Project – Banking System

Edureka!

Java, Java EE & SOA

Certification Training

Student: Akram M'Tir

Date: 02-01-2018

Table of Contents

1 Learning Objectives, Aim, Purpose, Abstract

Learn and master the basic and advanced concepts of core Java SE & Java EE along with popular frameworks like Hibernate, Spring and SOA. Gain expertise in the concepts like Java Array, Java OOPs, Java Function, Java Loops, Java Collections, Java Thread, Java Servlet, Java Design Patterns, and Web Services using industry use-cases

To apply these new skills and concepts learned, a banking project need to be developed.

As ***requirements definition*** for the project , the following modules should be implemented :

Module 1 : This module accepts user id and password and authenticates the given credentials with the database using hibernate.

Module 2 : By accepting the type of account (SB / Current A/c) and user's details, account will be created.

Module 3 : Perform the debit and credit transactions.

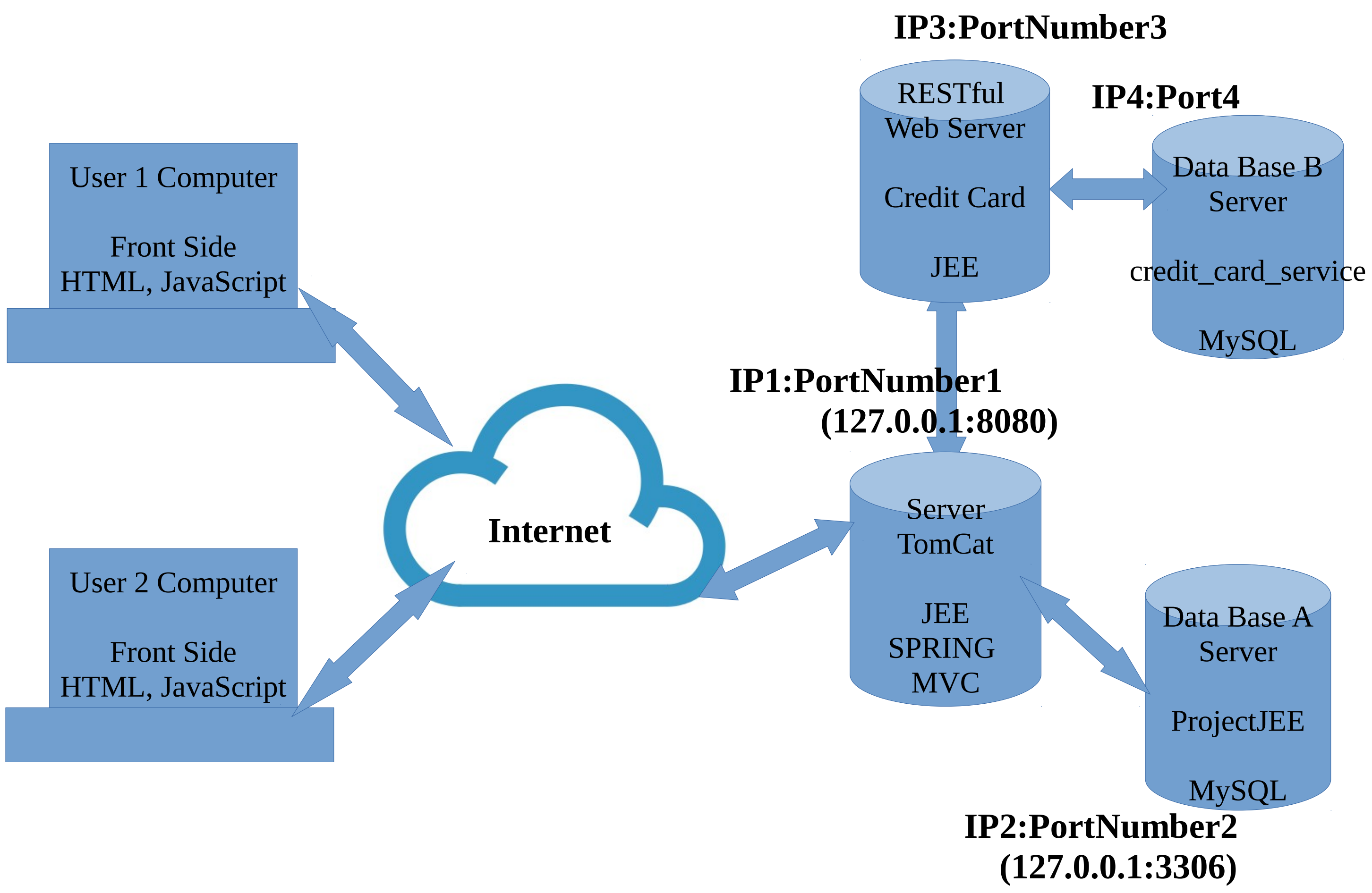
Module 4 : Accept credit card information with the desired details and authorize the credit card amount using web services.

Module 5 : Display transactions of the account based on given date range (From a specific date to a specific date).

Web Services and Project
Learning Objectives - In this module, you will learn SOA and implementation of web services. We will also discuss how to develop a project using Spring and Hibernate. This is a banking project with web services.

2 Architecture and components

- Spring MVC Model - View - Controller
  - Front Side (JSP, HTML, JavaScript, JSTL)
  - Back-Side JEE Spring MVC
- SOA Architecture witth a RESTful Web Service
- MariaDB (MySQL) Data - Base
- JDBC (Transaction and batch processing)
- Server: TomCat 8.5x



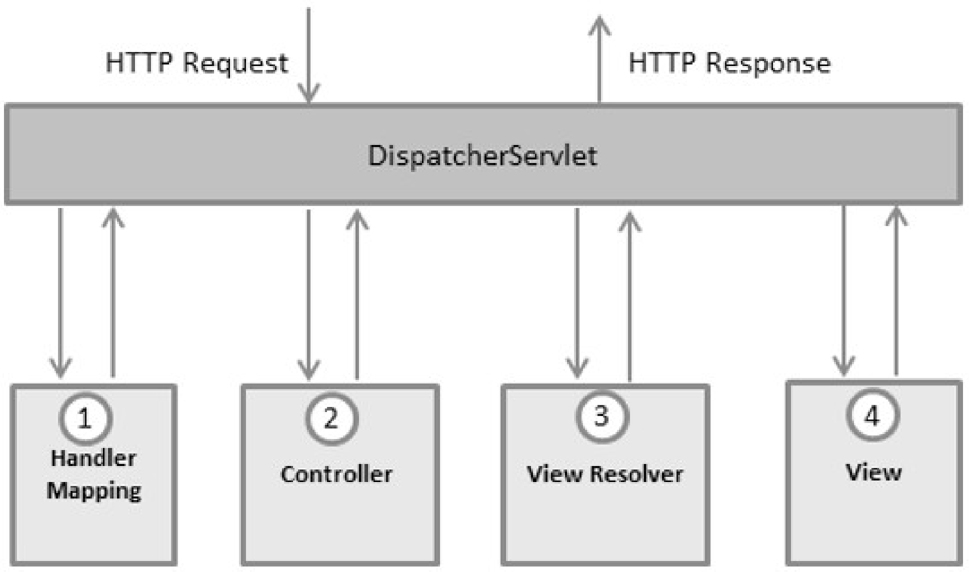
3 MVC Controller , Model, Views

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 DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC DispatcherServlet is illustrated in the following diagram.



Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet

- After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller.
- The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.

