to **Prof. Kamakshi . S,** Associate Professor Computer Science and Engineering, School of Computing, SASTRA UNIVERSITY for her encouragement and providing us with all the needed amenities.

we deeply thank our family and friends for supporting me in all the tasks that I have carried for the successful completion of this project.

LIST OF TABLES USED IN THIS PROJECT

TABLE OF FIGURES USED IN THIS PROJECT

TABLE OF CONTENTS

ABSTRACT

The purpose of this project is to build a web based system that will allow end clients to initiate their pledge and release process. Currently Clients are handling their own collateral funding and pledging process using various tools and platform, and they are using multiple tools to view securities that are available for pledging and release.

As per this web based project, the client will be able to view / perform the securities in a consolidated portal by which they can perform their pledging process for the Clearing house. Client will be using the Front end screens to view the securities for pledging and releasing. Appropriate levels of access are regulated via entitlements. No clients will be having direct control over pledge and release process. There will be an advisor / broker / relationship manager who takes care of all the client related process (pledge & release). This clearing house system mainly focuses on complete settlement policies between the two parties in a transaction. Some transaction may get cleared in two days, some may take more than two days which depends on their pledging policies, usually it is termed as T+2, T+3 days transaction. The complete clearance and settlement of the securities and the proof of transaction will be provided to both the parties. The transacted information can be viewed via this portal.

The archival data for pledging and releasing will be maintained for 10 years. The Pledging and Releasing Screen will be available 24/7. Action can only be taken on US business days from 6AM – 6 PM EST. During bank holidays, client can view their account holdings but no further action can be taken from the screens.

1) INTRODUCTION

EXISTING SYSTEM:

DonorSnap provides you with tools to manage the pledge process from start to finish. A donation forecast report is available to help an organization try to estimate incoming receipts based upon initial pledge information. But the system is globally available and are not specific to the requirements of our client.

PROPOSED SYSTEM:

In the proposed system, we create a standalone pledger management for our clients. It is easy to create a Pledge within the system and then to apply subsequent donation receipts to the pledge. The system allows you to monitor the status of a specific pledge or to view all pledge activity as a whole on printed reports.

The main purpose is to create a **Pledge Management** page which is used by administrators and staff members to manage pledge records. From this page, you can view clients who have pledged to a specific fund, add and delete pledge records, view pledge details, and make adjustments to existing pledges. The organization administrator can see all funds and their associated pledges while staff members can see only those pledges associated with the funds they granted permission to access. Every clients will be having their own broker dealer and every dealer will have their own unique IBD(Broker Dealer ID). Using that IBD, clients can either pledge or release shares and securities.

2) PROBLEM STATEMENT

A Stock Pledge is nothing but a transfer of stocks against a debt. It is an agreement between the client and the bank. The debtor pledges the shares as an asset against the amount of money taken from a lender and promises to return the amount within specific period. The debtor pledges the stocks as a security against the debt. According to the law, after the payment of the obligation the bank in which stocks are ledged must return the stocks to the debtor and the agreement stands void.

In Early, it was very difficult to manage this pledge process for the individual clients. Many components are needed to check the status of the individual user's pledge process. To view the pledge details, the client has to navigate to different modules which are tedious process. Since, many components are needed to check the status of the individual user's pledge process, the convenience of the user and the response time is affected.

In order to overcome all the burden of the user, a single page is developed to fetch all the details from various components and displayed in the required format. User can specify their own requirements and can change the columns to be displayed. Every shares and securities associated with each banks for the particular broker id can be viewed with great ease. The pledging process can be scalable for existing and new client.

3) Literature Review

A clearing house is intermediary between buyers and sellers of financial services further, it is the agency or Separate Corporation of futures exchange responsible for settling trading accounts, collecting and maintaining margin monies, clearing trades, regulating delivery, and reporting trading data. Clearing houses acts as third parties to all the futures and options contracts, as buyers to every clearing member seller, and as sellers to every clearing member buyer.

Here a broker called clearing broker is involved. A clearing broker is the member of exchange that acts as liaison between an investor and clearing corporation. A clearing broker helps the client to ensure that trade is settled appropriately and the transaction is always successful. Clearing brokers have the job of maintaining the paper work associated with clearing and executing of transaction.

4) SOFTWARE/HARWARE REQUIREMENT SPECIFICATION

ESTABLISHING CONNECTION WITH BACKEND MAINFRAME SERVER FROM FRONT END USING P-LINK AS MIDDLEWARE

4.1. HARDWARE REQUIREMENT SPECIFICATION:

4.1.1. Client Side:

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirement list is often accompanied by a Hardware Compatibility List (HCL), especially in case of operating systems. An HCL lists tested, compatible and sometimes incompatible hardware devices for a particular operating system or application.

Processor	Intel core 2 duo and advance
Speed	2.0 GHz
Hard Disk Drive	250 GB and above.
Operating System	Windows, linux
Memory	2 GB RAM and above
System Type	32,64 bit Operating System

Table: 4.1.Hardware Specification

4.1.2. Server Side:

Mainframe:

A Mainframe Computer is a high performance Multi User computer system which is the most scalable, available, reliable and secured machine in the world capable of performing some Million Instructions per second (up to 569,632 MIPS).

Characteristics:

1) Reliable single-thread performance

2) Maximum I/O connectivity

3) Maximum I/O bandwidth

4) Reliability, Availability & Serviceability (RAS)

5) Unbreakable Security & Scalability (USS)

Z13 Mainframe:

The z13 processor has faster I/O and the ability to address up to 10144 GiB of RAIM memories -- three times as much as its predecessor. It can house up to 168 processor units in a single system and run as many as 8,000 virtual servers.

At a maximum 5GHz, the z13's processor is slower in terms of clock speed than the chip in the z12, but IBM says it more than compensates for that with other improvements. The chip has eight cores compared with six for its predecessor, and it's manufactured on a newer, 22 nanometer process, which should mean smaller, faster transistors.

The z13 chip is IBM's first mainframe processor to incorporate simultaneous multi-threading, allowing it to execute two instructions in parallel on each clock cycle. SMT has long been offered on x86 chips and more recently with IBM's Power processors, but the company hasn't added SMT to its mainframe chips in the past because the zOS mainframe software isn't written to take advantage of it.

The z13 also gets support for SIMD (single instruction, multiple data), another parallel computing technology long supported by x86 chips, which should speed up the mainframe's analytics capabilities. Analytics also get a boost from DB2 BLU for Linux, an in-memory database technology now coming to the mainframe.

With the ability to have 10TB of memory with large memory frames/pages, the practice of bringing analytics to the data (rather than copying the data, via ETL, to the analytics) has more relevance than ever before. z13 is designed for cloud, mobile and advanced analytics capabilities, which is a function not only of speed but of design.

The z13 supports up to 8000 Linux images simultaneously for cloud computing. For the mobile economy the z13 does real-time encryption and can process 2.5 billion transactions per day. For the z13, IBM spent over 1 billion dollars and five years of development and with more than 500 new patents.

The following table shows the specifications of IBM z13 Mainframe

Available	March 9,2015
Memory	Up to 10TB
Number Of Models	5-NE1,NC9,N96,N63,N30
Channels	-PCIe Gen3 16GBps channel buses -SIX CSSs, upto 85 LPARs -4 Sub channels sets per CSS -Flash Express
Operating Systems	z/OS, z/VM, z/VSE, z/TPF, Linux on z Systems

Table:4.2. Mainframe Specification

The above mentioned configuration of mainframe is of high level standards. Apart from that, the system can run under the following mainframes:

-Z/800

-Z/890

-Z9

4.2. SOFTWARE REQUIREMENTS SPECIFICATION

4.2.1 npm and node js Installation

Node.js and npm are essential to modern web development with Angular2 and other platforms. Node powers client development and build tools. The npmpackage manager, itself a node application, installs JavaScript libraries.

Version:

Node js v4.x.x or higher

npm 3.x.x or higher

The versions of node and npm can be checked using following commands :

node -v

npm –v

Older versions produce errors.

After installing node and npm, the following steps are performed

1. Create a project folder named quickstart.
2. Clone the Quick Start seed into project folder.
3. Install npm packages.
4. Run npm start to launch the sample application.

git clone https://github.com/angular/quickstart.git quickstart

The above command is given in GIT bash console to clone the quickstart folder from the GitHub link. We can also download it separately and save it in a folder.

cd quickstart

Switch to the quickstart folder

npm install

Install all the packages necessary to run Angular JS 2 application

npm start

Once the command is executed, the npm server is started and https://localhost:3000 path is set by default.

4.2.2 Visual studio code editor

VS code is a new type of tool which combines the simplicity of a code editor with what developers need for their core edit-build-debug cycle. Code provides comprehensive editing and debugging support, an extensibility model, and lightweight integration with existing tools.

The general code structure looks like:

-src

-app

-app.component.ts

-app.module.ts

-main.ts

app.component.ts :

It is the root component of what will become a tree of nested components as the application evolves.

app.module.ts:

Defines AppModule, the root module that tells Angular how to assemble the application. Right now it declares only the AppComponent.

main.ts:

Compiles the application with the JIT compiler and bootstraps the application's main module (AppModule) to run in the browser.

The JIT compiler is a reasonable choice during the development of most projects and it's the only viable choice for a sample running in a *live-coding* environment.

