PROJECT REPORT ON

A RESTFUL DYNAMIC WEB MODULE TO MANAGE AND MAINTAIN THE CHIT FUND SYSTEM

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

GOMATHI.M - 117003062

Under the Guidance of

ANUJA RANI
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 **GOMATHI.M** with Register no.**117003062** 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**
ANUJA RANI	**Dr. A. UMAMAKESWARI**
INAUTIX TECHNOLOGIES Pvt Ltd, CHENNAI	**SCHOOL OF COMPUTING**

Submitted for University Examination held on_____

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 RESTFUL DYNAMIC WEB MODULE TO MANAGE AND MAINTAIN THE CHIT FUND SYSTEM**" 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 **ANUJA RANI**, iNautix Technologies, Chennai.

Date : **Name :** GOMATHI.M **Signature:**

Place : **Reg. No:** 117003062

ACKNOWLEDGEMENT

First and Foremost, we take pride in thanking the almighty who 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. 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

LIST OF FIGURES USED IN THIS PROJECT:

TABLE OF CONTENTS

1)ABSTRACT

In this era, money exchange has become an integral part of our daily lives. The Internet, the largest network of computer networks, is the medium usually favoured for electronic commerce because it allows an organization to cut service costs while increasing the speed of service delivery. The rapid adoption of digital payment systems and it's increasing usage needs lakhs of transactions to be made in one second.

There are MNC's that adapt high speed searching and caching algorithms for their clients. But small organizations that do financial transactions like chit fund transactions don't get to have those privileges. So, all those features are incorporated in an API and made available for the clients. There are mobile wallets for every client subscribed. You can transfer the money into these wallets online using credit/debit card or Net banking.This means that every time you pay a premium or buy stocks online via the wallet, you don't have to furnish the card details. You can use these wallets to pay premium, stocks and make online purchases.

The main aim of this project is to develop a RESTful API for a dynamic Maven web project, a Chit Fund System, using Spring Framework. Angular JS is used to make it highly responsive, dynamic and powerful. Data security is established using internal security systems. All data will be stored in a warehouse. Classifiers, predictive models are built and adaptive neuro-fuzzy inference system is used to predict the rates and accurate stocks for each clients. Bootstrap is used to provide user with a rich and user friendly experience.

2) INTRODUCTION

EXISTING SYSTEM:

There are many chit fund systems that exist. They provide an interface to transfer money. But the stock rates and accurate predictions are not available. The system is globally available but they do not meet the requirements of our clients.

OBJECTIVES

To develop a chit fund system which provides:

- Integrating with other platforms
- Organisation based customization.
- Low cost.
- Upgradability with upcoming database technologies
- Lightweight processing
- Concurrency
- High availability or disaster recovery
- Storage and compression
- Ease of integration with external libraries
- Full SQL support vs partial support
- Backup option
- Integration with reporting and archive tools
- Integration with other API’s

3) PROBLEM STATEMENT

A Chit Fund System can be considered as transfer of money or stocks with some predefined rules that dynamically changes. It is an agreement between the client and the host. 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.

Earlier, it was very difficult to manage this chit process for all the individual clients. The clients had to manually do all the calculations and had to make a call to notify them for receiving their premium. All the data are available only with the group holder. The user has to rely on that particular user to get the information. When the user is in need of information, they have to wait. This situation is negligible if the scope of the system is small. But it leads to a disastrous situation when that's not the case.

In order to overcome the burden of the user, a single page is developed to fetch all the details from various components and display in the required format. All the details of the clients and the users are made available in this system and it is easy to view them. 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 user id can be viewed with great ease. The chit fund system is highly responsible and user friendly for all clients.

4) REQUIREMENT SPECIFICATION

4.1. Functional Requirement Specification:

Functional requirement as the product capabilities are things that a product must do for its user. Functional requirements define how software behaves to meet user needs. A functional requirement is a requirement that, when satisfied, will allow the user to perform some kind of function. In Software engineering and systems engineering, a functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behaviour, and outputs.

Some of the functional requirements used in this system are:

- Data must be entered before a request can be approved.
- Clicking the Approve button moves the request to the Approval Workflow.

4.2. Non Functional Requirement Specification:

Non-functional requirements as the quality attributes, design and implementation constraints, and external interfaces which a product must have.Non-functional requirements may describe aspects of the system that don't relate to the execution, but rather to the evolution over time. Security, usability, testability, extensibility, reliability, scalability, portability and safety are the non-functional requirements that are met in this project.

4.3. HARDWARE REQUIREMENT SPECIFICATION:

4.3.1. 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 4.3.1.1

4.3.2. Server Side:

An n-tier web based database architecture model is used. Virtual dedicated hosting is followed for server side hosting. The server is up for 24/7. If the server is down, a backup alternate server is used seamlessly.

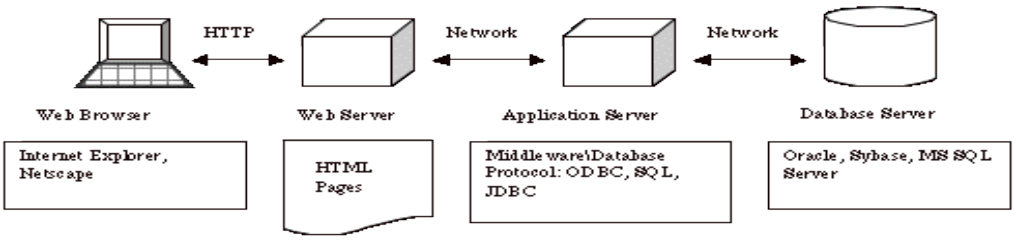

Figure 4.3.2.1

4.4. SOFTWARE REQUIREMENTS SPECIFICATION:

4.4.1) Apache Tomcat Installation

Apache Tomcat is an open-source Web server and servlet container.It requires a Java Standard Edition Runtime Environment (JRE) version 6 or later.

STEPS TO INSTALL:

1. Download and install JRE from http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Download and install Apache Tomcat, a binary distribution of tomcat from http://tomcat.apache.org/
3. Unpack the binary distribution so that it resides in its own directory (conventionally named "apache-tomcat-[version]").
4. Configure Environment Variables
   - **4.1.1** Set CATALINA_HOME and CATALINA_BASE(optional).
   - **4.1.2** The CATALINA_HOME environment variable should be set to the location of the root directory of the "binary" distribution of Tomcat.
   - **4.1.3** The CATALINA_BASE environment variable specifies location of the root directory of the "active configuration" of Tomcat. It is optional. It defaults to be equal to CATALINA_HOME.
5. Set JRE_HOME or JAVA_HOME.

   5.1.1 The JRE_HOME variable is used to specify location of a JRE. The JAVA_HOME variable is used to specify location of a JDK.

   5.1.2 Using JAVA_HOME provides access to certain additional start-up options that are not allowed when JRE_HOME is used. If both JRE_HOME and JAVA_HOME are specified, JRE_HOME is used. The best place to include these variables is a "setenv" script.
6. Other Variables like CATALINA_OPTS are optional to set with. It allows specification of additional options for the java command to start tomcat.
7. Start Tomcat

7.1.1 On Windows

%CATALINA_HOME%\bin\startup.bat

Or

%CATALINA_HOME%\bin\catalina.bat start

Or

$CATALINA_HOME/bin/catalina.sh start

4.4.2) DERBY Installation

cdC:\BXP Dev Env\eclipse-kepler-win32-1.0\eclipse-kepler-win32\jdk7-u51\db\bin

set DERBY_HOME=C:\Data\jdk7-u51\db;

set JAVA_HOME=C:\Data\jdk7-u51\bin;

set PATH=C:\Data\jdk7-u51\db\bin;C:\Data\jdk7-u51\bin;%PATH%;

set DERBY_HOME=C:\Data\jdk7-u51\db

C:\BXP Dev Env\eclipse-kepler-win32-1.0\eclipse-kepler-win32\jdk7-u51\db\bin>startNetworkServer -h 172.00.00.00

Bringing up your Client :

cdC:\BXP Dev Env\eclipse-kepler-win32-1.0\eclipse-kepler-win32\jdk7-u51\db\bin

set DERBY_HOME=C:\Data\jdk7-u51\db;

set JAVA_HOME=C:\Data\jdk7-u51\bin;

set PATH=C:\Data\jdk7-u51\db\bin;C:\Data\jdk7-u51\bin;%PATH%;

```
set DERBY_HOME=C:\Data\jdk7-u51\db
ij;
```


4.4.3) AngularJS

Angular is a client side JavaScript framework for dynamically adding interactivity for HTML. Data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes.It saves developer's productivity by reducing considerable amount of code for manipulating, traversing and listening to DOM.AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

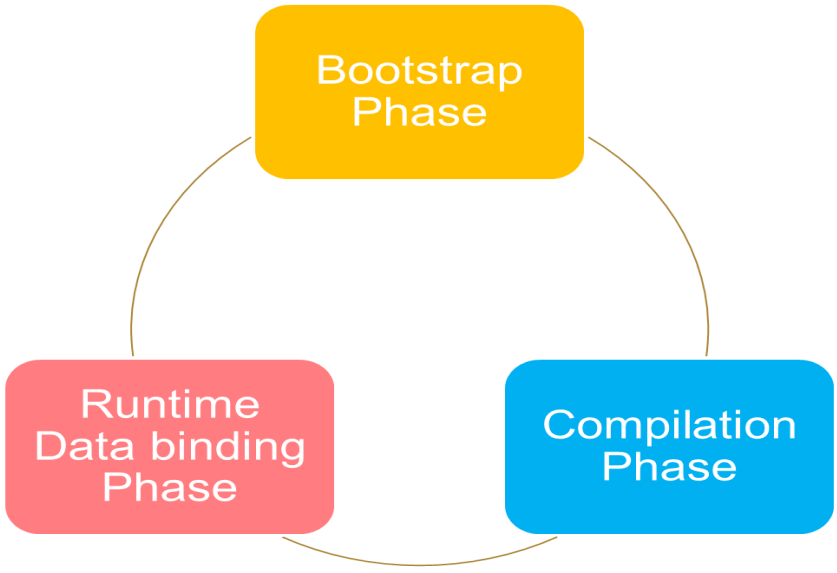

Figure 4.4.3.1

To use Angular JS include the following script in the head tag.