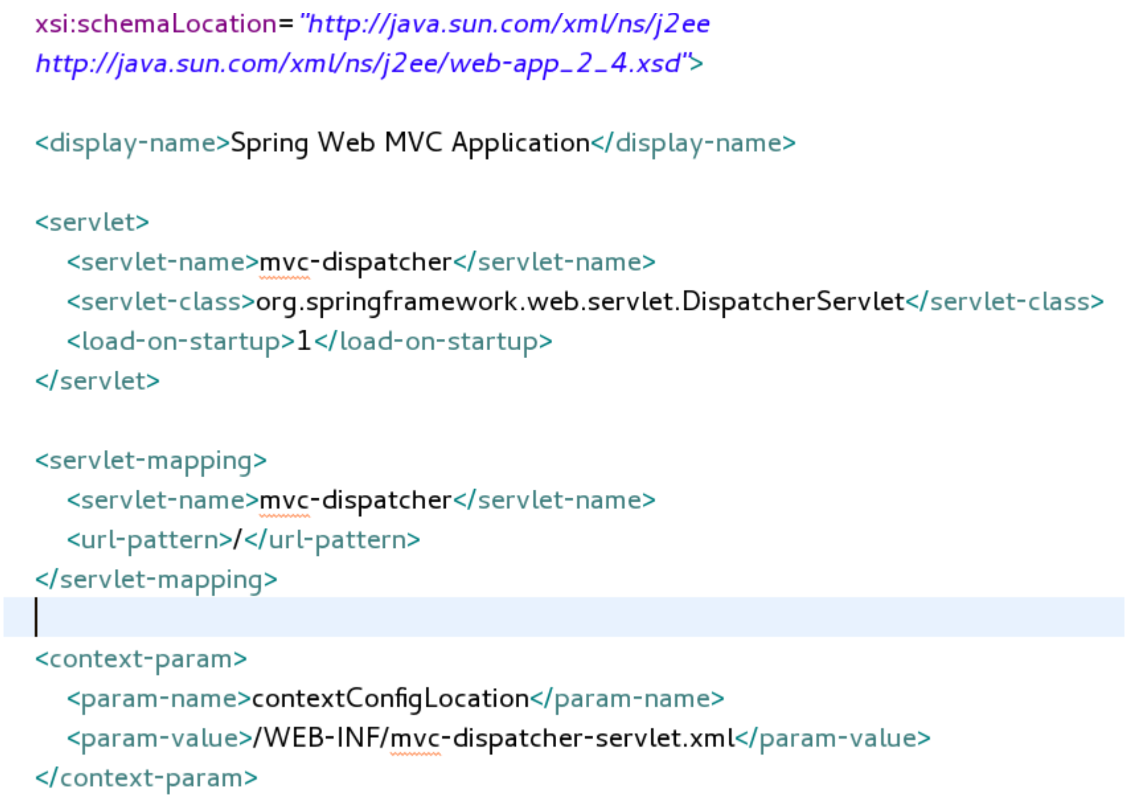
- The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.

- Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

All the above-mentioned components, i.e. HandlerMapping, Controller, and ViewResolver are parts of WebApplicationContext which is an extension of the plainApplicationContext with some extra features necessary for web applications.

Required Configuration

I need to map requests that I want the DispatcherServlet to handle, by using a URL mapping in the web.xml file. The following is an example to show declaration and mapping for mvc-dispatcher DispatcherServlet example



```
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<display-name>Spring Web MVC Application</display-name>

<servlet>
   <servlet-name>mvc-dispatcher</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
   <servlet-name>mvc-dispatcher</servlet-name>
   <url-pattern>/</url-pattern>
</servlet-mapping>

<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
</context-param>
```

The web.xml file is kept in the WebContent/WEB-INF directory of your web application. Upon initialization of mvc-dispatcher DispatcherServlet, the framework will try to load the application context from a file named [servlet-name]-servlet.xml located in the application's WebContent/WEB-INFdirectory. In this case, our file will be mvc-dispatcher-Webservlet.xml.

Next, <servlet-mapping> tag indicates what URLs will be handled by which DispatcherServlet. Here all the HTTP requests are handled by the mvc-dispatcher DispatcherServlet.

We can customize the the application context file name and location by adding the servlet listener ContextLoaderListener in your web.xml file as follows.

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
</context-param>

<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

Now, let us check the required configuration for mvc-dispatcher-servlet.xml file, placed in the web application's WebContent/WEB-INF directory.

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
     http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context-3.0.xsd
     http://www.springframework.org/schema/mvc
     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

        <context:component-scan base-package="com.projectJEE.common" />
        <mvc:annotation-driven />
                <bean id="viewResolver"
                        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                        <property name="prefix">
                                <value>/WEB-INF/jsp/</value>
                        </property>
                        <property name="suffix">
                                <value>.jsp</value>
                        </property>
                </bean>
                <bean class="org.springframework.context.support.ResourceBundleMessageSource"
                        id="messageSource">
                        <property name="basename" value="messages" />
                </bean>
        </beans>
```

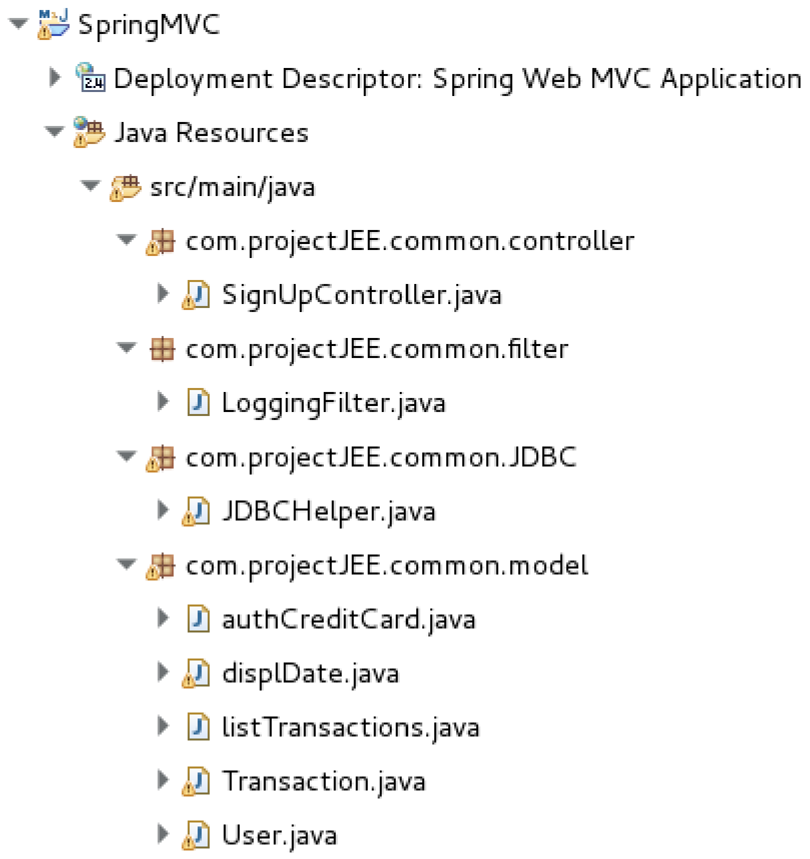
Following are the important points about mvc-dispatcher-servlet.xml file.

- The [servlet-name]-servlet.xml file will be used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.

- The <context:component-scan...> tag will be use to activate Spring MVC annotation scanning capability which allows to make use of annotations like @Controller and @RequestMapping etc.

- The InternalResourceViewResolver will have rules defined to resolve the view names.

As per the above defined rule, a logical view named index is delegated to a view implementation located at /WEB-INF/jsp/index.jsp .