The following is the small snippet of code which is used to display the table in front end using Angular JS 2 technology:

app.component.js

```
angular.module('ngTableTutorial', ['ngTable'])

    .controller('tableController', function ($scope, $filter, ngTableParams) {


$scope.records=/*data fetched from rest services*/;

    $scope.usersTable = new ngTableParams({

        page: 1,

        count: 10

      }, {

          total: $scope.records.length,



getData: function ($defer, params)
```

```
            {
              $scope.data=params.sorting()? $filter('orderBy')($scope.records, params.orderBy()) : $scope.records;
              $scope.data = $scope.data.slice((params.page() - 1) * params.count(), params.page() * params.count());
              $defer.resolve($scope.data);
            }
          });
        });
```

ngtable.html

```
<div ng-controller="tableController">
<table ng-table="releaseTable" showfilter=”true” class="table table-striped">
  <tr ng-repeat="row in data">
    <td data-title="'CUSIP'" >
      {{row.cusip}}
    </td>
    <td data-title="'SYMBOL'" >
      {{row.symbol}}
    </td>
    <td data-title="'DESCRIPTION'" >
      {{row.description}}
    </td>
    <td data-title="'RELEASE QUANTITY" >
      {{row.release_quantity}}
    </td>
    <td data-title="'PLEDGE QUANTITY'">
```

```
        {{row.pledge_quantity}}
      </td>
      <td data-title="'PLEDGE  VALUE'" >
        {{row.pledge_value}}
      </td>
      </td>
      <td data-title="'PLEDGE  BANK'" >
        {{row.pledge_bank}}
      </td>
    </tr>
</table>
</div>
```

4.2.3 Bootstrap framework

Bootstrap is an open-source collection of tools which is used in creating websites and web applications. It contains HTML and CSS based design templates for forms, buttons, and navigation and so on, as well as optional Java Script extensions. This aims to ease the development of dynamic websites and also web applications.

Bootstrap is a front end web framework which is an interface for the user, unlike the server-side code which lies on the "back end" or server. Bootstrap comes with several JS(JavaScript) components in the form of jQuery plugins. They provide additional UI(user interface) elements such as dialog boxes, tooltips, and so on. They also extend the functionality of existing interface elements, including auto-complete function for input fields. The following JavaScript plugins are supported by bootstrap: Dropdown, Scroll spy, Modal, Tab, Tooltip, Popover, Collapse, Alert, Button, Carousel and Type ahead.

ADVANTAGES OF BOOTSTRAP

- Ease of Use
- Highly Flexible
- Responsive Grid
- Comprehensive List of Components
- Leveraging JavaScript Libraries
- Frequent Updates
- Detailed Documentation and Vast Community
- Consistency

4.2.4 Interfacing Client FrontEnd and RESTful Services Using Spring MVC framework:

SPRING MVC

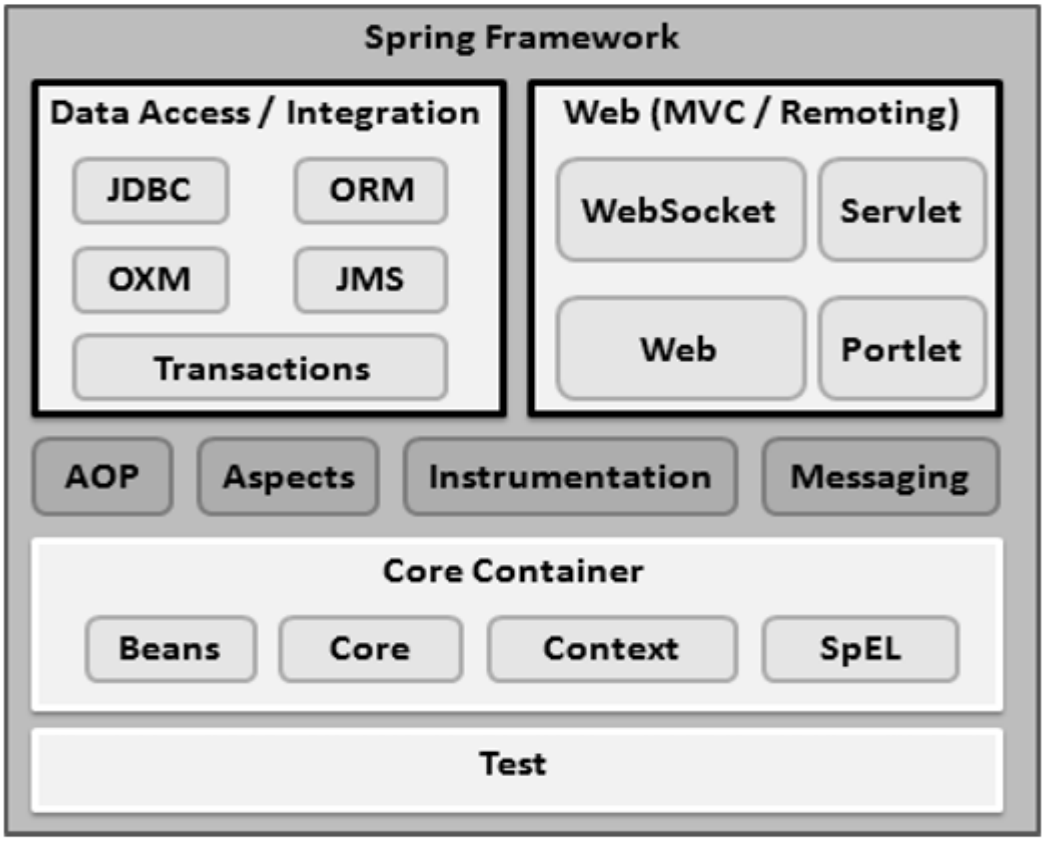


Fig: 4.1. Spring Framework

The spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

Model - encapsulates the application data and in general they will consist of POJO.

View - responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.

Controller - responsible for processing user requests and building appropriate model and passes it to the view for rendering.

The below code is responsible for invoking rest services from Angular 2:

```
import {Injectable, Injector} from '@angular/core';
import {HTTP_PROVIDERS, Http, Request, RequestMethod} from '@angular/http';

@Injectable()
class AutoAuthenticator {
  constructor(public http:Http) {}
  request(url:string) {
    return this.http.request(new Request({
      method: RequestMethod.Get,
      url: url,
      search: 'password=123'
    }));
  }
}

var injector = Injector.resolveAndCreate([HTTP_PROVIDERS, AutoAuthenticator]);
var authenticator = injector.get(AutoAuthenticator);
authenticator.request('people.json').subscribe(res => {
  //URL should have included '?password=123'
  console.log('people', res.json());
});
```

As shown in the code, the AutoAuthenticator class which calls the rest service using the url and gets the list of records in JSON format and return it to the called method. This result is printed using console.log () method.

4.2.5 Interfacing RESTful Services and Backend Mainframe Server Using Message Queuing:

The following are the configuration files used :

Pom.xml

Spring-Servlet.xml

Web.xml

Pom.xml:

This file contains all the dependencies that are required to get the application work. The code dependencies and the properties tag are given below.

<project xmlns="*http://maven.apache.org/POM/4.0.0*"
xmlns:xsi="*http://www.w3.org/2001/XMLSchema-instance*"

xsi:schemaLocation="*http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd*">

<modelVersion>4.0.0</modelVersion>

<groupId>pledge</groupId>

<artifactId>pledge.test</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>Test Pledge Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<jdk.version>1.7</jdk.version>

```
        <spring.version>4.1.1.RELEASE</spring.version>
        <jstl.version>1.2</jstl.version>
        <junit.version>4.11</junit.version>
        <logback.version>1.0.13</logback.version>
        <jcl-over-slf4j.version>1.7.5</jcl-over-slf4j.version>
</properties>
<dependencies>
        <!-- Unit Test -->
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>${junit.version}</version>
</dependency>
        <!-- Spring Core -->
<dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
                <version>${spring.version}</version>
                <exclusions>
                        <exclusion>
                                <groupId>commons-logging</groupId>
                                <artifactId>commons-logging</artifactId>
                        </exclusion>
                </exclusions>
</dependency>

<dependency>
```

```
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${jcl-over-slf4j.version}</version>
</dependency>

<dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
</dependency>
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
</dependency>
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
</dependency>

        <!-- jstl -->
<dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
</dependency>
<dependency>
```

```
                <groupId>org.springframework</groupId>
                <artifactId>spring-jdbc</artifactId>
                <version>${spring.version}</version>
        </dependency>
        <!-- Jackson JSON Processor -->
         <dependency>
                <groupId>com.fasterxml.jackson.core</groupId>
                 <artifactId>jackson-databind</artifactId>
                <version>2.4.1</version>
        </dependency>
</dependencies>
<build>
   <finalName>Test</finalName>
  </build>
</project>
```

Maven Dependency :

Maven is a powerful tool that allows users to import dependencies into their software projects and also automatically manage transitive dependencies. In order to use Maven, it is necessary to explicitly add dependencies to the Maven pom.xml file. Once added to the Maven pom.xml file, dependencies will be automatically downloaded, updated, and have their sub-dependencies managed by Maven.