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

Using Angular JS with RESTful web services a responsive table like the below can be generated.

Figure 4.4.3.2

Sample Code for ng-table

```
<divclass="row">
      <divclass="col-lg-3"></div>
            <divid="searchPP"class="col-lg-6">
                  <divclass="form-group has-warning has-feedback">
                  <inputtype="text"class="form-control"id="srPP"ng-
model="searchMsg">
                  <spanclass="glyphiconglyphicon-search form-control-
feedback"></span>
                  </div>
      </div>
<divclass="col-lg-3"></div>
</div>
            <br>
```

```
<tableclass="table table-bordered"ng-controller="MessageController">
                        <thead>
                              <tr>
                                    <th>Message ID</th>
                                    <th>From</th>
                                    <th>Message</th>
                                    <th>Sent Date</th>
                                    <th>Reply or Ignore</th>

                              </tr>
                        </thead>
                        <tbodyng-repeat="data in datas | filter:searchMsg">
                        <tr>
                              <td> {{ data.mid}}</td>
                                          <td> {{ data.pid}}</td>
                                          <td> {{ data.message}}</td>
                                          <td> {{ data.sentDate}}</td>
                                          <td> {{ data.replied==0 ? 'Replied' :
'Not Replied'}}</td>
                        </tr>
                        </tbody>
                  <tfoot>
                        <tr>
                              <tdcolspan="5"align="center">
                                                      <!--<label
class="radio-inline">
                                                <input type="radio"
name="yes" value="yes" ng-model="myOpt"">Reply
                                                </label> -->
                                                <labelclass="checkbox-
inline">
                                                <inputtype="checkbox"ng-
model="myOpt">Reply
                                                </label>

                              </td>
                        </tr>
                        <tr>
                              <tdcolspan="5"ng-show="myOpt"align="center">
                                    <formmethod="Post"action="UReply">
                              <divclass="col-lg-4">
                              </div>
                        <divclass="col-lg-4">

                        <divclass="form-group">
                        <labelfor="per">User ID</label>
                        <divclass="input-group">
                        <inputtype="number"class="form-
control"id="per"name="per"placeholder="XX123"required>

                        </div>
                        </div>
                        <divclass="form-group">
                        <labelfor="rep">Reply Message</label>
                        <divclass="input-group">
```

```
                        <textareaclass="form-control"id="rep"name="rep"requiredng-
model="sub"></textarea>
                        </div>
                        </div>

      <inputtype="submit"name="repmsg"id="repmsg"value="Reply"class="btnbtn-default
float-xs-right"ng-show="sub">
                        </div>
                        <divclass="col-lg-4">
                              </div>
                        </form>
                                    </td>
                              </tr>
                        </tfoot>
                  </table>

                  <divclass="col-lg-8"ng-show="myOpt"></div>
```


A controller is created for the above table. It helps to get the data from the https RESTful web services using GET.

```
app.controller("MessageController",function($scope,$http){
      $http.get("message/getMsg").then(function(response){
            $scope.datas=response.data;
      });
});
```

4.4.4) Spring MVC

The Spring Web MVC framework provides Model-View-Controller (MVC) 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.

- The **Model** encapsulates the application data and in general they will consist of POJO.
- The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

The following Configurations should be made,


```
	<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
<display-name>OCF</display-name>
	<servlet>
<servlet-name>spring</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet
</servlet-class>
	<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
	<servlet-name>spring</servlet-name>
	<url-pattern>/</url-pattern>
	</servlet-mapping>
</web-app>
```

A Controller is defined in the following way,

```
	@RestController
@RequestMapping(value="/message")
publicclassMessageController {
	@RequestMapping(value = "/getMsg" , method = RequestMethod.GET )
	publicArrayList<MessageBean>getAllMessages() {
		DBMSDaodd=newDBMSDao();
		ArrayList<MessageBean>mAl=dd.getMessages();
		returnmAl;
  }
}
```

It returns a JSON object.Since the JSON format is text only, it can easily be sent to and from a server. It is used for data transfer through all the pages.

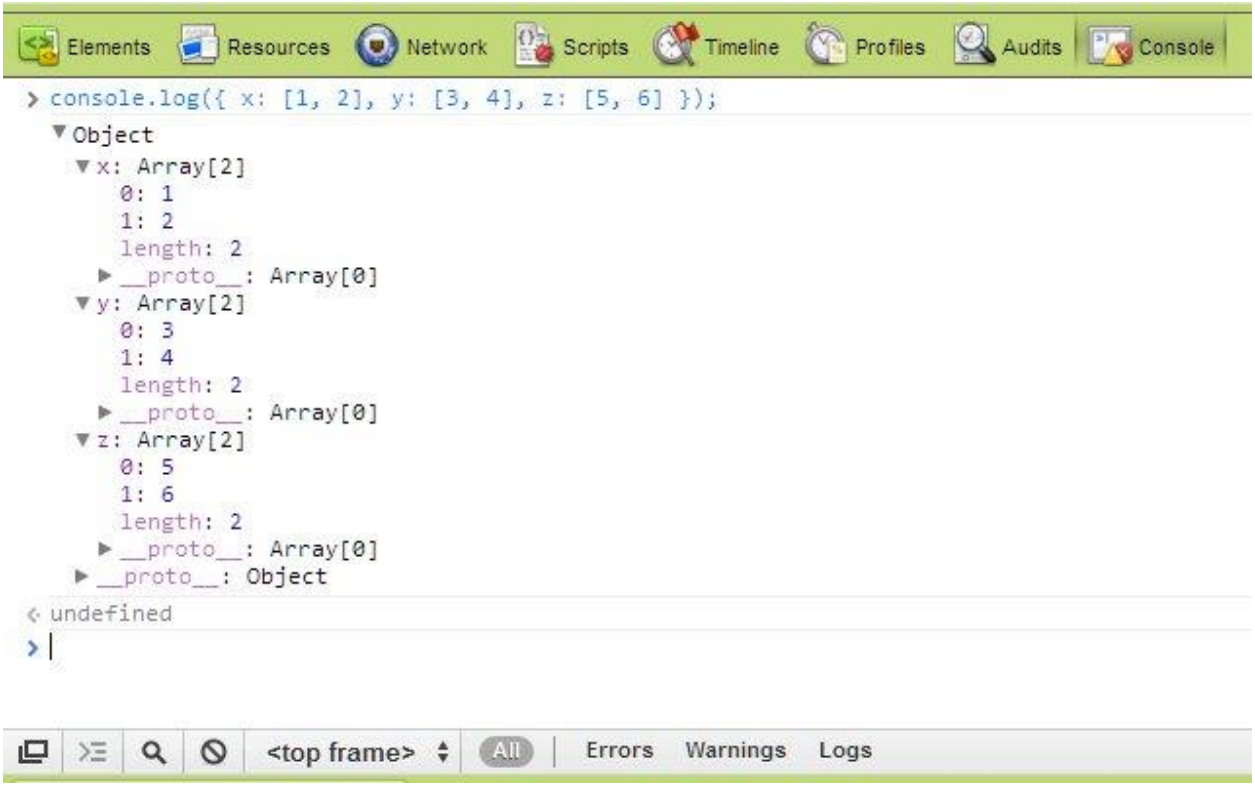

Figure 4.4.4.1

Spring framework provides a simplified approach in handling database access with the Spring JDBC Template. It allows cleaning up the resources automatically, e.g. releasing the database connections.

```
public interface IDao {
voidsetDataSource(DataSource ds);
void create(String firstName, String lastName);
    List<Person>select(String firstname, String lastname);
    List<Person>selectAll();
voiddeleteAll();
void delete(String firstName, String lastName);
}
public class PersonResultSetExtractor implements ResultSetExtractor {
    @Override
public Object extractData(ResultSetrs) throws SQLException {
        Person person = new Person();
person.setFirstName(rs.getString(1));
person.setLastName(rs.getString(2));
return person;
```

```
    }
}

public class PersonRowMapper implements RowMapper {
    @Override
public Object mapRow(ResultSetrs, int line) throws SQLException {
PersonResultSetExtractor extractor = new PersonResultSetExtractor();
returnextractor.extractData(rs);
    }
}
public class DerbyDao implements IDao {
privateDataSourcedataSource;
public void setDataSource(DataSource ds) {
dataSource = ds;
    }
public void create(String firstName, String lastName) {
JdbcTemplate insert = new JdbcTemplate(dataSource);
insert.update("INSERT INTO PERSON (FIRSTNAME, LASTNAME) VALUES(?,?)",
new Object[] { firstName, lastName });
    }
public List<Person> select(String firstname, String lastname) {
JdbcTemplate select = new JdbcTemplate(dataSource);
return select
                .query(
                        "select  FIRSTNAME, LASTNAME from PERSON where
FIRSTNAME = ? AND LASTNAME= ?",
new Object[] { firstname, lastname },
newPersonRowMapper());
    }
public List<Person>selectAll() {
JdbcTemplate select = new JdbcTemplate(dataSource);
```

```
returnselect.query("select FIRSTNAME, LASTNAME from PERSON",
newPersonRowMapper());
    }
public void deleteAll() {
JdbcTemplate delete = new JdbcTemplate(dataSource);
delete.update("DELETE from PERSON");
    }

public void delete(String firstName, String lastName) {
JdbcTemplate delete = new JdbcTemplate(dataSource);
delete.update("DELETE from PERSON where FIRSTNAME= ? AND LASTNAME = ?",
new Object[] { firstName, lastName });
    }
}
```