```
SpringMVC
  Deployment Descriptor: Spring Web MVC Application
  Java Resources
    src/main/java
      com.projectJEE.common.controller
        SignUpController.java
      com.projectJEE.common.filter
        LoggingFilter.java
      com.projectJEE.common.JDBC
        JDBCHelper.java
      com.projectJEE.common.model
        authCreditCard.java
        displDate.java
        listTransactions.java
        Transaction.java
        User.java
```

\

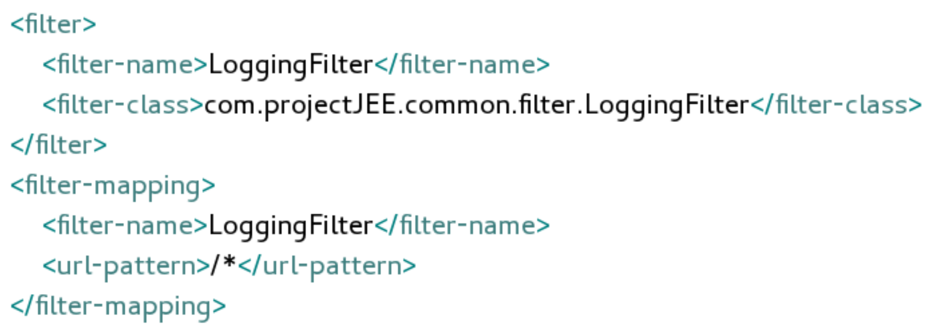
First a ***Maven project*** with a basic web app template is created. Then the the required libraries such as ***Spring MVC*** (spring-core, spring-web, spring-webmvc, hibernate-validator, servlet-api, javax.websocket-api, jstl, mysql-connector-java) are added in the ***pom.xml***. All dependencies are managed automatically.

DAO Data Access Object library *:* All the data base functions are located in the *com.projectJEE.common.JDBC* package in the ***JDBCHelper.java* class** (open, close connection , CRUD operations…)

All ***model data objects*** exchanged between the views and the controller are placed in the package *com.projectJEE.common.model*.

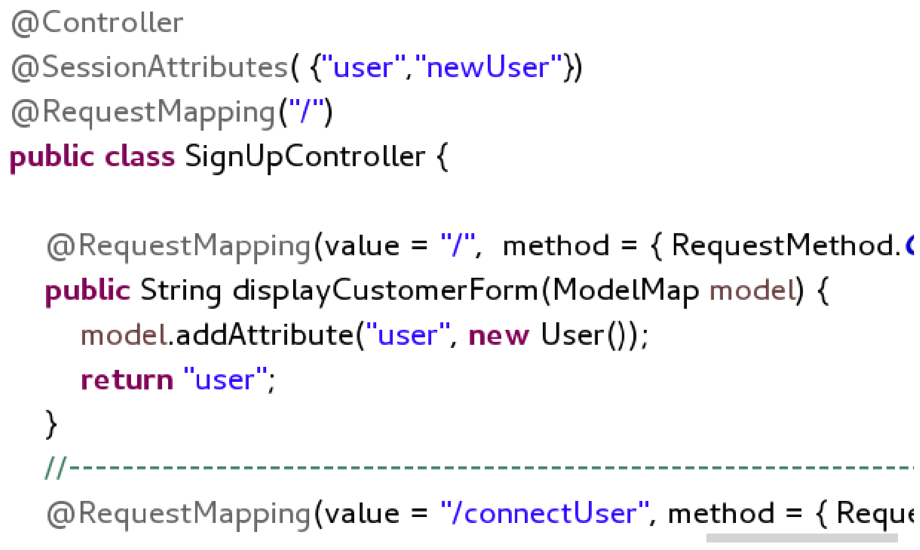
The view resolver is located in the *mvc-dispatcher-servlet.xml*. In this project, all the ***views*** are ***jsp pages*** located under the jsp folder.

*A **filter LoggingFilter.java** in the package com.projectJEE.common.filter is used to authenticate the user each request.*

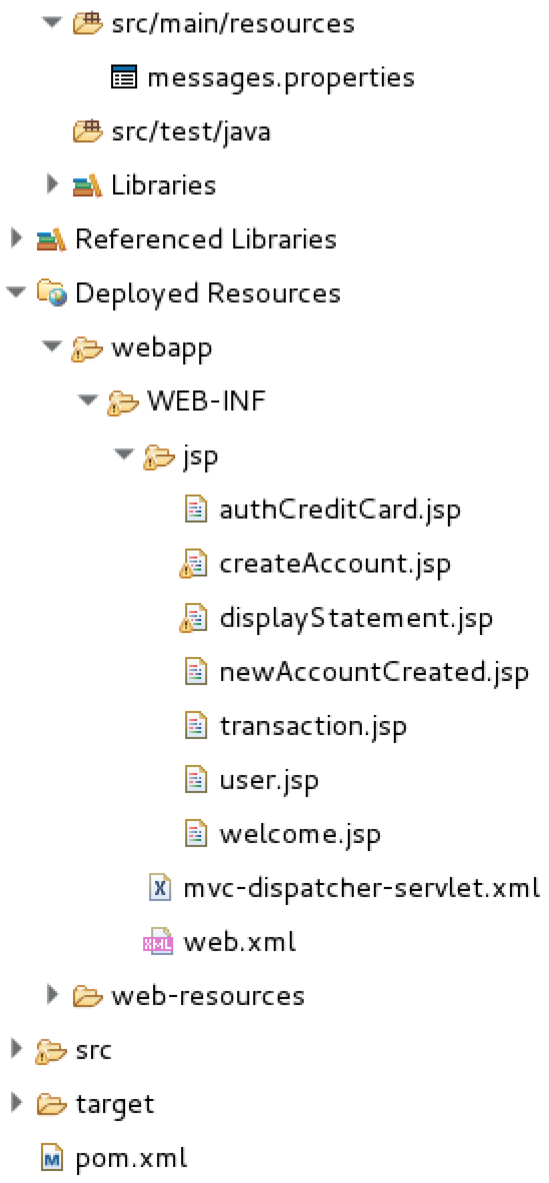


```
<filter>
   <filter-name>LoggingFilter</filter-name>
   <filter-class>com.projectJEE.common.filter.LoggingFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>LoggingFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

The ***controller*** which will process all the requests is placed in the *SignUpController.java* class in the *com.projectJEE.common.controller*.