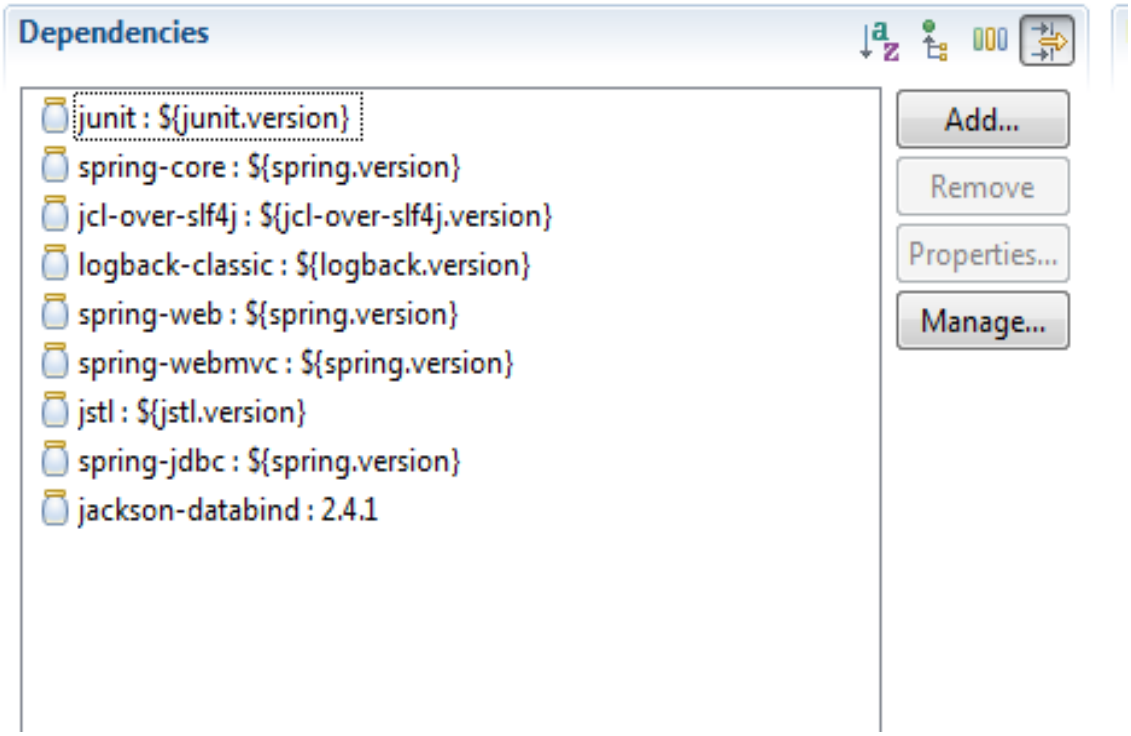


Fig: 4.2. Dependency injection

The above screenshot contains some of the core dependency injected and each one is responsible for importing some specific packages. For example, Jackson-databind will import the packages at run time which will convert list into JSON objects and vice versa.

Spring-servlet.xml:

<beans>

xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.1.xsd

http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"

</beans>

The above shown configuration file contains a beans tag which is used to download the required packages from the url provided.

5) Conceptual Model / Proposed Architecture

FLOW DIAGRAM:

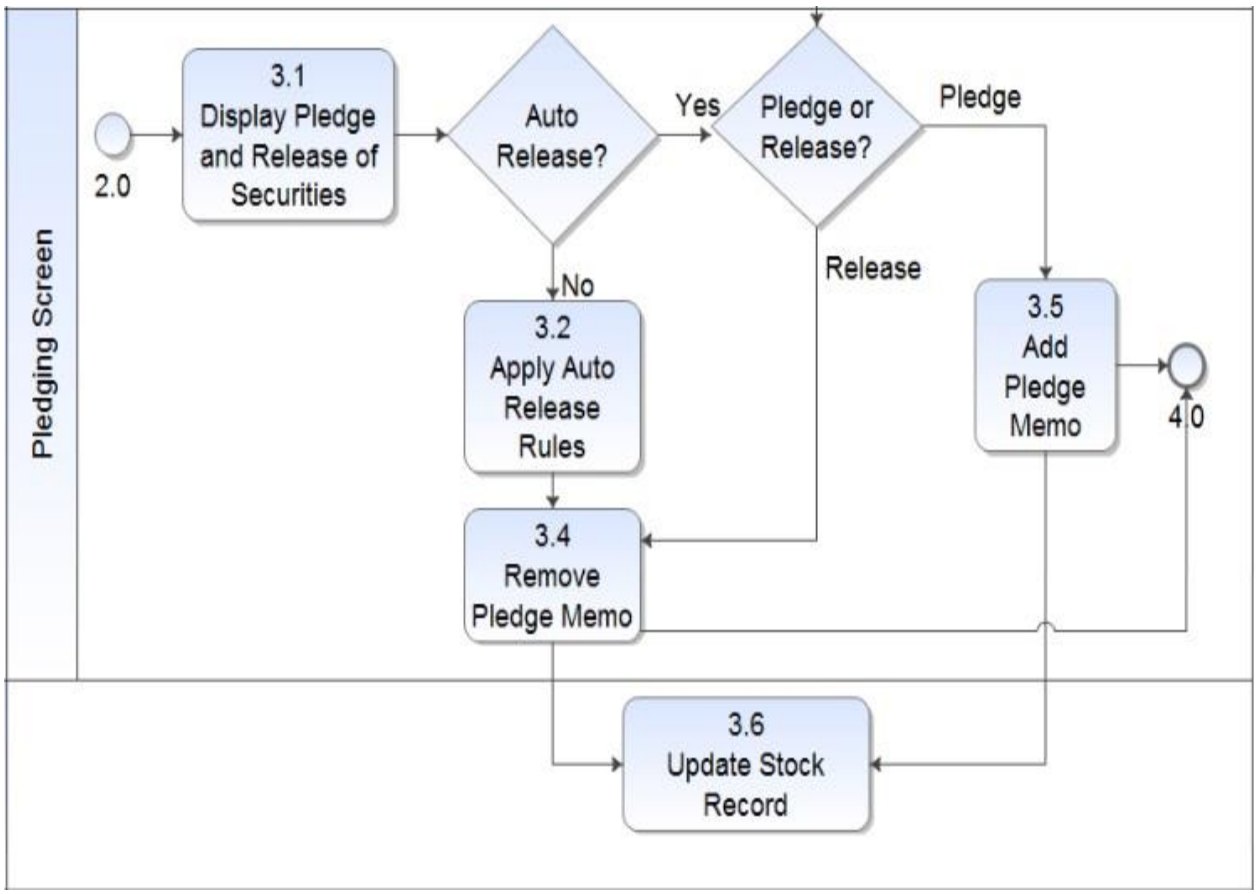


Fig 5.1: Flowchart of Pledging Screen

The flowchart shown above depicts how the pledging and releasing process works. In the first place, the broker selected by the client sees the pledge and release security records under each bank. There is an option of Auto Release. In Auto Release, After 24 hours of time, the pledged amount will be automatically released and the amount is deducted from the client account. If Auto release is not applied, then the broker can release the pledged stock manually. After the above mentioned process is completed, the record is updated to view the changes made.

BLOCK DIAGRAM:

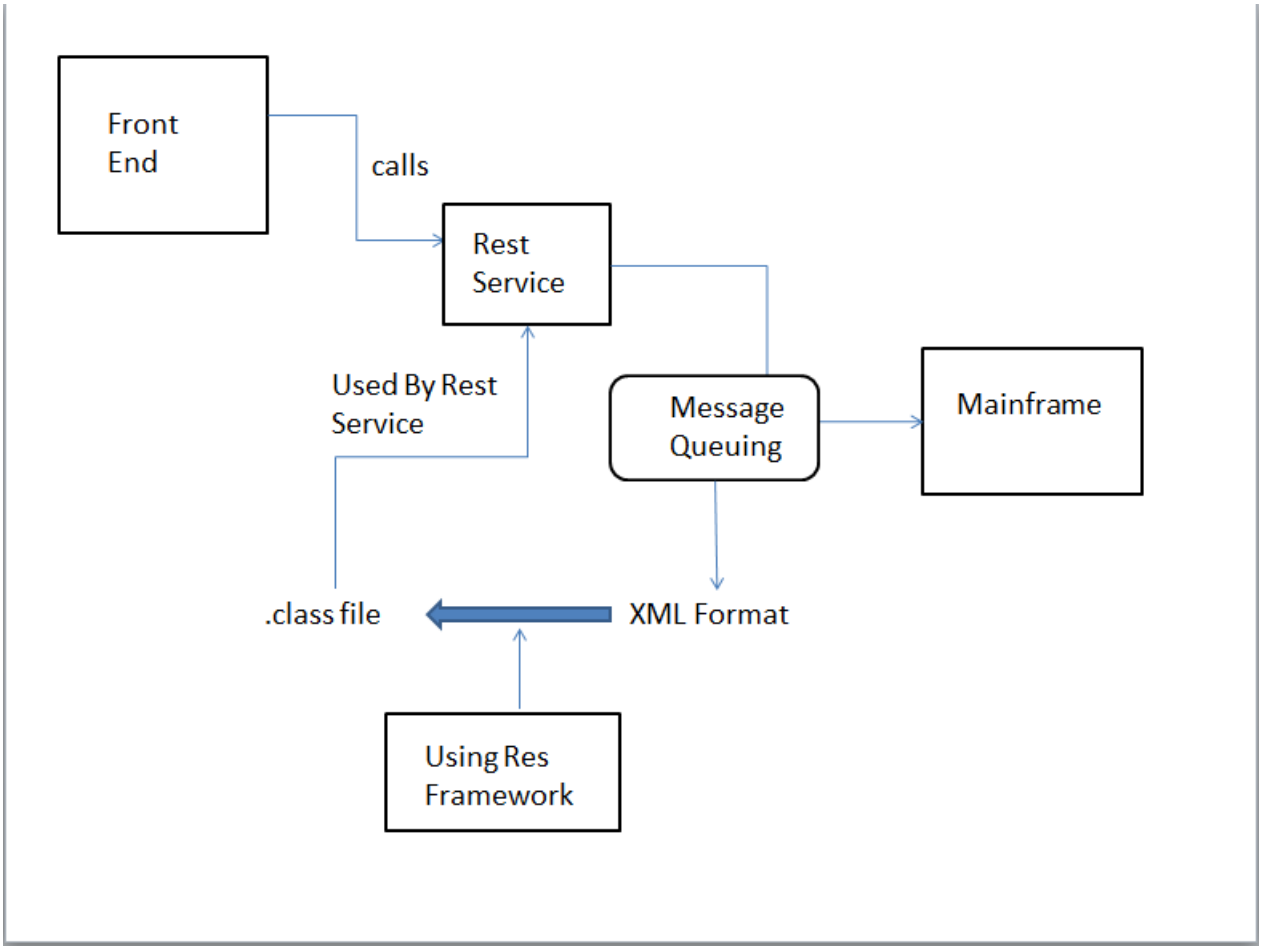


Fig:5.2 Block diagram showing flow of control from front end to back end mainframe

The above block diagram shows the flow in which the client interacts with the user interface. Initially, upon client request, the front end calls the rest service. The REST service also called as web service is used because of its low response rate when compared to SOAP API. To fetch data, REST API calls the Mainframe. Here message queuing is used to retrieve data from the mainframe, and the format is XML. A framework called Res Framework is used which generates equivalent java class file for the XML file. This class objects can be used by the initially invoked REST Service. The REST service returns the object in the JSON format. Angular JS 2 technology is implemented in viewing the retrieved data in single page application.