4.4.5) Maven

Maven is a comprehension and software project management tool that allows a developer to develop based on project object model(POM). Maven's key feature is dependency management. It helps to download the required JAR's for the project build.

```
<properties>
    <jdk.version>1.7</jdk.version>
    <spring.version>4.3.3.RELEASE</spring.version>
</properties>
<dependencies>
<dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.12</version>
```

```
                <scope>test</scope>
            </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>4.3.3.RELEASE</version>
        </dependency>
        <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.3.RELEASE</version>
        </dependency>
        <dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>4.3.3.RELEASE</version>
</dependency>
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-core</artifactId>
<version>2.5.0</version>
```

```
</dependency>
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>servlet-api</artifactId>
<version>2.5</version>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>javax.ws.rs</groupId>
<artifactId>javax.ws.rs-api</artifactId>
<version>2.0</version>
</dependency>
<dependency>
<groupId>org.webjars</groupId>
<artifactId>jquery</artifactId>
<version>1.12.4</version>
```

```
</dependency>

</dependencies>
```

Beans are initialized as follows so that Maven can use it to implement Spring injections like JdbcTemplate.

```
<?xml version="1.0" encoding="UTF-8"?>

<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">

    <context:annotation-config />

    <context:component-scan base-package="com.chitfund.Controller" />

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">

        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
```

<property name="url" value="jdbc:oracle:thin:@10[redacted]" />

<property name="username" value="[redacted]" />

<property name="password" value="[redacted]" />

</bean>

<mvc:default-servlet-handler/>

<mvc:annotation-driven />

</beans>

5) INTERACTION SCENARIO

5.1 ACTIVITY DIAGRAM

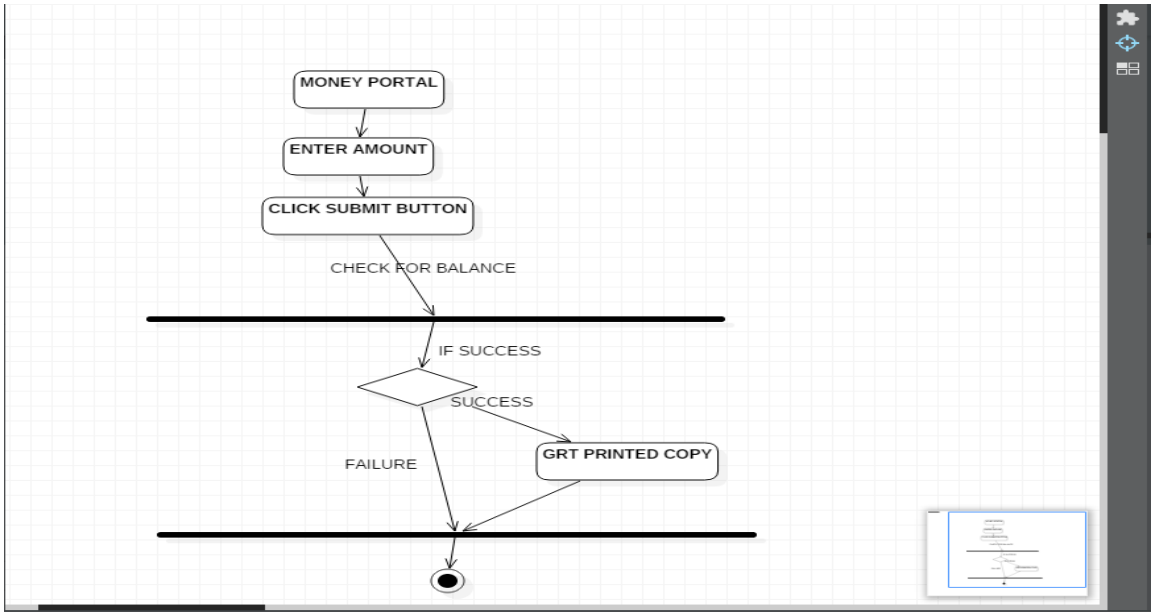

Figure 5.1.1

In this activity diagram, the generalised workflow of transferring money from a client to other host is depicted. A secured payment gateway acts as an interface for the transfer of money. If the transaction is success, the user is notified with push notification feature. It prompts to get a hard copy of the transaction. This data is then stored in a warehouse for further prediction purposes.

5.2. SEQUENCE DIAGRAM

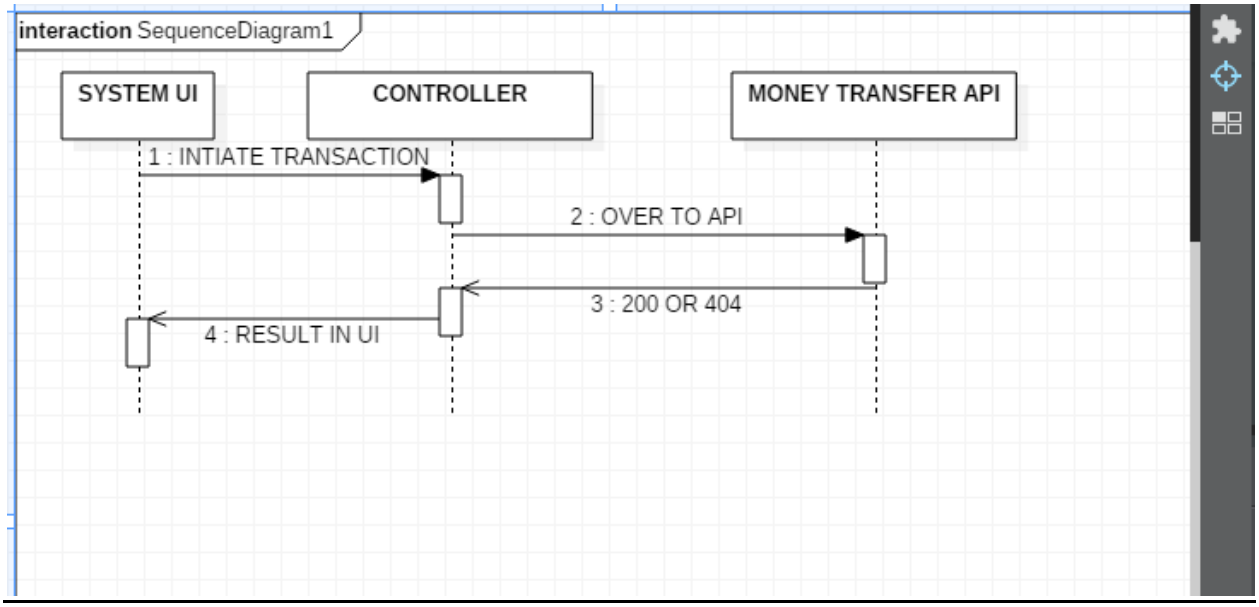

Figure 5.2.1

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 end user interacting API and feeding input and this input is taken to server using RESTful API http get, post, delete, put methods.

The client uses a secured post method to access the data or login to the service. The client can transfer money to the selected member of the particular chit for a particular month. Every client who subscribed to a particular plan has to pay through the gateway. Two way data binding is used for the client's seamless experience.