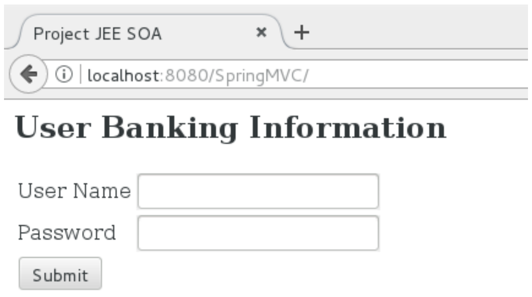
```
@Controller
@SessionAttributes( {"user","newUser"})
@RequestMapping("/")
public class SignUpController {

   @RequestMapping(value = "/",  method = { RequestMethod.
   public String displayCustomerForm(ModelMap model) {
      model.addAttribute("user", new User());
      return "user";
   }
   //-----------------------------------------------------------------
   @RequestMapping(value = "/connectUser", method = { Reque
```



```
▾ src/main/resources
    messages.properties
  src/test/java
▸ Libraries
▸ Referenced Libraries
▾ Deployed Resources
  ▾ webapp
    ▾ WEB-INF
      ▾ jsp
          authCreditCard.jsp
          createAccount.jsp
          displayStatement.jsp
          newAccountCreated.jsp
          transaction.jsp
          user.jsp
          welcome.jsp
        mvc-dispatcher-servlet.xml
        web.xml
  ▸ web-resources
▸ src
▸ target
  pom.xml
```

4 Web Application: Functionality and User interface

4.1 User logging screen



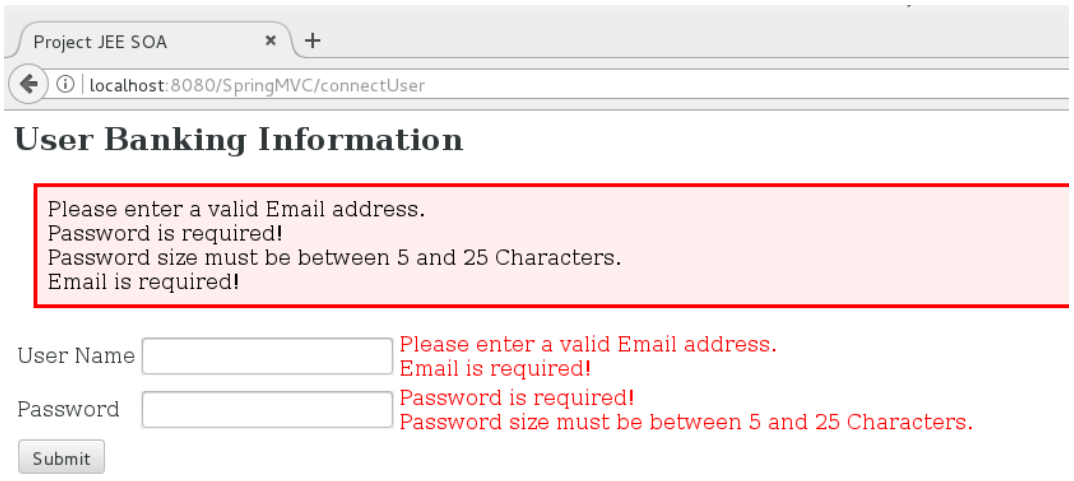
User Banking Information

User Name

Password

Submit

In this logging interface screen, the user is prompted to enter his credential, namely the user id and the Password. The user id is in the form of email pattern. The password is required and the size must be between 5 and 25 Characters. The validation is performed at the server-side using JSR 303 validation as shown bellow.



User Banking Information

Please enter a valid Email address.
Password is required!
Password size must be between 5 and 25 Characters.
Email is required!

User Name — Please enter a valid Email address. Email is required!

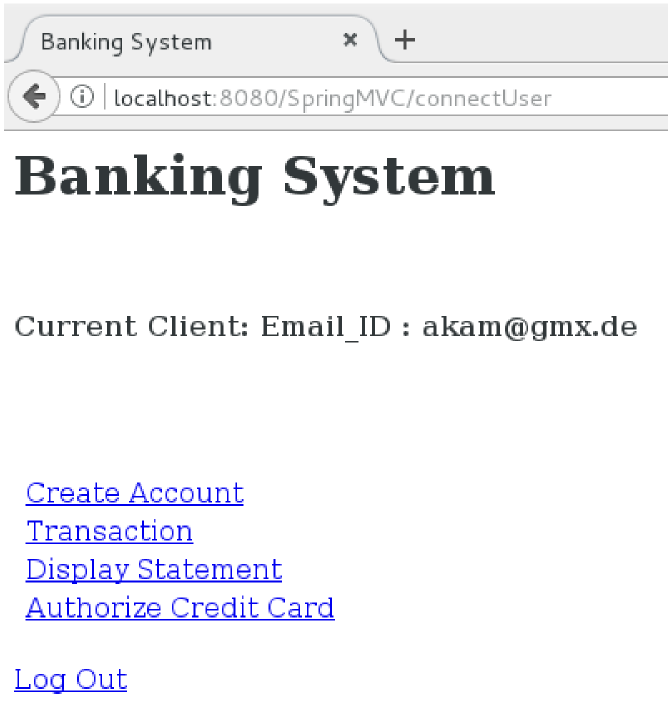
Password — Password is required! Password size must be between 5 and 25 Characters.

Submit

The credential are verified against the data stored in the Data-Base, in the table user_pass. The password could be stored in the form of a MD5 or SHA-hash in case of data leakage.

4.2 Main Menu screen

Once the user has entered the correct credential, he/she is presented with the main menu offering various banking operations such as account creation, money transfer via transaction, list of transactions filtered by date and so on.



Banking System

Current Client: Email_ID : akam@gmx.de

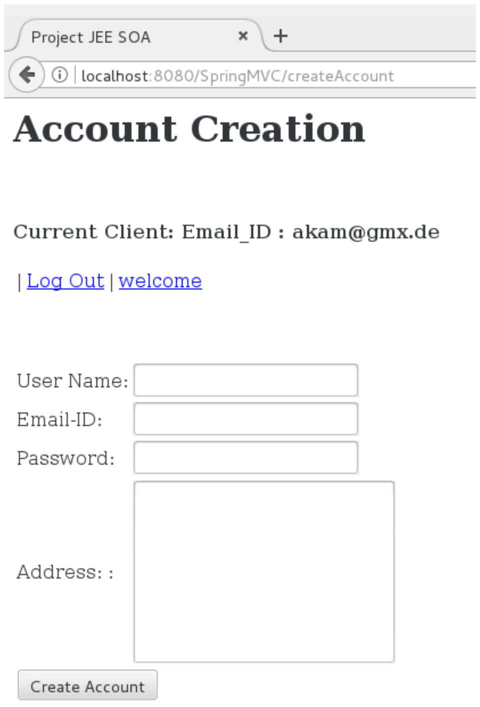
Create Account
Transaction
Display Statement
Authorize Credit Card

Log Out

In case of incorrect credential, the user is presented with the user logging page again and prompted to enter a valid user id and password.

4.3 Account Creation

The user id (email) and password are required for the creation of a new account.