6) Interaction Scenario:

USECASE DIAGRAM

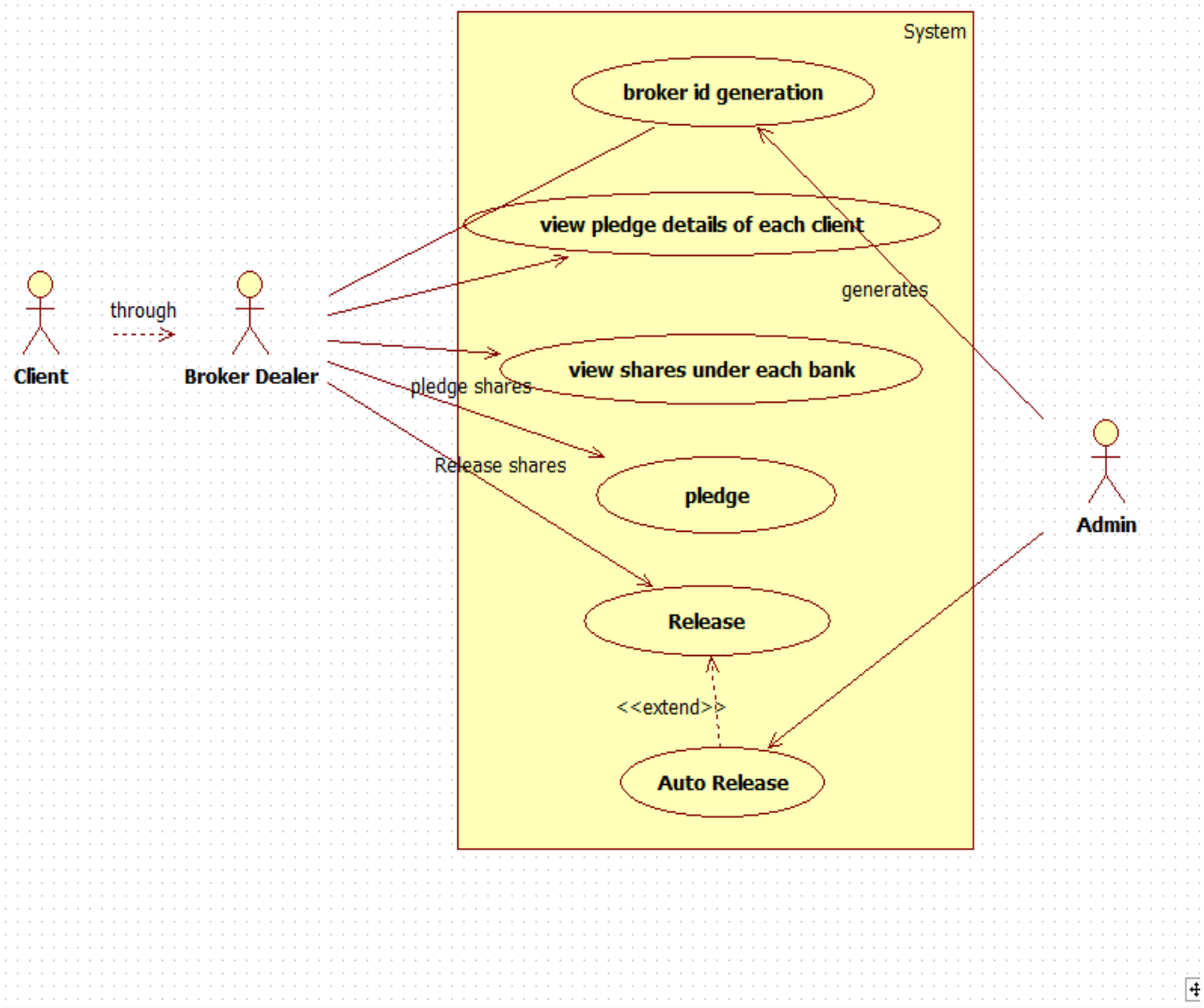


Fig: 6.1 : Use case Diagram for Pledging and Releasing

In this Use case diagram, client, broker and admin are the actors who are interacting with the application software. The client interacts with the application through a broker who acts as a mediator to pledging and releasing process.

For each broker a unique broker id will be generated which is used for authentication purpose. The broker not only views the pledge details of each client but also the shares which are under each bank. The broker has all the privileges to pledge the shares and release the pledged shares. Release of stocks can also happen automatically which is termed as Auto Release.

The admin's job is to generate the broker id and has all the rights to perform the Auto Release. The Auto Release will happen 24 hours after the stock is being pledged.

Auto Release process deducts the amount from the client account and release the shares correspondingly. If the required amount is not available in the client's account, penalty is given to the client.

Since, same transaction can be performed concurrently in two or more systems, some transaction may fail. For every unsuccessful process, an error popup message is thrown with corresponding error statements. The log of every unsuccessful transactions are saved and can be viewed later.

Use case Diagram Process flow:

1. Client to Broker Interaction.

2. Unique Broker id will be generated.

3. Broker views all the pledge details of each client.

4. Broker places the shares in the pledge.

5. Broker can also have the option to release the shares from the bank.

6. Finally there is an Auto release option to make the release of equities with 24 hours from the stocks pledged.

ACTIVITY DIAGRAM

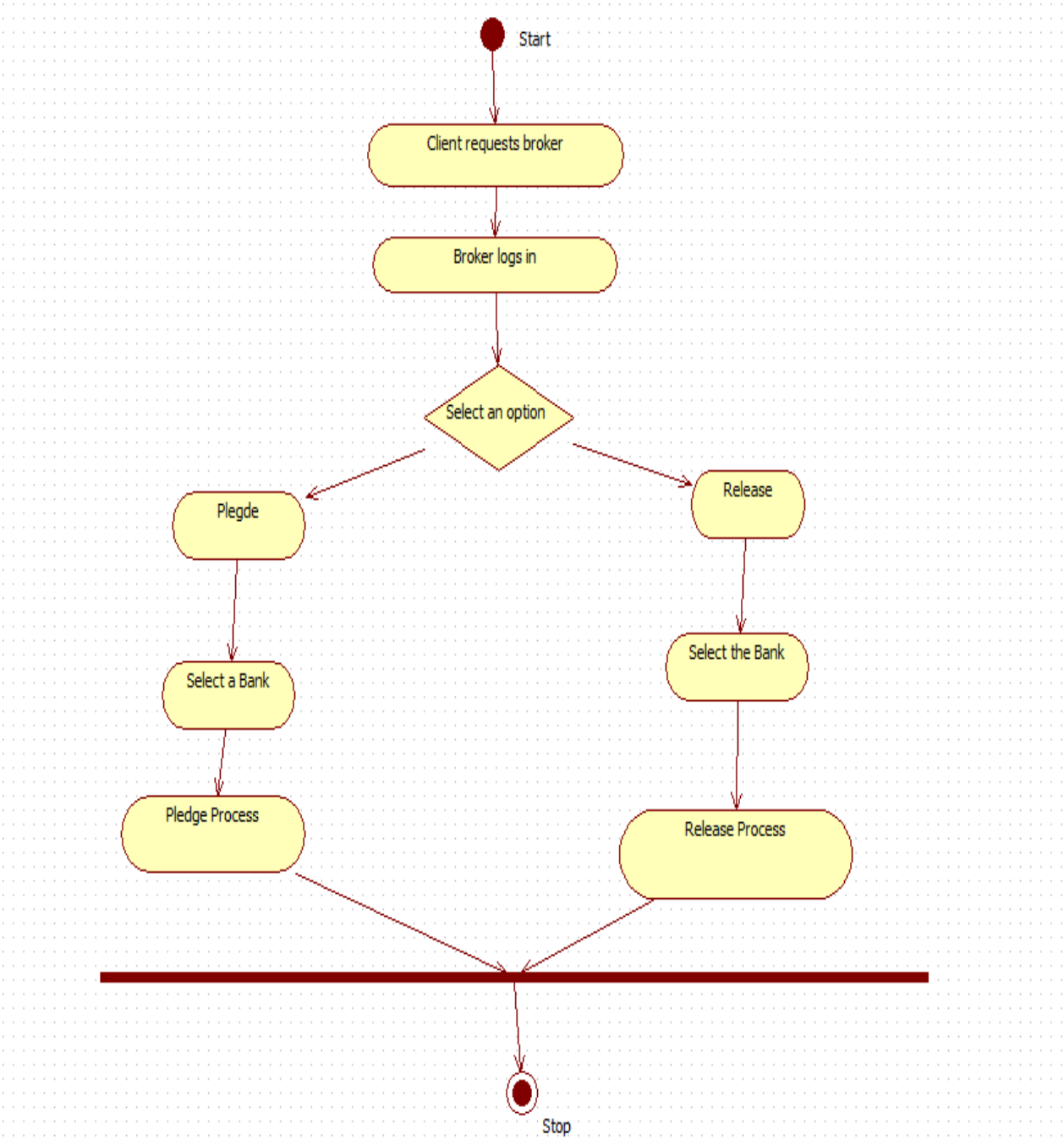


Fig 6.2(1) Activity Diagram for overall pledging and releasing process.