SEQUENCE PROCESS FLOW

1. Check in to the chit system through User Interface
2. The client checks for the particular month's premium and the selected subscriber.
3. The foreman auctions the chit and manages it.
4. Money transaction is done through a secured payment gateway.
5. The selected subscriber from the auction gets the premium.
6. An appropriate http status code is sent to the client side using RESTful API

5.3. CLASS DIAGRAM

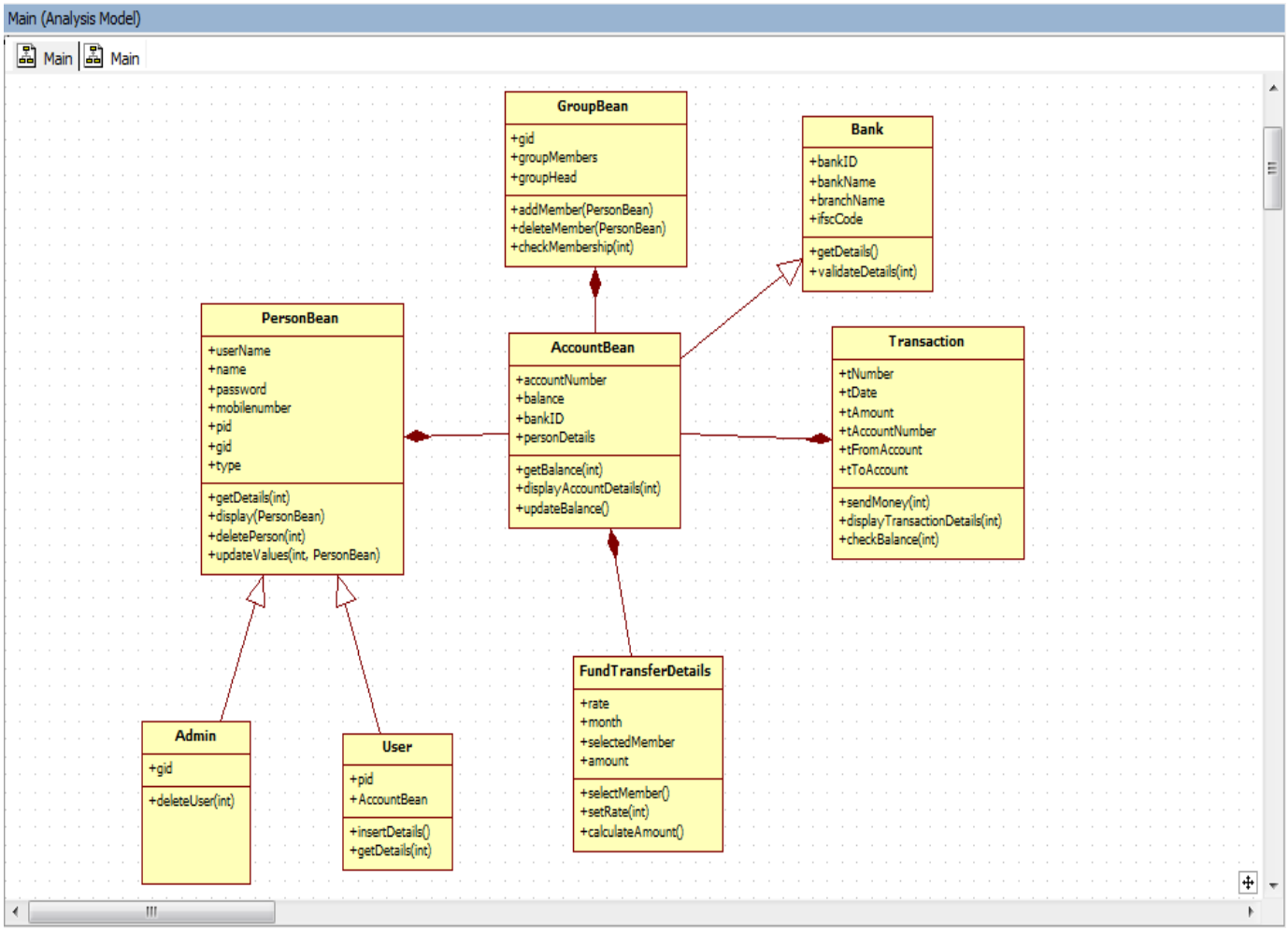

Figure 5.3.1

In the above class diagram the main actors involving in the system are mentioned. Every class are interdependent and follows facade design pattern for the relationships between the entities or the classes for a simplified interface of a complex subsystems like chit system.

Actors

- PersonBean – all functionalities of the system are accessed using this class
- AccountBean – It has a composition relationship with the PersonBean class.
- GroupBean – It has an aggregate relationship with the PersonBean and composition relationship with the AccountBean.
- TransactionBean – It has a composition relationship with the account bean. It implements a secured payment gateway.
- Admin, User – It has inherits PersonBean and has a generalisation relationship.

5.4. USE CASE DIAGRAM

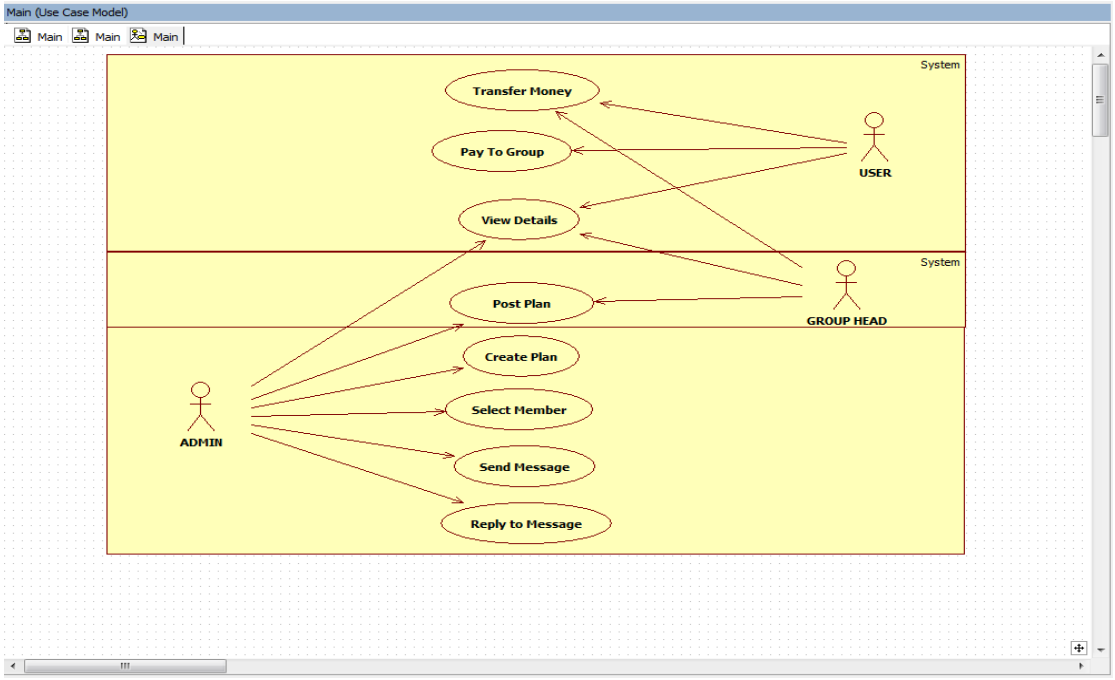

Figure 5.4.1

The use case diagram above gives the core functionalities of the chit system. There are three actors: Admin, User and Group Head. The admin can create the plan. The admin can post the chit plan to the group head every month. The group head collects the premium from the subscribed members in the group. The user and the group head can view the details of the plan. The user can transfer money to the selected member.