Project JEE SOA

localhost:8080/SpringMVC/createAccount

Account Creation

Current Client: Email_ID : akam@gmx.de

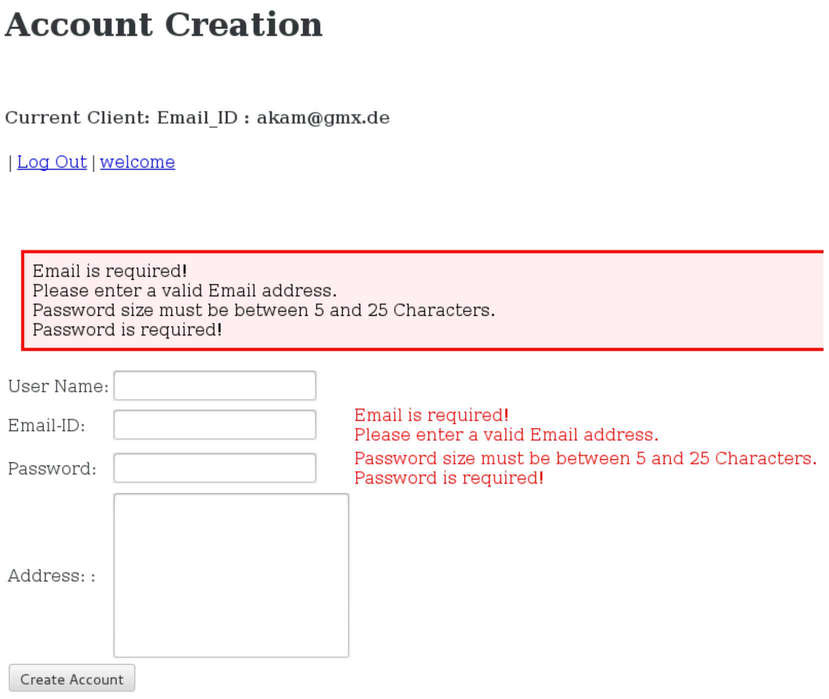
| Log Out | welcome

User Name:

Email-ID:

Password:

Address: :



Account Creation

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Email is required!
Please enter a valid Email address.
Password size must be between 5 and 25 Characters.
Password is required!

User Name:

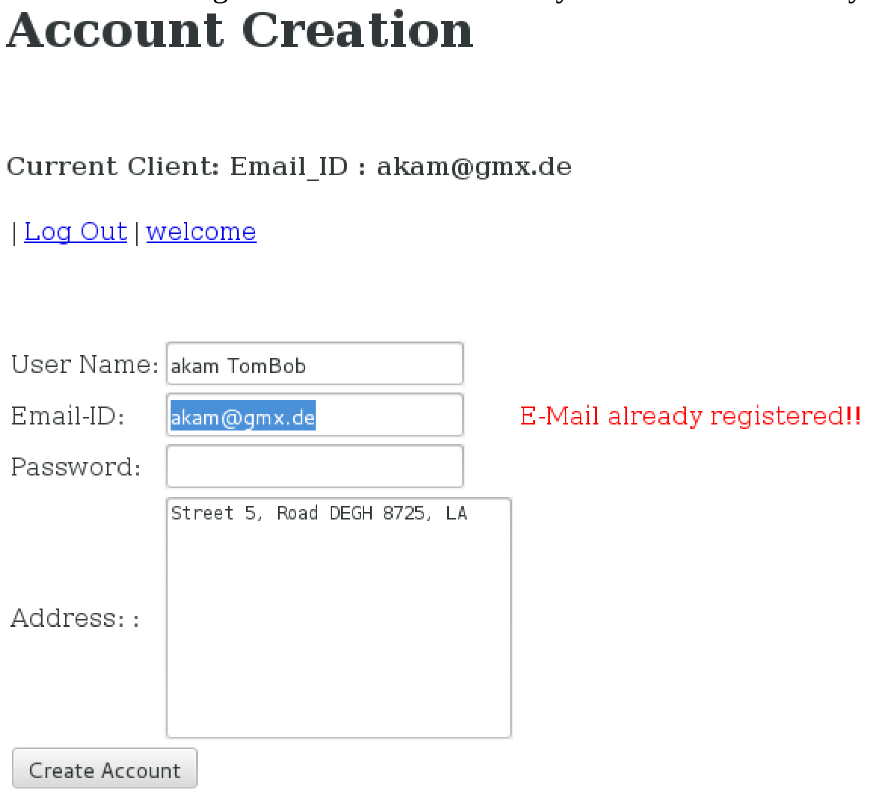
Email-ID:

Password:

Email is required!
Please enter a valid Email address.
Password size must be between 5 and 25 Characters.
Password is required!

Address: :

The Email_ID field is checked against the database to verify if this email is already registered.



Account Creation

Current Client: Email_ID : akam@gmx.de

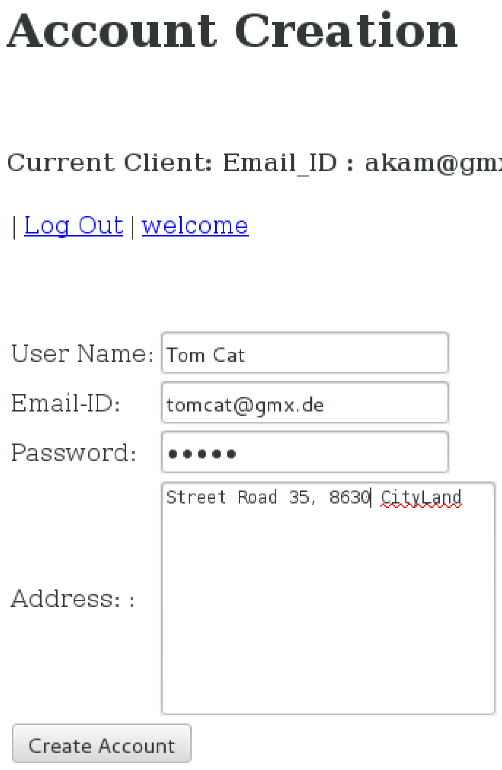
| Log Out | welcome

User Name: akam TomBob

Email-ID: akam@gmx.de   E-Mail already registered!!

Password:

Address: : Street 5, Road DEGH 8725, LA



Account Creation

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

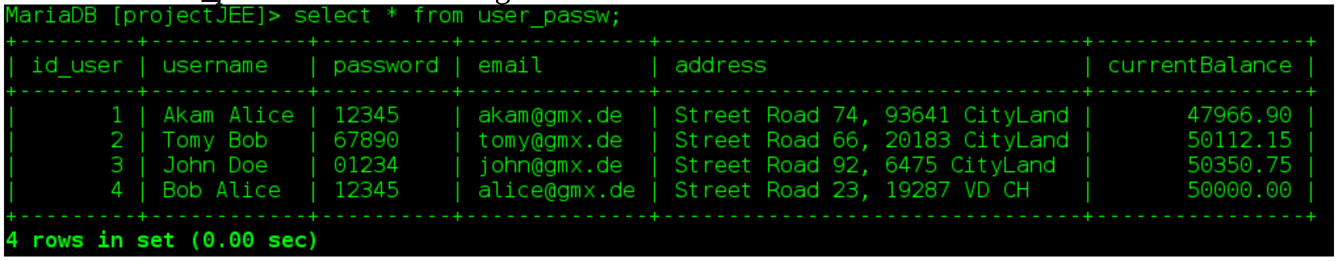
User Name: Tom Cat

Email-ID: tomcat@gmx.de

Password: •••••

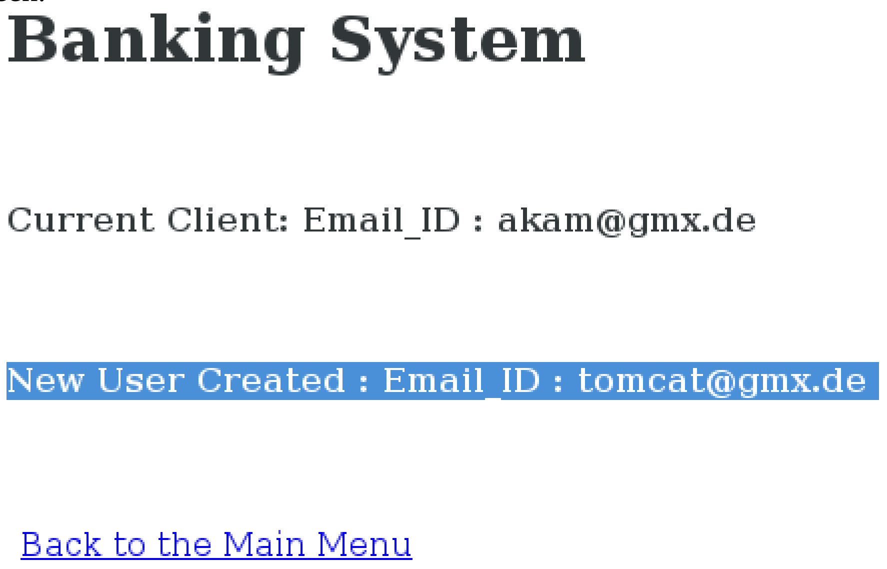
Address: : Street Road 35, 8630 CityLand

The table “*user_passw*” before creating a new user.