In this Activity diagram, basic flow of process starts from client requesting the broker to pledge or release the shares which the client is holding. Many clients can request one or more brokers for pledging and releasing process. The client will also give additional information to the broker like As soon as broker gets the information from the clients; he starts logging in to the application. The broker has an option to select pledge or do release. When pledge is selected, the broker then selects a bank from the list of banks available. All the banks will not encourage pledging process. Only few banks which are allowed are displayed. After selecting the bank, the quantity of shares to be pledged is given and pledge process is done.

When Release option is selected, the broker selects the banks for releasing the stocks from bank. Now all the stocks which belong to the client; that are held by the bank are given to Client.

Activity Diagram Process flow:

1. Client requests Broker and gives required information.

2. Broker logs in to the application.

3. If the Pledge option is selected, the available bank is listed down along with the details of all the client's shares under the specific bank.

4. In pledge process, the clients' shares are given to the bank and in return bank gives the money to the client based on share value.

5. If release option is selected, then broker selects the user share's CUSIP ID to be released and proceeds with Release process.

6. In release process the client pays the money back to the bank and takes back his shares. When failed to lend the money to the bank, then the bank puts penalty on the client which is to be paid.

ACTIVITY DIAGRAM FOR PLEDGING

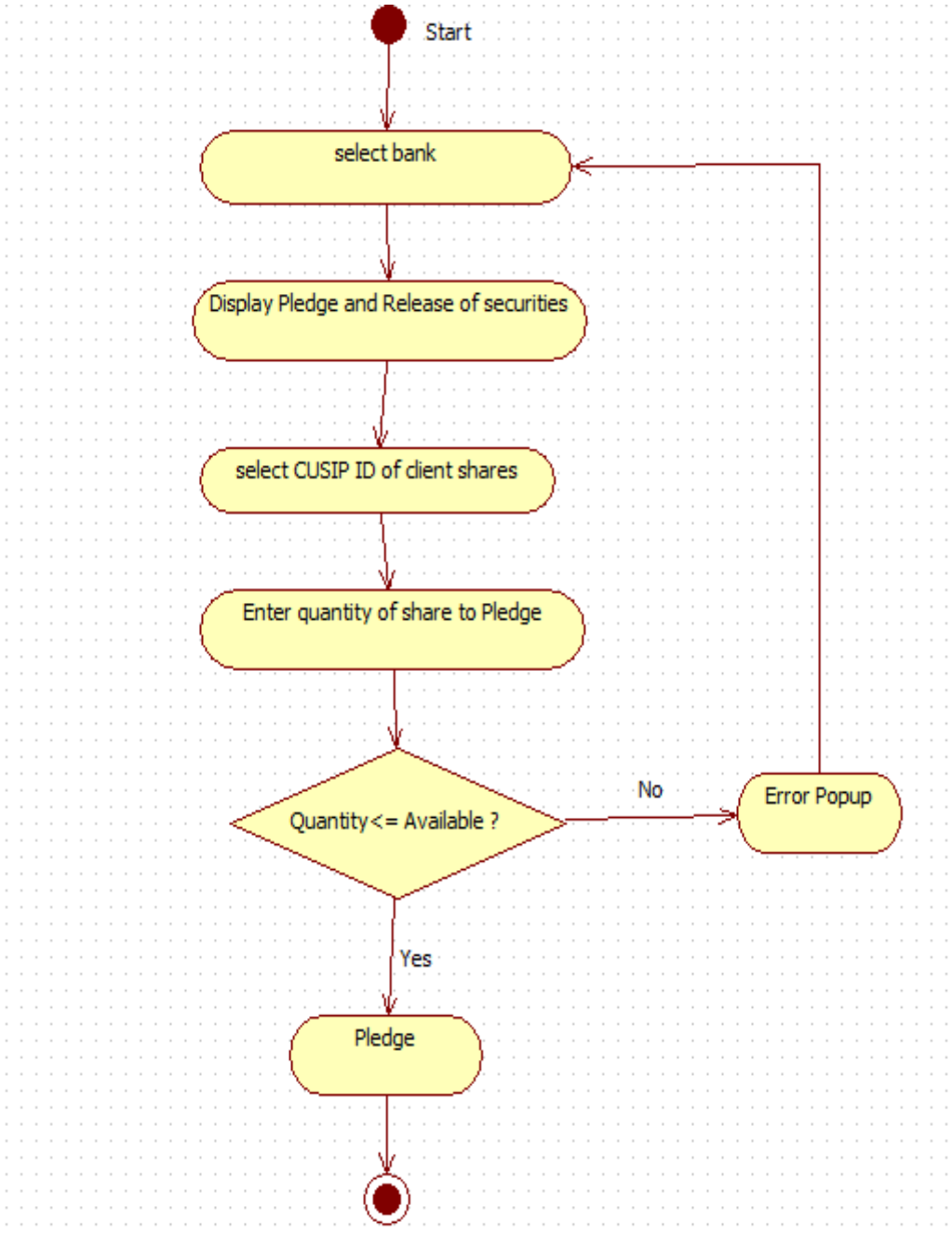


Fig 6.2(2) Activity Diagram for Pledging Process

Initially the broker selects the bank to perform the pledging process and the user interface displays the pledge and release security policies.

The Broker then selects the CUSIP id of the client shares and enters the quantity of each share to pledge. If the quantity of the shares to be pledged under the selected bank is less than or equal to the available quantity, then the stocks are successfully pledged and given to the bank.

Here, bank becomes the owner of the stocks and client borrows money from the bank. If the quantity is not available, then the application prompts the popup message stating insufficient quantity and it is not accepted.

Since, same transaction can be performed concurrently in two or more systems, some transaction may fail. For every unsuccessful process, an error popup message is thrown with corresponding error statements. The log of every unsuccessful transactions are saved and can be viewed later.

Activity Diagram process flow:

1. Broker selects the bank to perform pledging
2. Pledging and release securities are displayed.
3. CUSIP id of client shares is selected.
4. The quantity of the stock is entered.
5. If quantity of stock is available, then the stocks are pledged to bank.
6. Otherwise error popup is thrown and process is restarted.

ACTIVITY DIAGRAM FOR RELEASE:

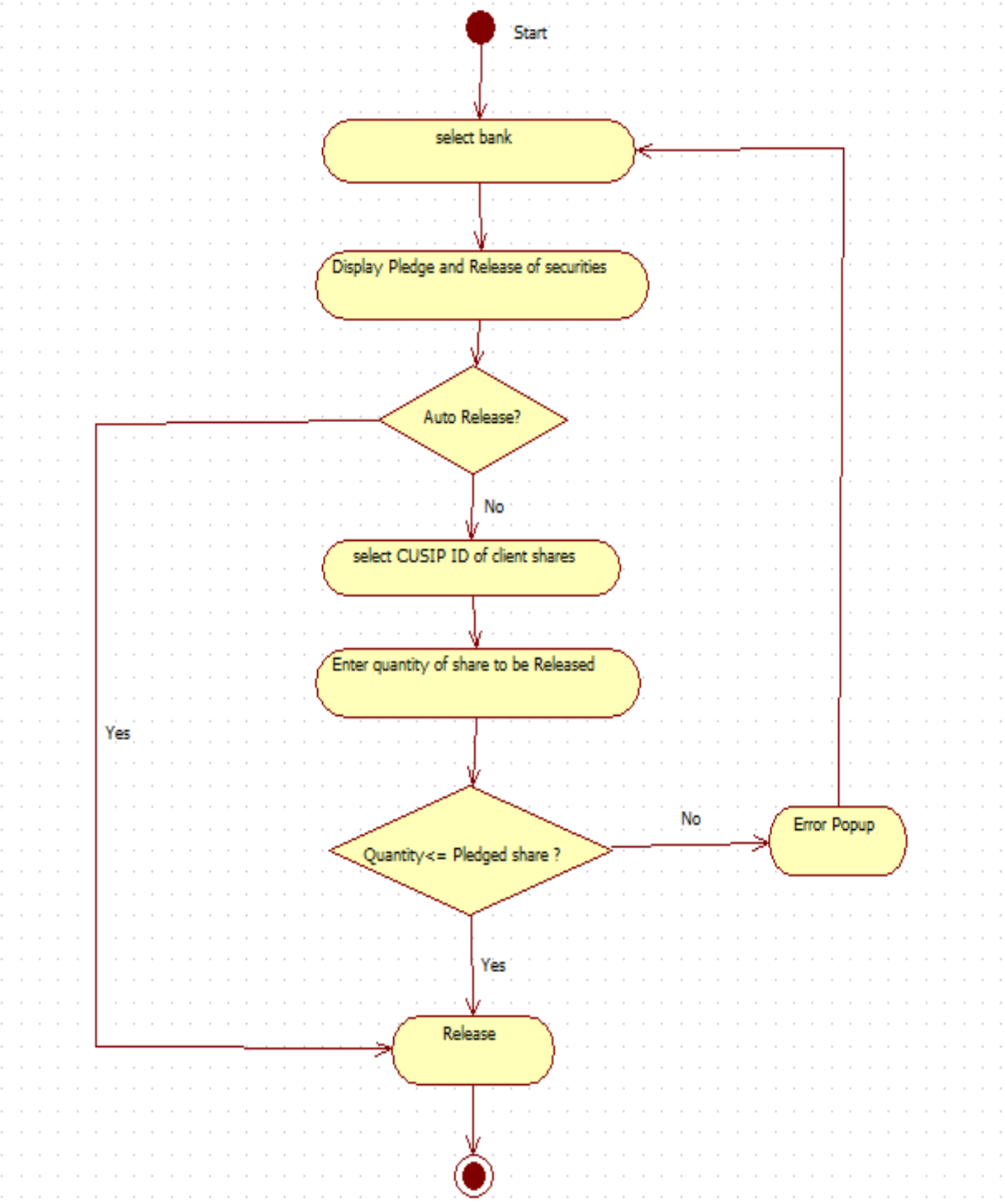


Fig 6.2(3) Activity Diagram for Releasing Process

Initially the broker selects the bank to perform the release process and the user interface displays the pledge and release security policies.