5.5. ER DIAGRAM

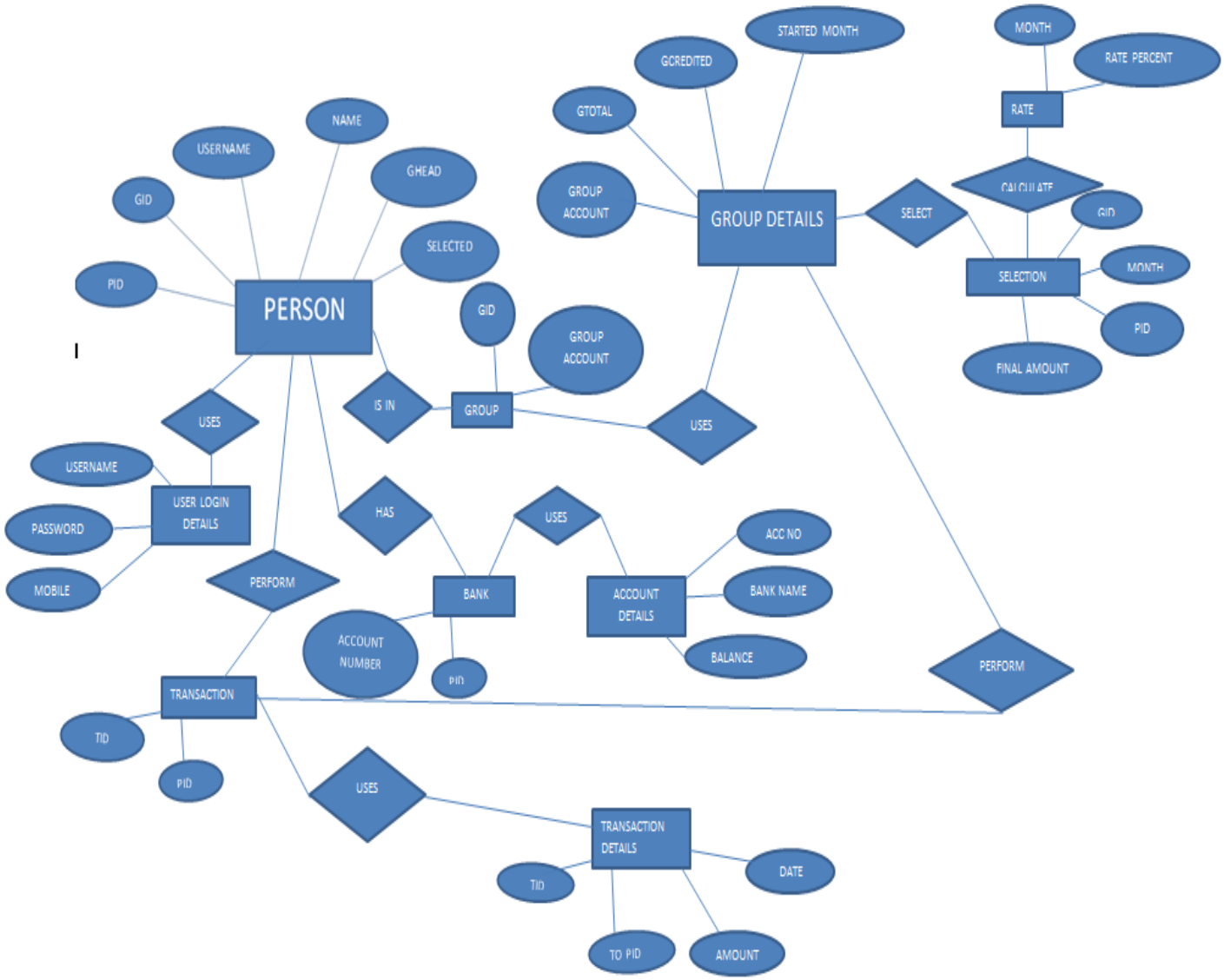

Figure 5.5.1

The above ER diagram gives the basic table structure used for the chit system. The core tables are

- Person Table
  - PID
  - GID
  - USERNAME
  - NAME
  - GHEAD
  - SELECTED
- Transaction Table
  - PID
  - TID
  - AMOUNT
  - DATE
- Group Details Table
  - GROUPACCOUNTNUMBER
  - GTOTAL
  - GCREDITED
  - STARTEDMONTH
  - DUE
  - GMTOTAL
  - GMCREDITED
- Selection Table
  - GID
  - PID
  - MONTH
  - AMOUNT
- Account Details
  - ACCOUNTNUMBER
  - BANKNAME

- BALANCE

6) Conceptual Model / Proposed Architecture

6.1.SPRING MVC ARCHITECTURE

6.1.1.BLOCK DIAGRAM

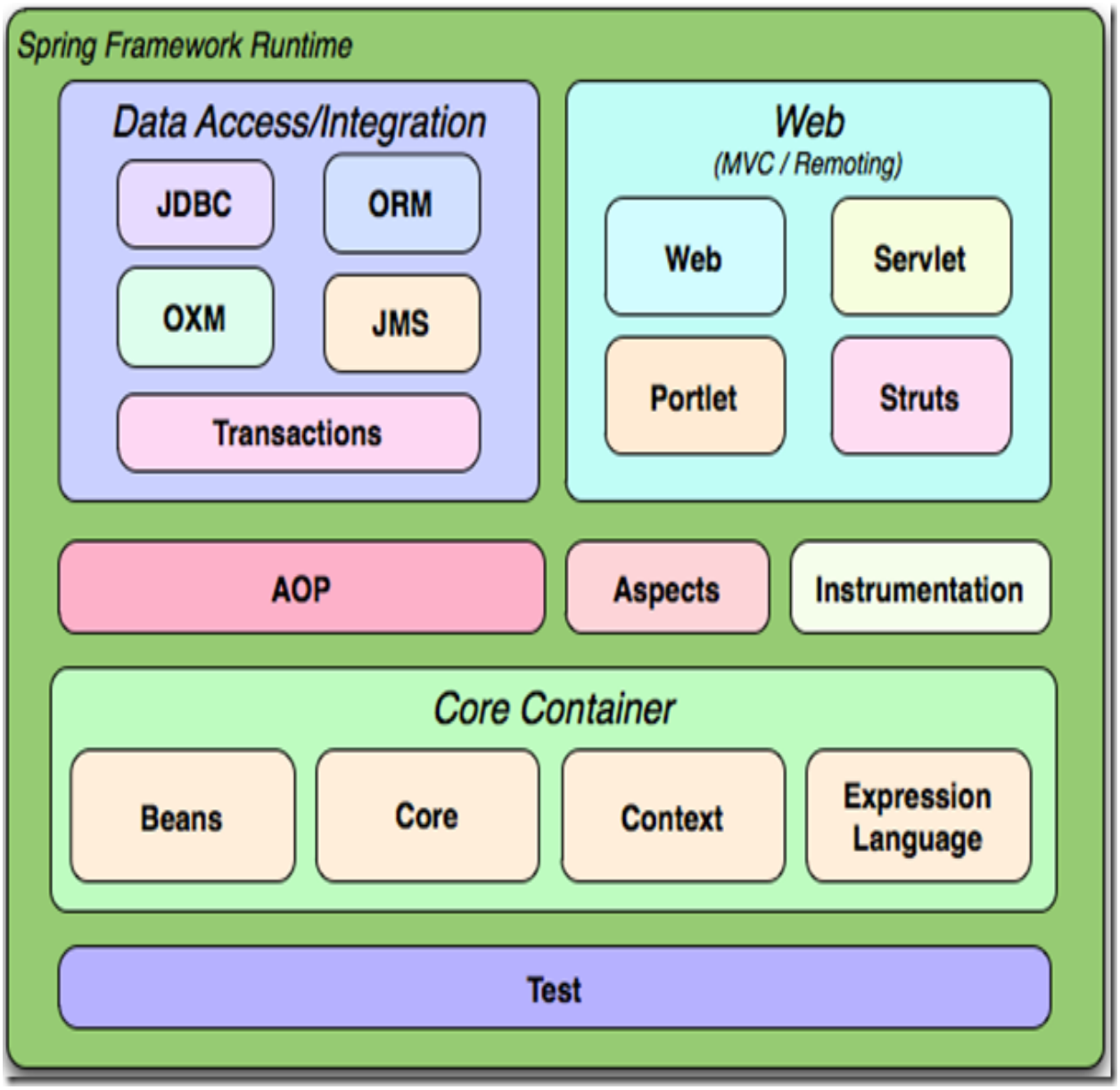

Figure 6.1.1.1

The spring web MVC framework provides model-view-controller architecture and in-built components that could be used to develop flexible and loosely coupled web applications. The MVC pattern helps in separating the different parts of the application (input logic, business logic, and User Interactive logic), by 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.

6.1.2. BENEFITS OF SPRING FRAMEWORK:

- Spring helps developers to develop enterprise-class applications using POJOs. The advantage of using only POJOs is that you need have an EJB container product such as an application server instead use only a robust servlet container such as Tomcat Server.
- Spring is organized in a modular structure.
- Spring truly makes use of some of the existing technologies like ORM frameworks, logging frameworks, JEE, Quartz and JDK timers.
- Spring applications are easy to test and simple because environment dependent code is moved into framework. In addition, by using Java Bean-style POJOs, it becomes easier to use dependency injection for injecting test data.
- Spring provides a suitable and compatible API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.
- Lightweight IoC containers are useful for developing and deploying applications on systems with limited memory and CPU resources.
- Spring provides consistent transaction management interface which scales down to a local transaction (using a single database, for example) and scales up to global transactions (using JTA, for example).

- Spring Session provides an API and implementations to manage user's session information. It also provides transparent integration with:
    1. Http Session -replaces the Http Session in an application container (i.e. Tomcat)
    2. Clustered Sessions - Spring Session support clustered sessions without being connected to application container specific solution.
    3. Multiple Browser Sessions - Spring Session supports managing multiple users' sessions in a single browser instance.
    4. RESTful APIs - Spring Session provides session ID in headers to work with RESTful APIs.
    5. Web Socket - keeps the Http Session alive when receiving Web Socket messages

6.1.3.SPRING IOC CONTAINERS:

The spring container is the core element of Spring Framework. The container creates objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction.

The spring container uses dependency injection (DI) to manage the components which constitute an application. These objects are called Spring Beans .The container gets information on what objects to create, configure, and assemble from configuration metadata provided which can be represented either by XML, Java annotations, or Java code. The following diagram is a high-level view of how spring works. The Spring IoC container makes use of POJO classes and configuration metadata to produce a completely configured and executable application.