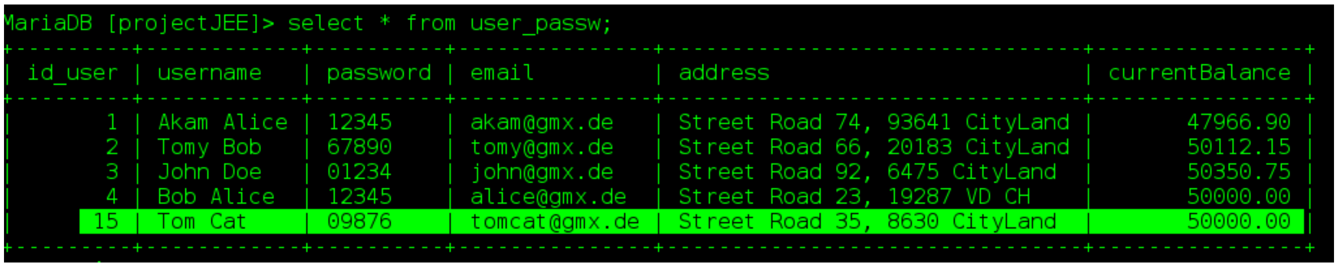
```
MariaDB [projectJEE]> select * from user_passw;
+---------+------------+----------+--------------+----------------------------------+----------------+
| id_user | username   | password | email        | address                          | currentBalance |
+---------+------------+----------+--------------+----------------------------------+----------------+
|       1 | Akam Alice | 12345    | akam@gmx.de  | Street Road 74, 93641 CityLand |       47966.90 |
|       2 | Tomy Bob   | 67890    | tomy@gmx.de  | Street Road 66, 20183 CityLand |       50112.15 |
|       3 | John Doe   | 01234    | john@gmx.de  | Street Road 92, 6475 CityLand  |       50350.75 |
|       4 | Bob Alice  | 12345    | alice@gmx.de | Street Road 23, 19287 VD CH    |       50000.00 |
+---------+------------+----------+--------------+----------------------------------+----------------+
4 rows in set (0.00 sec)
```

After pressing/clicking the “Create Account” button, the user will be presented with the following confirmation screen.



Banking System
Current Client: Email_ID : akam@gmx.de
New User Created : Email_ID : tomcat@gmx.de
Back to the Main Menu

Here a new account “tomcat@gmx.de” has been created. This could be verified by listing the users in the table as shown bellow.



```
MariaDB [projectJEE]> select * from user_passw;
+---------+------------+----------+---------------+----------------------------------+----------------+
| id_user | username   | password | email         | address                          | currentBalance |
+---------+------------+----------+---------------+----------------------------------+----------------+
|       1 | Akam Alice | 12345    | akam@gmx.de   | Street Road 74, 93641 CityLand |       47966.90 |
|       2 | Tomy Bob   | 67890    | tomy@gmx.de   | Street Road 66, 20183 CityLand |       50112.15 |
|       3 | John Doe   | 01234    | john@gmx.de   | Street Road 92, 6475 CityLand  |       50350.75 |
|       4 | Bob Alice  | 12345    | alice@gmx.de  | Street Road 23, 19287 VD CH    |       50000.00 |
|      15 | Tom Cat    | 09876    | tomcat@gmx.de | Street Road 35, 8630 CityLand  |       50000.00 |
+---------+------------+----------+---------------+----------------------------------+----------------+
```

By default each new user created receives 50000 as current balance.

4.4 Debit – Credit

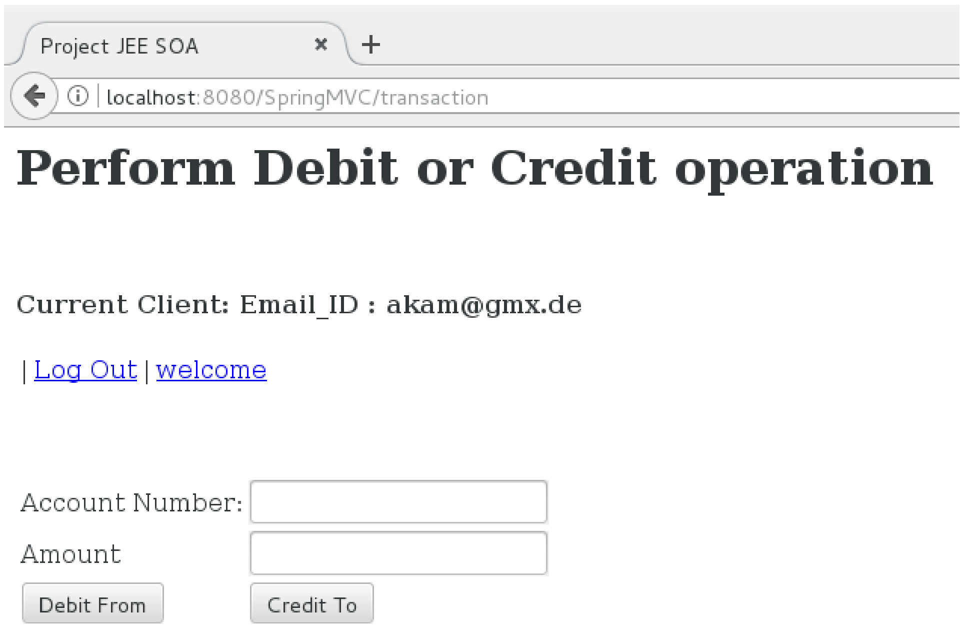
Debit – Credit from the current account to another account (transactions)

It is crucial for a banking system, to maintain the correct state of the data. When multiple related operations are performed on the database, and if at least one operation fails then it is required to bring the database to its original state (rollback). For example when we transfer the money from one account to the another account, in other words when we debit an account to credit another account, it is crucial to ensure that either both operations (debit and credit) are performed or neither of these operations are executed.

I have decided to use ***JDBC API*** which is the industry standard for database-independent connectivity The JDBC driver is provided by the database vendor and JDBC offers ***Transaction management***.

I have grouped all related instructions to the transaction in one ***Batch process*** and these operations will be executed as single operation. By using *Commit()*, changes done in the database will be made permanent. By using *RollBack()* changes done in the database will be removed.

Clicking the Transaction link in the main menu will bring the following screen interface where we can perform debit and credit operations. Two fields should be specified. The first field is the account number in the form of an Email_ID. The second field is the amount to be debited or credited.