The Broker then selects the CUSIP id of the client shares and enters the quantity of each share to release. If the quantity of the shares to be released under the selected bank is less than or equal to the pledged quantity, then the stocks are successfully released and given back to the client.

Here, bank returns the stocks back to the owner upon receiving the money. Once the release process is done, the amount is deducted from the client's account depends on the number of share they released.

Since, same transaction can be performed concurrently in two or more systems, some transaction may fail. For every unsuccessful process, an error popup message is thrown with corresponding error statements. The log of every unsuccessful transactions are saved and can be viewed later.

Activity Diagram process flow:

1. Broker selects the bank under which the client has pledged their stocks.
2. Pledging and release securities are displayed.
3. CUSIP id of client share is selected.
4. The quantity of the stock to be released is entered.
5. If quantity of stock is valid, then the stocks are released and given back to the client.
6. The equivalent amount is deducted from the client's account.

SEQUENCE DIAGRAM:

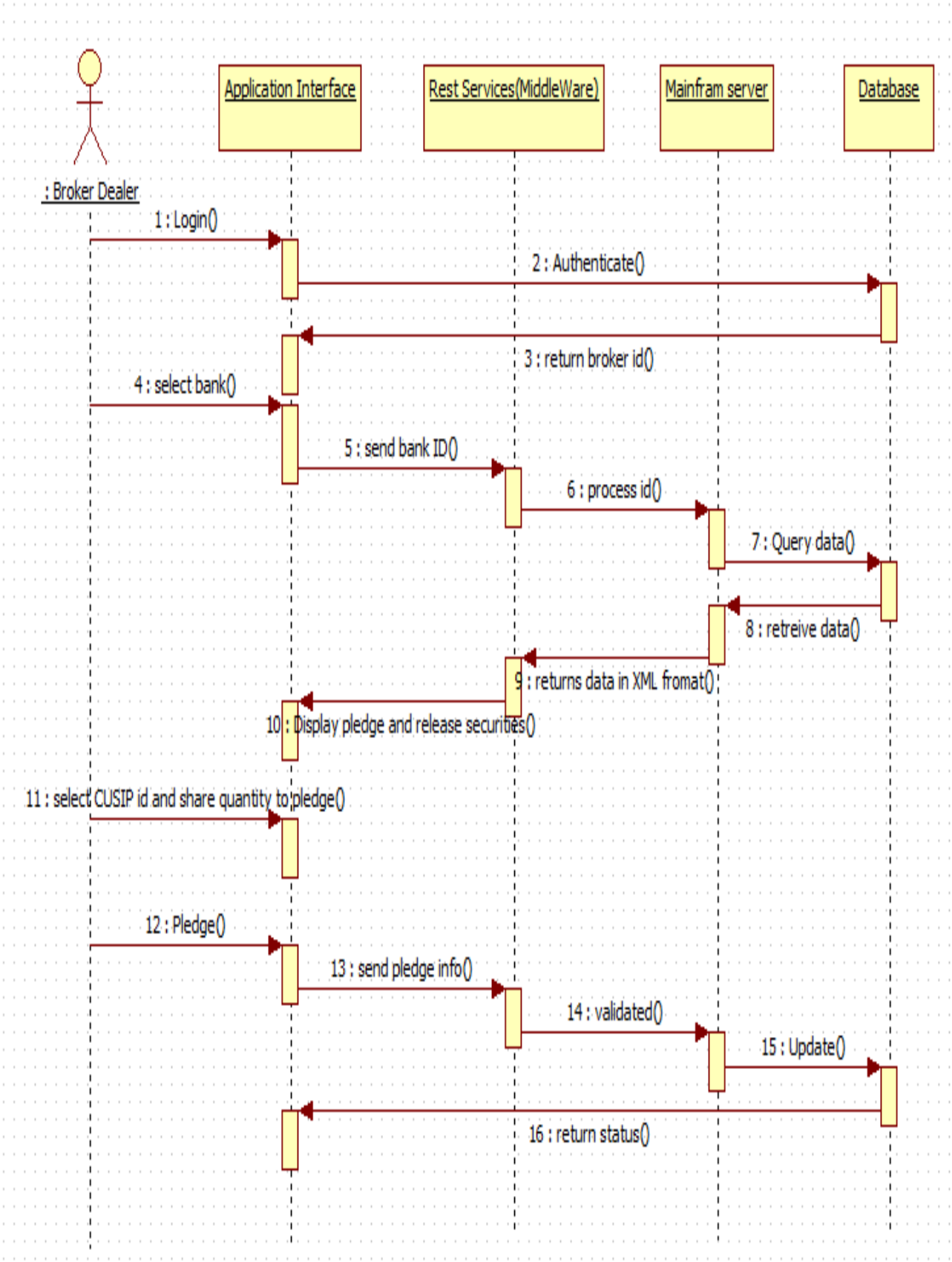


Fig 6.3 Sequence Diagram for Pledging and Release Process

This sequence diagram explains process flow and interaction of processes in a sequence. It explains how input is taken at each stage and processed to next stage.

It explains sequence of events through which entire process flow occurs. In the given diagram, process flow starts with broker dealer interacting with API and feeding input and this input is authenticated from database which returns the broker id. After the broker id is returned, the broker can select the bank out of all lists of banks available.

Based on the bank id and the process id queried to mainframe, the database returns the pledge and release share records under that bank. The data that is returned from Mainframe server to middleware is in XML format. The pledge and release security records under the given bank id is displayed in the application interface.

For the pledge process, the CUSIP ID and quantity of shares to be pledged is given as input in UI. That info is sent to rest service and processed and updated in database using Mainframe Server. The status of the current transaction is returned back to the application interface and displayed. The same process is carried out for release process also except it deals with releasing pledged shares.

Sequence Diagram process flow:

1. End user(broker dealer) to Application Interface.

2. End user provides inputs.

3. The inputs are sent to REST services in JSON format.

4. Mainframe server processes this inputs from REST Services.

5. Database gets updated based on the inputs provided.

6. Finally, the status of the transaction is returned back to the end user.

CLASS DIAGRAM:

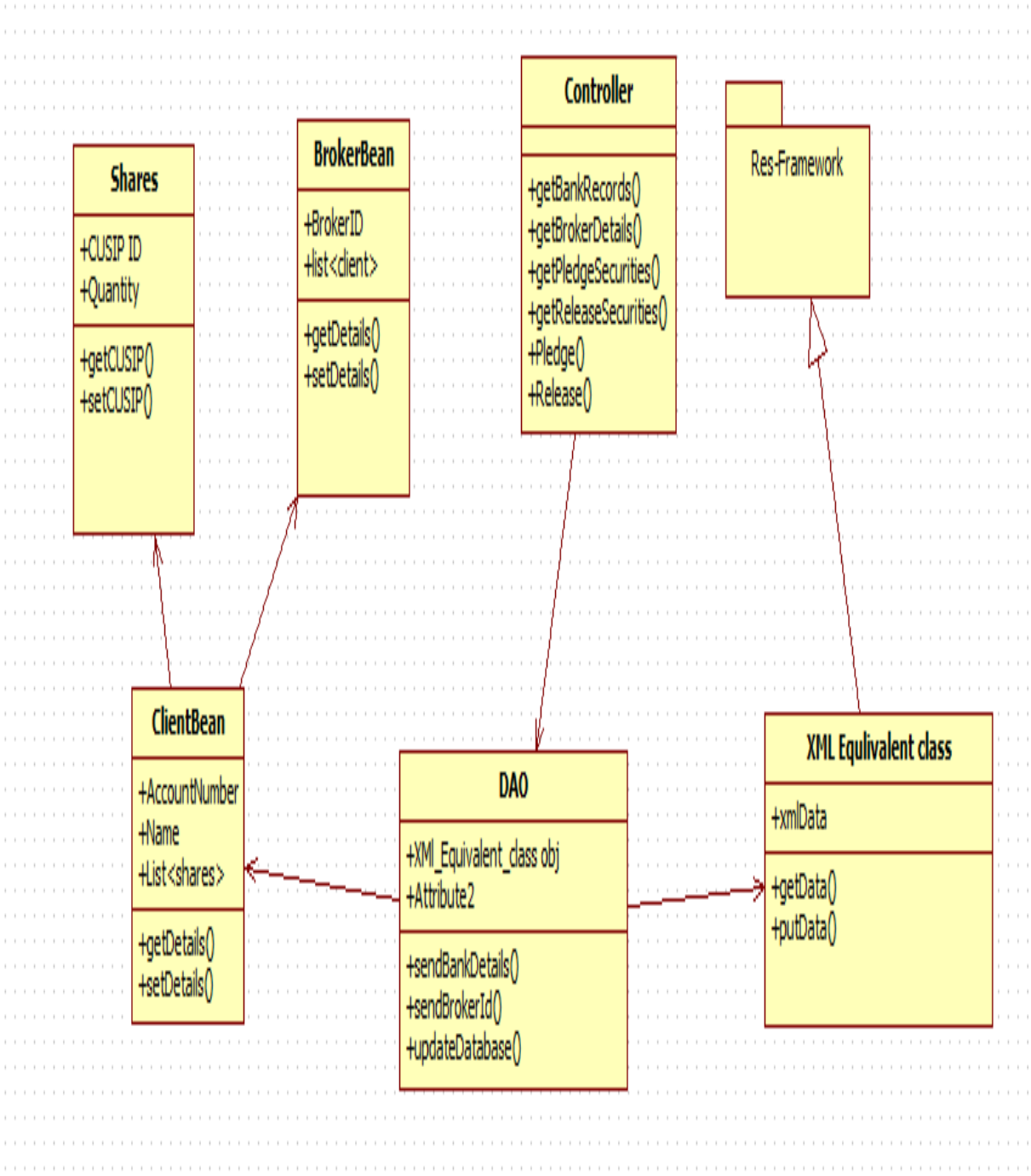


Fig 6.4: Class Diagram for Pledging and Releasing Process

In the above class diagram, six classes are available namely, Shares, BrokerBean, Controller, ClientBean, DAO, XMLEquivalentClass. These six classes form main contents of project and help in achieving goals.