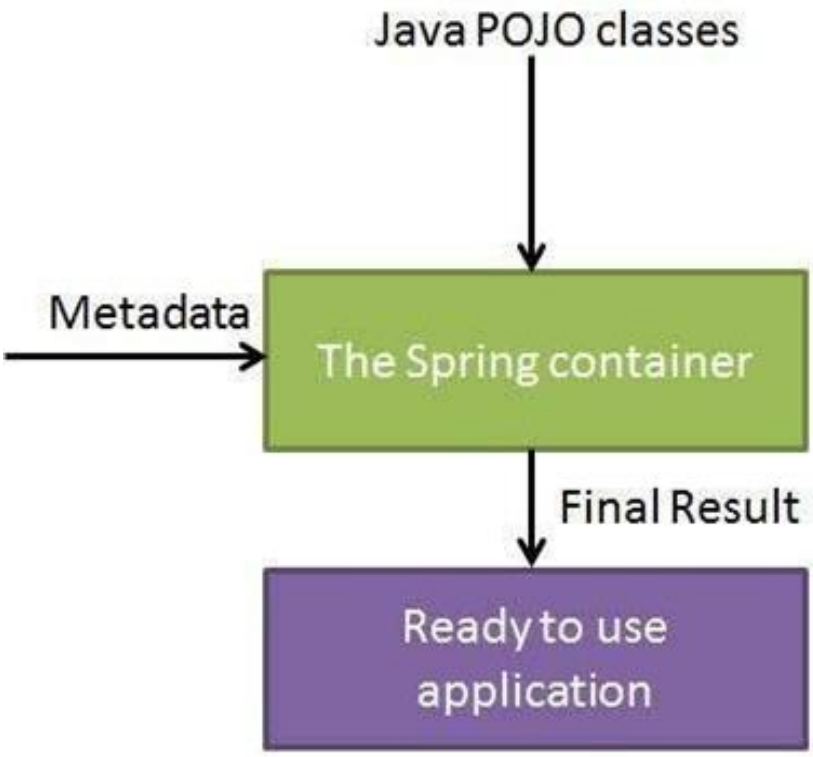

Figure 6.1.3.1

6.1.4.ASPECT -ORIENTED PROGRAMMING:

Aspect oriented programming (AOP) is one of the key components of spring framework. The functions which span multiple points of an application are called cross-cutting concerns and they are theoretically separate from the application's business logic. There are numerous examples of aspects including logging, declarative transactions, security, and caching.

The key aspect of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. The AOP module provides aspect-oriented programming implementation which enables you to define method-interceptors and point cuts to clearly decouple code which implements functionality that should be separated.

- Aspect: modularization of a concern that cuts across multiple classes. Eg :Transaction management
- Join point: a point which occurs during the execution of a program or the handling of an exception. In Spring AOP, it always represents a method execution.
- Advice : Action taken at a particular joinpoint occurred. Different types of advice are “after”, ”before” and “around”. Spring model an advice as an interceptor.

Pointcut: a predicate which is matched with join points. Advice is associated with a pointcut expression and runs at any join point matched by the point cut.

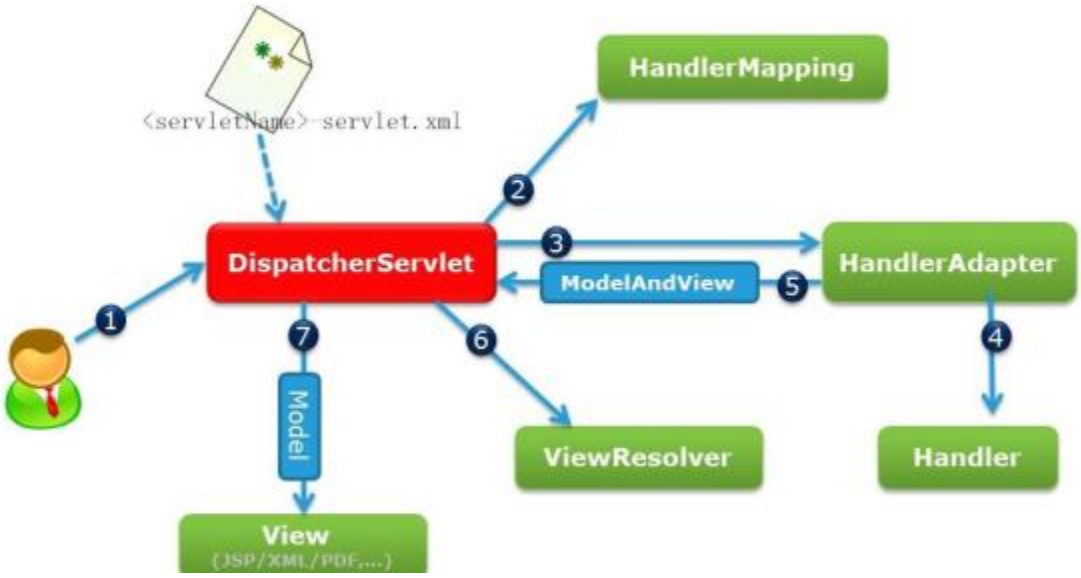

Figure 6.1.4.1

6.1.5.DEPENDENCY INJECTION(DI):

The spring framework is recognized and used widely for having Dependency Injection (DI) as flavour of Inversion of Control. Dependency Injection is a concrete example of Inversion of Control.

When we build complex Java application, DAO classes should be independent as possible of other Java classes to increase the reusability and to test independently Dependency Injection helps in connecting these classes together and same time keeping them independent.

Dependency is something which translates into an association between two classes. For example, class A is dependent on class B. Now, Injection is that class B will get injected into class A by the IoC. Dependency injection can be as of passing parameters to the constructor or by post-construction using setter methods.

6.1.6. DISPATCHER SERVLET:

The Spring Web(MVC) framework is designed with a DispatcherServlet that handles all the HTTP requests and responses.

Sequence of events for an incoming HTTP request to DispatcherServlet:

- After receiving an HTTP request, DispatcherServlet goes to handler mapping to call the appropriate Controller.
- The Controller processes request and calls the appropriate service methods based on GET or POST methods. The service method will set model data based on business logic defined and view name is returned to the DispatcherServlet.
- With the help of ViewResolver, DispatcherServlet picks defined view for the request.
- DispatcherServlet passes model data to view when finalized which is rendered on the browser.