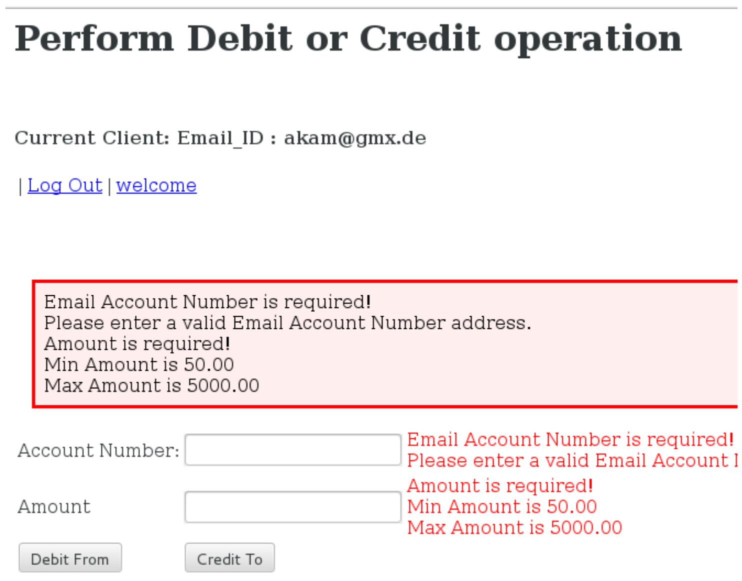
Perform Debit or Credit operation

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Account Number:
Amount

Here are the requirements definitions for the debit and credit operations.



Perform Debit or Credit operation

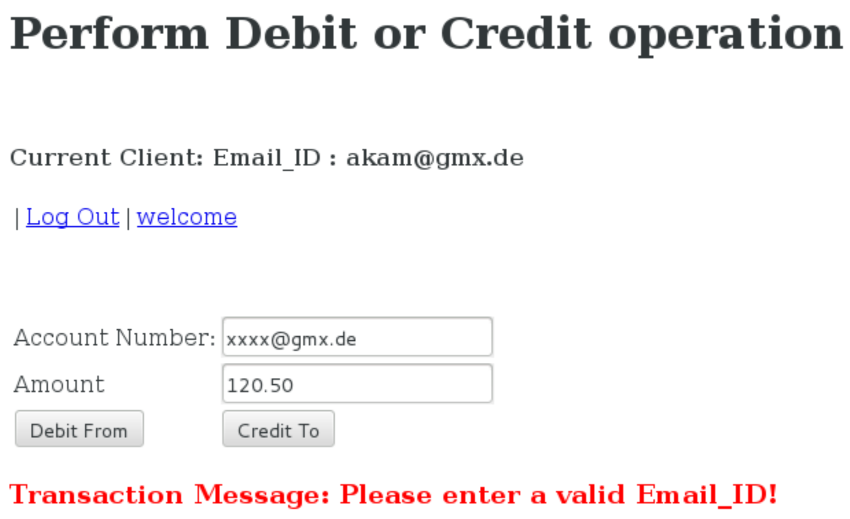
Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Email Account Number is required!
Please enter a valid Email Account Number address.
Amount is required!
Min Amount is 50.00
Max Amount is 5000.00

Account Number:		Email Account Number is required! Please enter a valid Email Account I
Amount		Amount is required! Min Amount is 50.00 Max Amount is 5000.00
Debit From	Credit To	

The validation is performed at the server-side using JSR 303 validation as shown. The account number should follow an Email pattern and the amount filed should between 50 to 5000.



Perform Debit or Credit operation

Current Client: Email_ID : akam@gmx.de

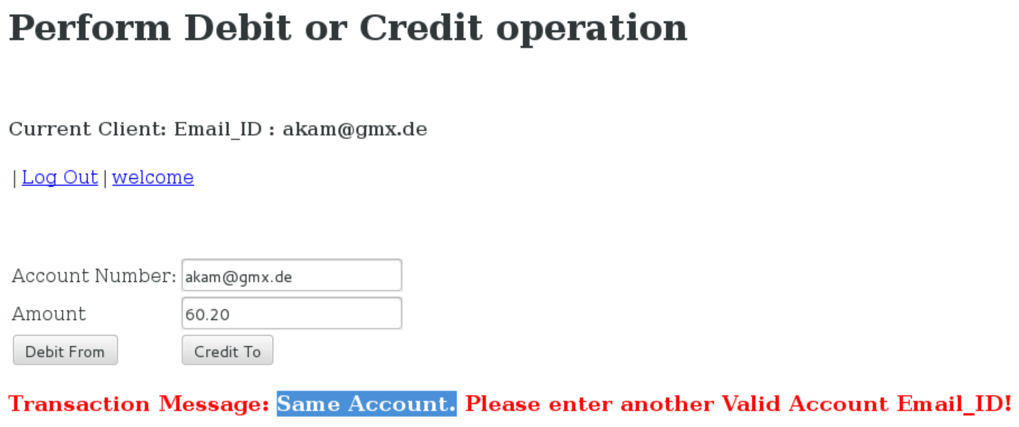
| Log Out | welcome

Account Number:	xxxx@gmx.de
Amount	120.50
Debit From	Credit To

Transaction Message: Please enter a valid Email_ID!

The account number is then checked against the database to verify the user validity.

If the account number (Email_ID) specified in the first field is the same as the logged current user, then no operation will be performed and the application will ask the user to enter another account.



Perform Debit or Credit operation

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Account Number: akam@gmx.de

Amount 60.20

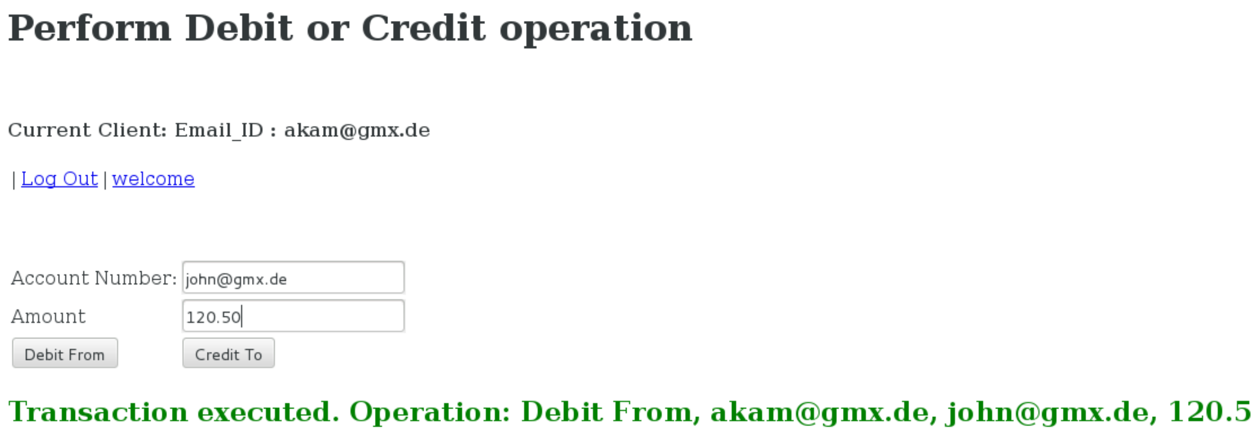
Transaction Message: Same Account. Please enter another Valid Account Email_ID!

If the Account number (Email_ID) is registered as a valid account in the Database and the amount is between 50 to 5000, then Debit and Credit operation could be performed .

Pressing the "*Debit From*" button will debit the account specified in the first field and credit the current logged client.