Each class has several attributes which determines properties of class respectively. These classes interact with each other for successful functioning of the application.

Class Diagram Process Flow:

1. Controller to DAO.

2. Class Diagrams contains Attributes and Functions.

EXAMPLE:

*DAO contains attributes and functions as follows:

*Attributes:

* XMl_Equivalent_class obj

*Functions:

* sendBankDetails()

* sendBrokerId()

* updateDatabase()

3. In this way all the class Diagrams are linked :

Controller to DAO.

DAO uses ClientBean, Shares , BrokerBean, XMLEquivalentClass.

ClientBean will have Shares and BrokerBean Object.

XMLEquivalentClass uses classes present is Res Framework package.

COLLABORATION DIAGRAM:

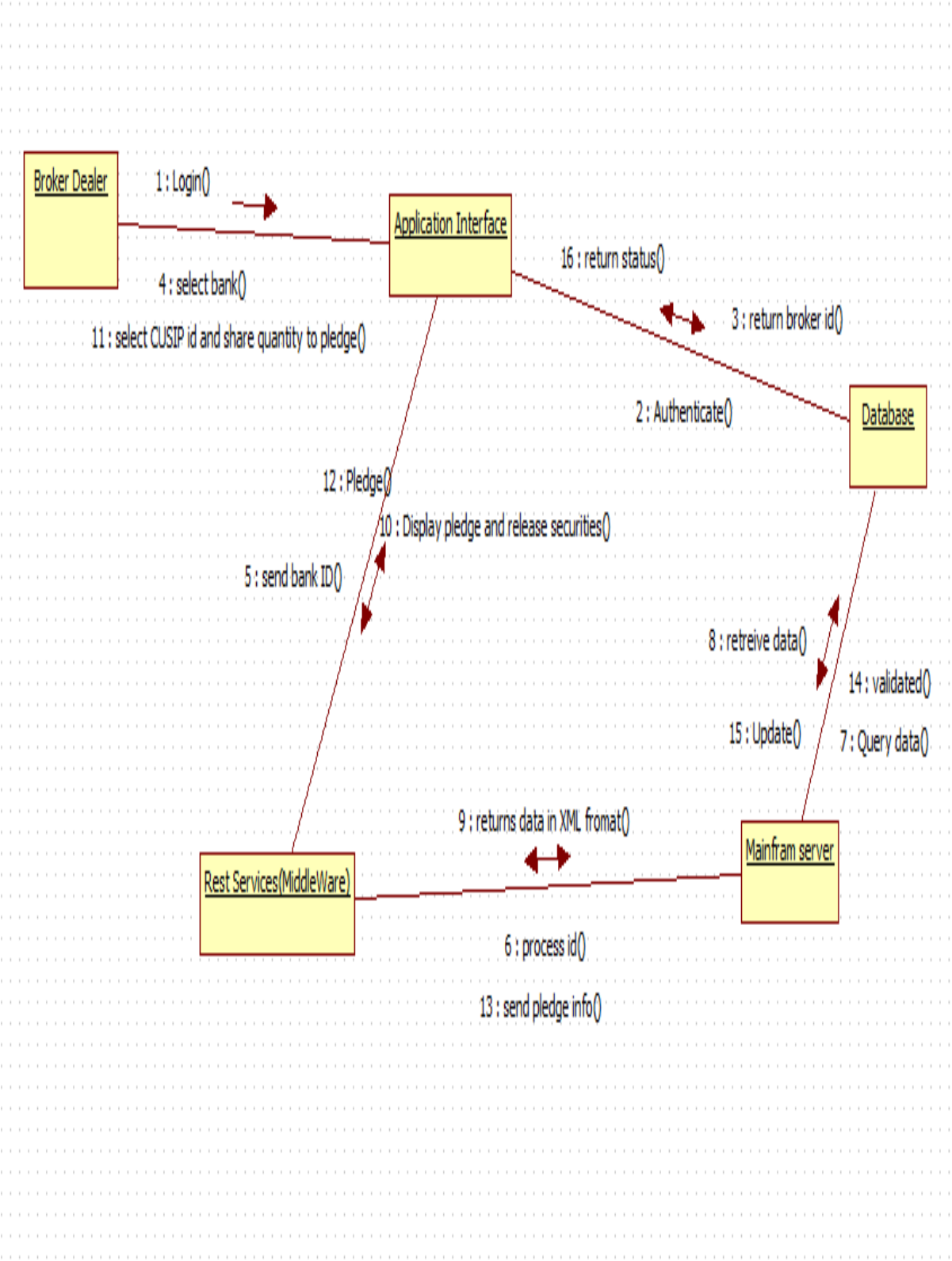


Fig 6.5: Collaboration Diagram for Pledging and Releasing Process

This collaboration diagram describes how various objects are organized in the project. It also depicts communication between objects which send and receive messages. It describes dynamic behaviour of the system.

In the above diagram, various objects like Broker Dealer, Application Interface , Database, Res Framework and Mainframe Server interact with each other transferring system messages which help in successful functioning of the system.

The collaboration Diagram of this project explains about the way of process flow with Arrow marks having inputs to next object.

Process Flow:

1. Broker Dealer logs on to application interface and authentication is done with the database.

2. A Broker id is returned to the application interface from the database.

3. Broker Dealer selects the bank and sends to middle ware which sends the process id to mainframe.

4. With the given requirements a query is made to the database which returns the data as output.

5. The data is returned from mainframe to middle ware in XML format

6. The Interface displays the pledge and release securities.

7. Now the broker dealer selects the cusip id and the quantity of shares to perform pledge.

8. The pledge request is sent to REST Service.

9. The information of the pledge is sent to mainframe.

10. The requested details that are sent to the database are validated.

11. Finally if the validation is true then the database gets updated and the status report is sent back to the application interface.

7) Methodology and Approach:

Module No. 1: Setting up the Initial Software and configuring all the configuration files

In the Eclipse IDE, all the API files like servlet.api are added into the build path. The configuration files like pom.xml, spring-servlet.xml, and web.xml are edited. All the required files are included. The created project is converted into Maven which helps in downloading the required dependencies from the internet. The versions of the required dependency are mentioned in pom.xml file.

Module No. 2: Performing the Pledging Process

Initially the broker selects the bank to perform the pledging process and the user interface displays the pledge and release security policies.

The Broker then selects the CUSIP id of the client shares and enters the quantity of each share to pledge. If the quantity of the shares to be pledged under the selected bank is less than or equal to the available quantity, then the stocks are successfully pledged and given to the bank.

Here, bank becomes the owner of the stocks and client borrows money from the bank. If the quantity is not available, then the application prompts the popup message stating insufficient quantity and it is not accepted.

Module No. 3: Performing the Release Process

Initially the broker selects the bank to perform the release process and the user interface displays the pledge and release security policies.

The Broker then selects the CUSIP id of the client shares and enters the quantity of each share to release. If the quantity of the shares to be released under the selected bank is less than or equal to the pledged quantity, then the stocks are successfully released and given back to the client.

Here, bank returns the stocks back to the owner upon receiving the money. Once the release process is done, the amount is deducted from the client's account depends on the number of share they released.

Since, same transaction can be performed concurrently in two or more systems, some transaction may fail. For every unsuccessful process, an error popup message is thrown with corresponding error statements. The log of every unsuccessful transactions are saved and can be viewed later.

Mainframe Server:

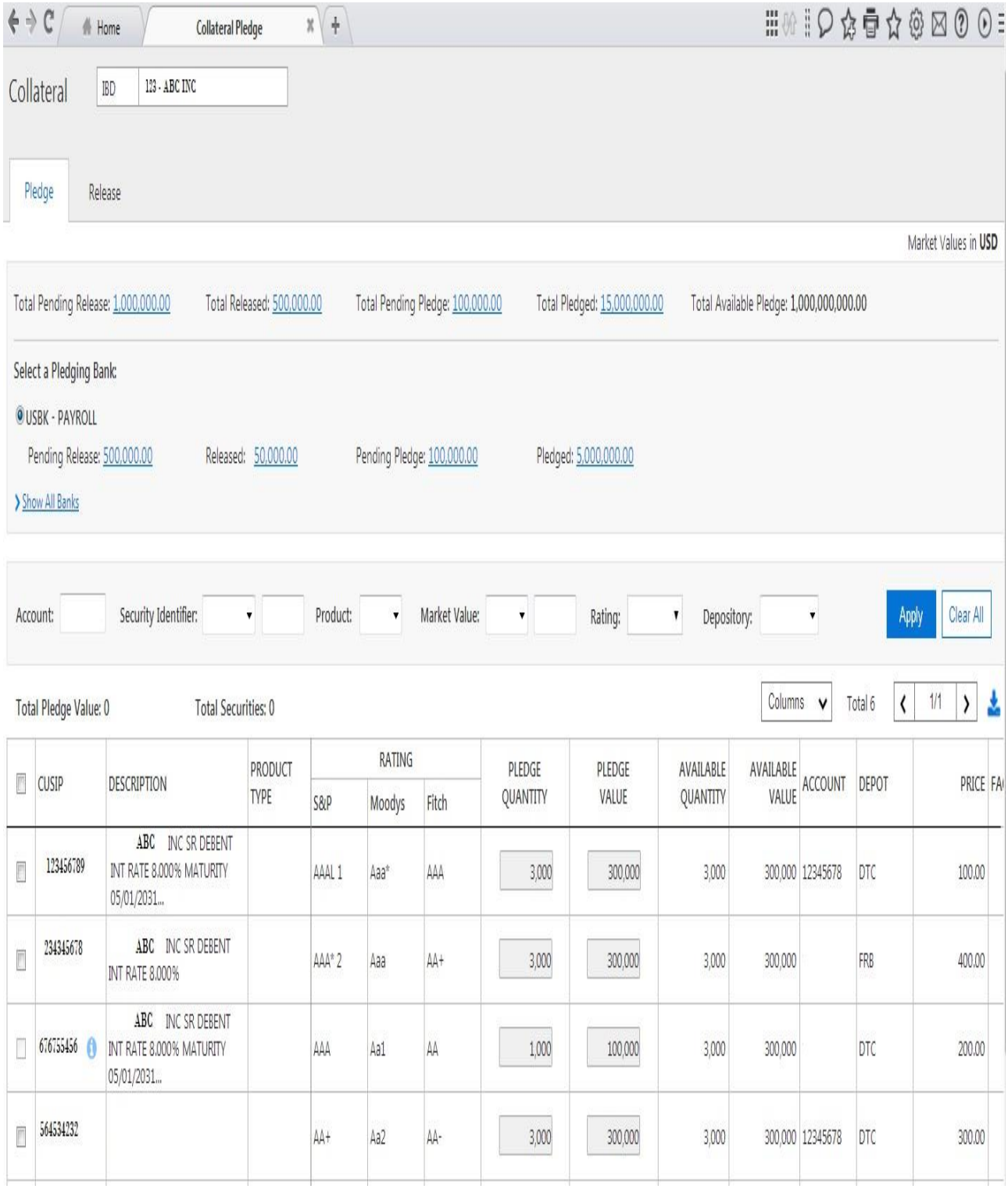
A Mainframe Computer is a high performance Multi User computer system which is the most scalable, available, reliable and secured machine in the world capable of performing some Million Instructions per second (up to 569,632 MIPS).

Eclipse IDE:

Eclipse is an IDE (integrated Development Environment) used in computer programming, and most widely used in Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and the primary use is for developing Java applications, but this may also be used to develop applications in other programming languages. It can also be used to develop documents with LaTeX (via a TeXlipse plug-in). Development environments include the Eclipse JDT (Java development tools) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

8) Output/Results:

Pledge Screen:

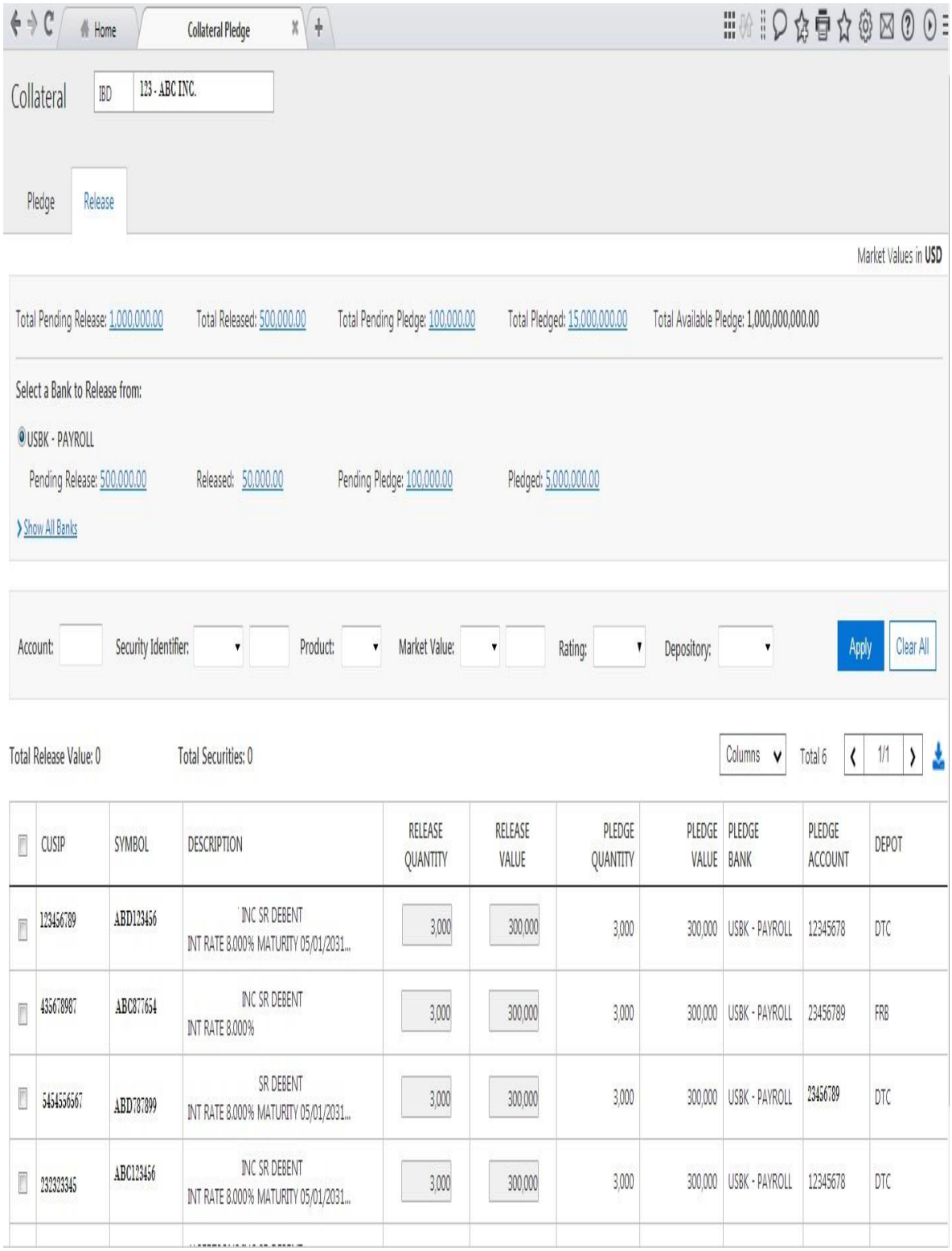


CUSIP	DESCRIPTION	PRODUCT TYPE	RATING			PLEDGE QUANTITY	PLEDGE VALUE	AVAILABLE QUANTITY	AVAILABLE VALUE	ACCOUNT	DEPOT	PRICE	FA
			S&P	Moodys	Fitch								
123456789	ABC INC SR DEBENT INT RATE 8.000% MATURITY 05/01/2031...		AAAL 1	Aaa*	AAA	3,000	300,000	3,000	300,000	12345678	DTC	100.00	
234345678	ABC INC SR DEBENT INT RATE 8.000%		AAA* 2	Aaa	AA+	3,000	300,000	3,000	300,000		FRB	400.00	
676755456	ABC INC SR DEBENT INT RATE 8.000% MATURITY 05/01/2031...		AAA	Aa1	AA	1,000	100,000	3,000	300,000		DTC	200.00	
564534232			AA+	Aa2	AA-	3,000	300,000	3,000	300,000	12345678	DTC	300.00	

Fig 8.1: Output Screen of Pledging Process

In the Fig 9.1: output screen of Pledge Process is shown. The broker id is displayed on the top. The details like total pending release, total release and total pending pledge and total available pledge are available. Filtering facilities are provided and the table is shown.

Release Screen:



CUSIP	SYMBOL	DESCRIPTION	RELEASE QUANTITY	RELEASE VALUE	PLEDGE QUANTITY	PLEDGE VALUE	PLEDGE BANK	PLEDGE ACCOUNT	DEPOT
123456789	ABD123456	INC SR DEBENT INT RATE 8.000% MATURITY 05/01/2031...	3,000	300,000	3,000	300,000	USBK - PAYROLL	12345678	DTC
43667898?	ABC877654	INC SR DEBENT INT RATE 8.000%	3,000	300,000	3,000	300,000	USBK - PAYROLL	23456789	FRB
3454556567	ABD787899	SR DEBENT INT RATE 8.000% MATURITY 05/01/2031...	3,000	300,000	3,000	300,000	USBK - PAYROLL	23456789	DTC
23233345	ABC123456	INC SR DEBENT INT RATE 8.000% MATURITY 05/01/2031...	3,000	300,000	3,000	300,000	USBK - PAYROLL	12345678	DTC

Fig 8.2: Output Screen of Releasing Process

In the Fig 9.1: output screen of Release Process is shown. It is similar to the pledge screen except that few columns in the table differ. Here, the pledged quantity and release quantity is displayed.

9) Conclusion:

Thus, the main aim of designing this application software is to ease out the user experience in dealing with the pledging and release process. This application also displays all the security policies of the software and makes the user interface self-explanatory, thereby increasing the comfort level of the user. The application is very secure since end to end encryption and decryption algorithms are used. This developed module responds to user queries in a fraction of time because of highly sophisticated technologies like Angular JS 2, Node JS . The best part of the application software is that there is no external agent is required to perform clearing process, instead a single module will take care of all the settlement process completely. The development and maintenance job of the application is also very easy. This application also acts as an intermediary between buyers and sellers of financial instruments. Such systems are called as Clearing House System.

10) References:

REFERENCE:

https://inautixonline.inautix.com/ – iNautix Company.