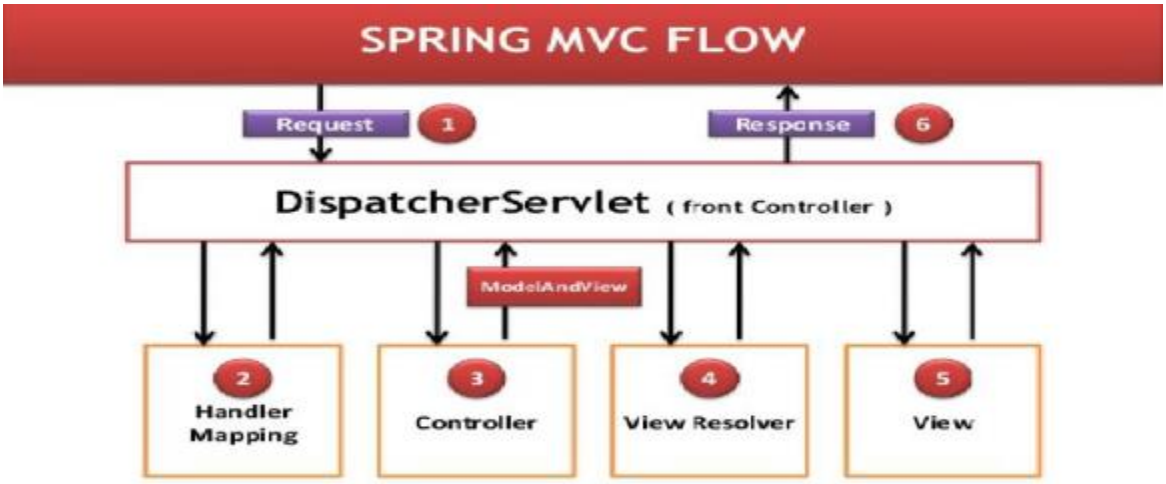

Figure 6.1.6.1

6.2.APACHE MAVEN

6.2.1. BLOCK DIAGRAM

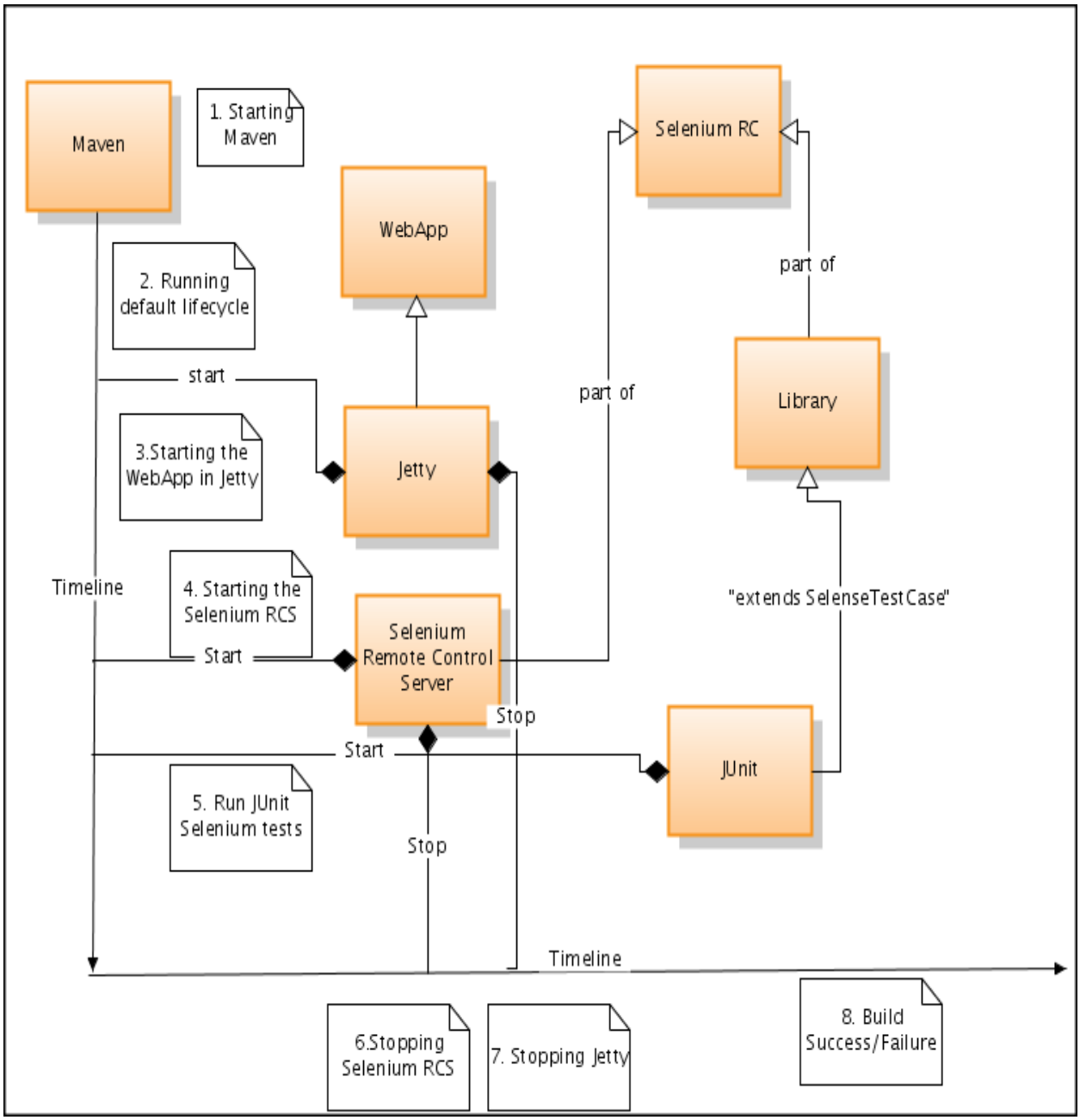

Figure 6.2.1

Maven is a comprehension and software project management tool that allows a developer to develop based on project object model(POM). Maven's key feature is dependency management. It helps to download the required JAR's for the project build.

6.2.2. Dependencies

The core feature of Maven is dependency management. Its dependency handling mechanism is well built up around a coordinate system that identifies the individual artefacts such as software libraries and modules.

A POM provides all the configurationsneeded for a single project. The general configuration covers the project's name, its dependencies and its owner on other projects. One can also configure the individual phases of the project's build process that are implemented as plugins. The POM used for the project is

```
<?xml version="1.0"?>

<projectxsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://maven.apache.org/POM/4.0.0"><modelVersion>4.0.0</modelVersion><groupId>chit.maven</groupId><artifactId>ocui</artifactId><packaging>war</packaging><version>0.0.1-SNAPSHOT</version><name>ocui Maven Webapp</name><url>http://maven.apache.org</url><properties><jdk.version>1.7</jdk.version><spring.version>4.3.3.RELEASE</spring.version></properties><dependencies><dependency><groupId>junit</groupId><artifactId>junit</artifactId><version>4.12</version><scope>test</scope></dependency><dependency><groupId>log4j</groupId><artifactId>log4j</artifactId><version>1.2.17</version></dependency><dependency><groupId>org.springframework</groupId><artifactId>spring-web</artifactId><version>4.3.3.RELEASE</version></dependency><dependency><groupId>org.springframework</groupId><artifactId>spring-core</artifactId><version>4.3.3.RELEASE</version></dependency><dependency><groupId>org.springframework</groupId><artifactId>spring-webmvc</artifactId><version>4.3.3.RELEASE</version></dependency><dependency><groupId>com.fasterxml.jackson.core</groupId><artifactId>jackson-core</artifactId><version>2.5.0</version></dependency><dependency><groupId>com.fasterxml.jackson.core</groupId><artifactId>jackson-databind</artifactId><version>2.5.0</version></dependency><dependency><groupId>javax.servlet</groupId><artifactId>servlet-api</artifactId><version>2.5</version><scope>provided</scope></dependency><dependency><groupId>javax.ws.rs</groupId><artifactId>javax.ws.rs-api</artifactId><version>2.0</version></dependency>
```

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->

<dependency><groupId>org.springframework</groupId><artifactId>spring-aop</artifactId><version>4.3.3.RELEASE</version></dependency><dependency><groupId>org.webjars</groupId><artifactId>jquery</artifactId><version>1.12.4</version></dependency><dependency><groupId>org.aspectj</groupId><artifactId>aspectjrt</artifactId><version>1.6.11</version></dependency><dependency><groupId>org.aspectj</groupId><artifactId>aspectjweaver</artifactId><version>1.6.11</version></dependency><dependency><groupId>org.springframework</groupId><artifactId>spring-jdbc</artifactId><version>3.2.1.RELEASE</version></dependency></dependencies><build><finalName>ocui</finalName></build></project>

7)OUTPUT/RESULT

7.1. FIRST PAGE UI

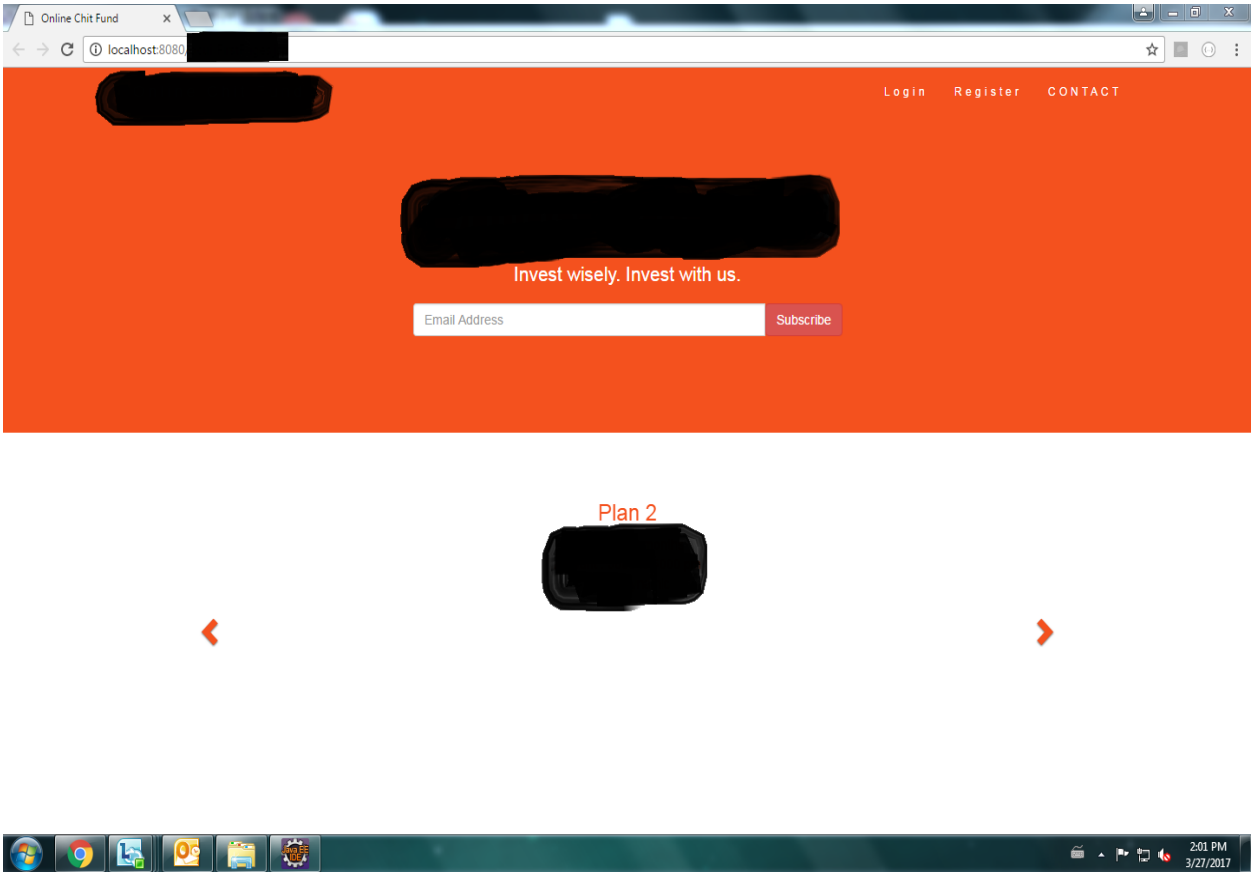

Figure 7.1.1

7.2. LOGIN PAGE

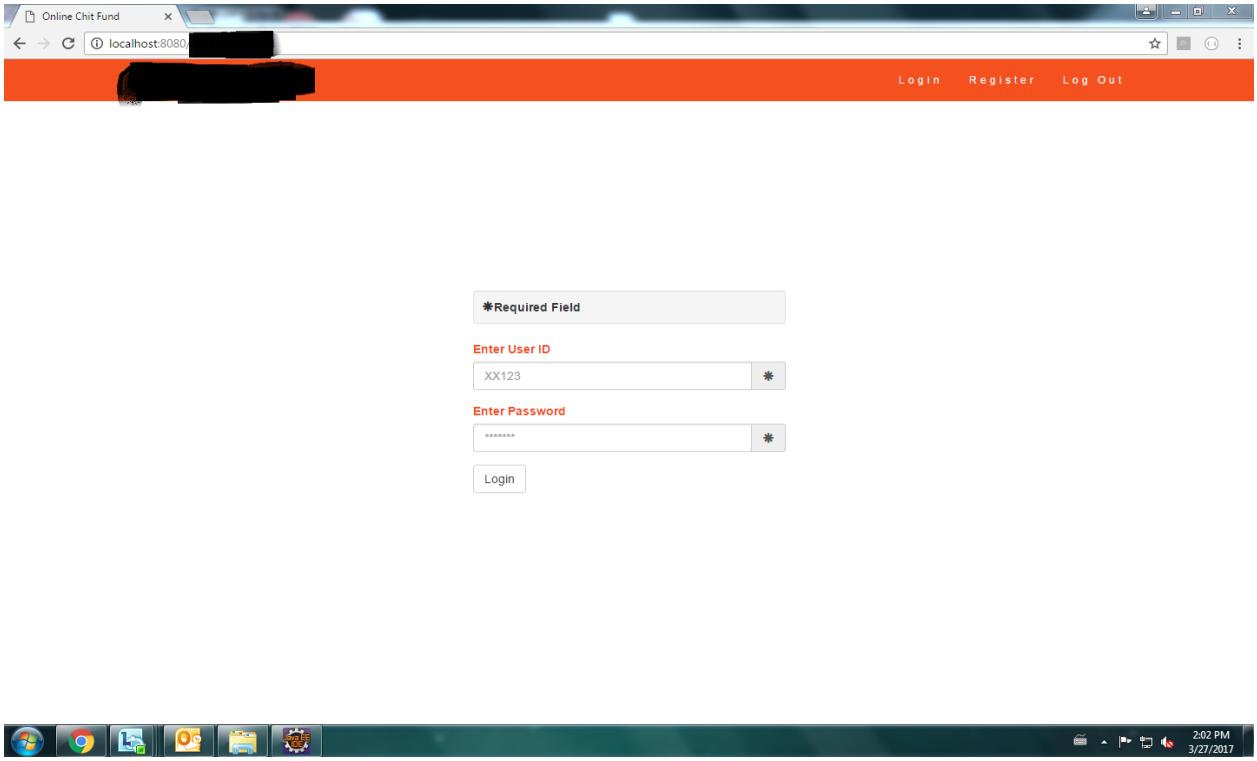

Figure 7.2.1

7.3. ADMIN PAGE

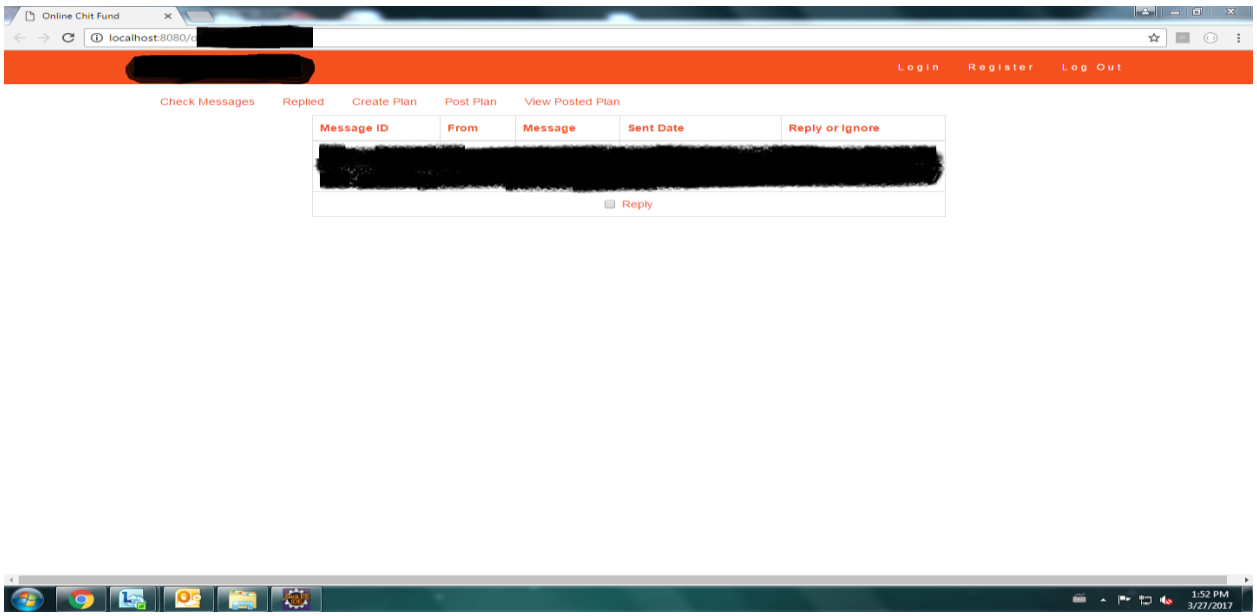

Figure 7.3.1

7.4. USER PAGE

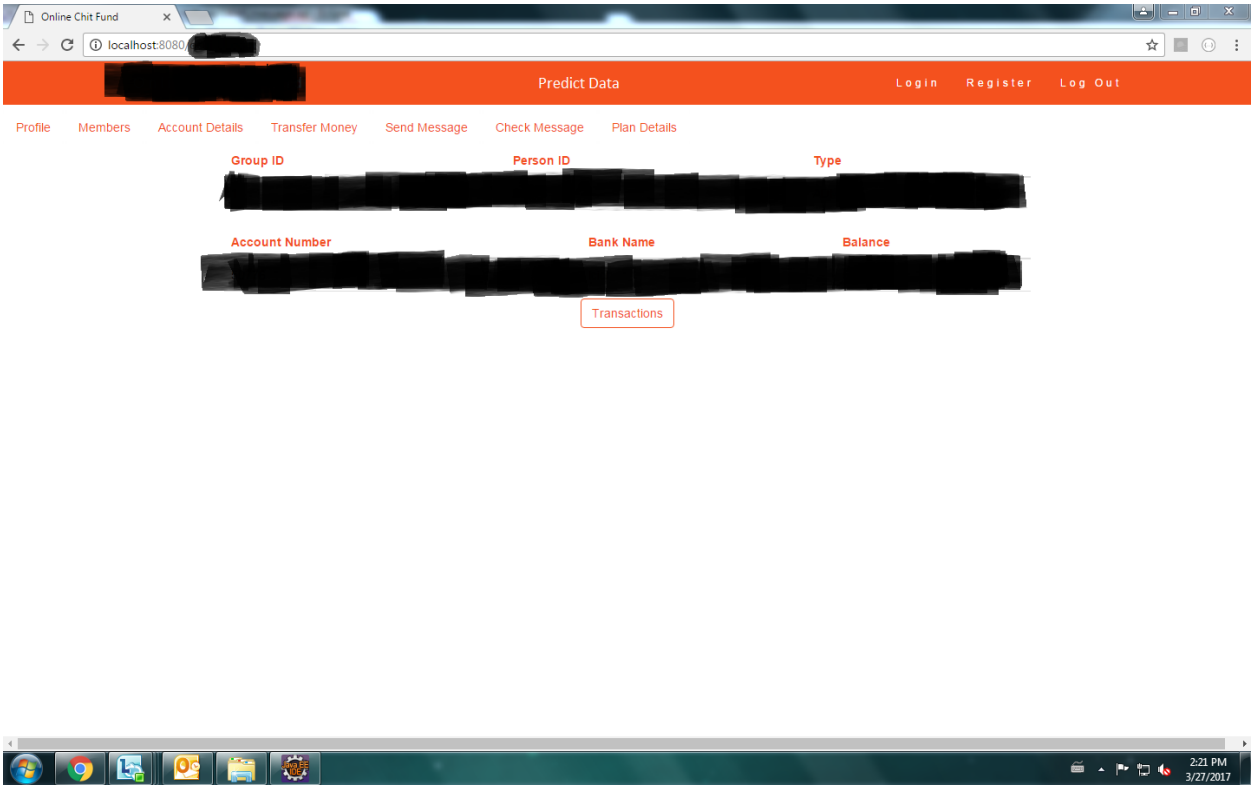

Figure 7.4.1

Admin page

The actor ADMIN makes use of the admin page to manage the overall system. The UI is designed with Bootstrap, JQuery, HTML and Angular JS. Two way data binding in Angular JS is used with RESTful API to interact with the server side. It makes the web page responsive and dynamic.

User page

The actor USER can access the first page, login page and the directed User page. It is coded with Spring framework for flexibility and high modularity. RESTful API get, post and put are used with Spring MVC controllers to access data in no time.

8) CONCLUSION

The system developed has been tested and with various metrics. The main aim of this project is to ease out the user experience with secured and data prediction facility. It makes the user interface self-explanatory there by increasing the comfort level of the user. The application is very secure since end to end encryption and decryption algorithms are used. It is highly responsive and responds quickly to user commands with Angular JS and RESTful API.

9) References:

REFERENCE:

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