Pressing the "*Credit To*" button, the account specified in the first field will be credited and the current logged account will be debited.

Below is an example where the "Debit From" button is pressed.



Perform Debit or Credit operation

Current Client: Email_ID : akam@gmx.de

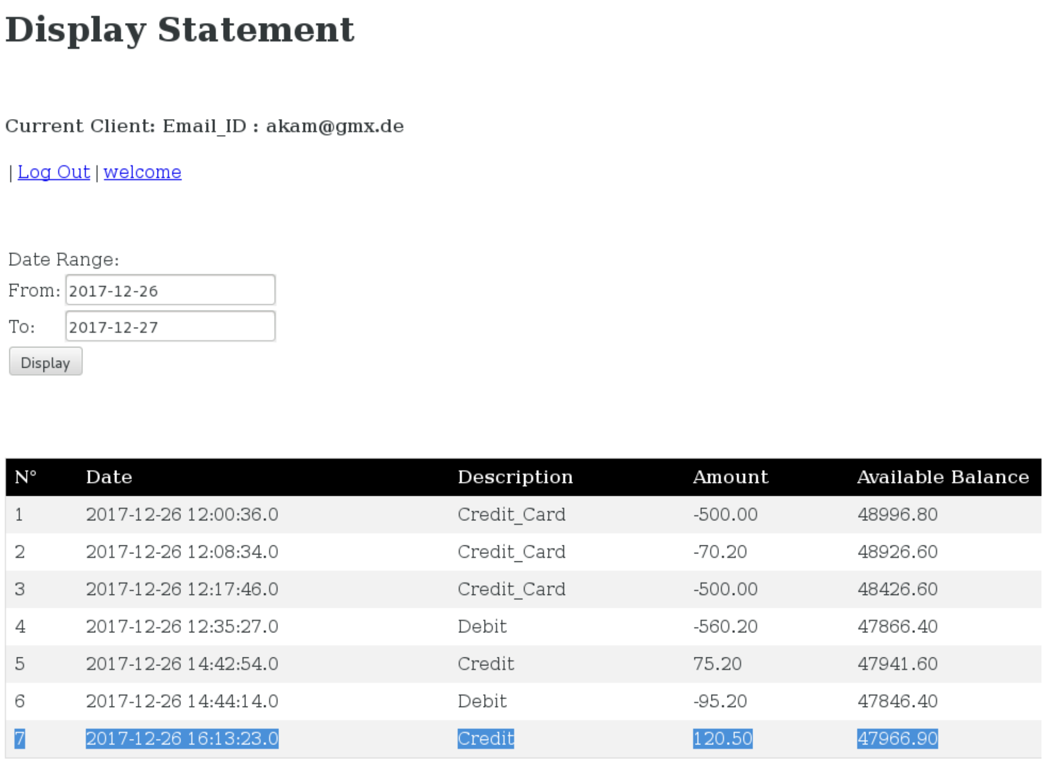
| Log Out | welcome

Account Number: john@gmx.de

Amount 120.50

Transaction executed. Operation: Debit From, akam@gmx.de, john@gmx.de, 120.5

In this example the account john@dmx.de is debited by an amount of 120.5 and the current account akam@gmx.de is credited by same amount 120.5. These operations could be verified by listing the operations as shown bellow.


Display Statement

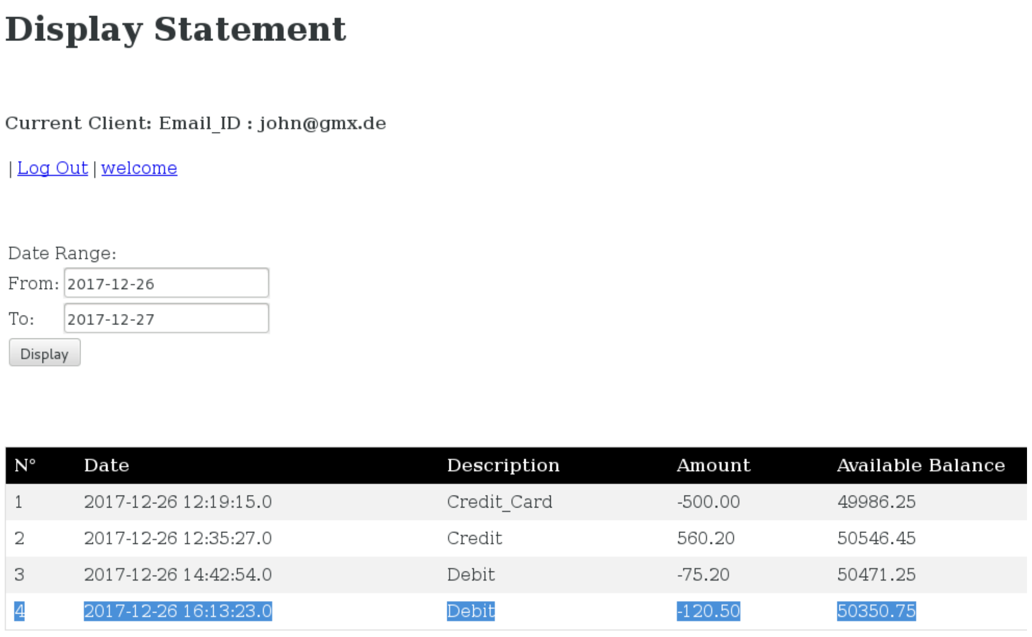
Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Date Range:
From: 2017-12-26
To: 2017-12-27
Display

N°	Date	Description	Amount	Available Balance
1	2017-12-26 12:00:36.0	Credit_Card	-500.00	48996.80
2	2017-12-26 12:08:34.0	Credit_Card	-70.20	48926.60
3	2017-12-26 12:17:46.0	Credit_Card	-500.00	48426.60
4	2017-12-26 12:35:27.0	Debit	-560.20	47866.40
5	2017-12-26 14:42:54.0	Credit	75.20	47941.60
6	2017-12-26 14:44:14.0	Debit	-95.20	47846.40
7	2017-12-26 16:13:23.0	Credit	120.50	47966.90

The current account, akam@gmx.de has been credited by 120.50.


Display Statement

Current Client: Email_ID : john@gmx.de

| Log Out | welcome

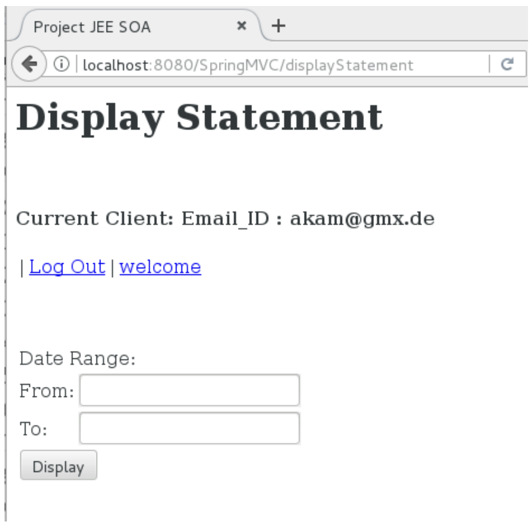
Date Range:
From: 2017-12-26
To: 2017-12-27
Display

N°	Date	Description	Amount	Available Balance
1	2017-12-26 12:19:15.0	Credit_Card	-500.00	49986.25
2	2017-12-26 12:35:27.0	Credit	560.20	50546.45
3	2017-12-26 14:42:54.0	Debit	-75.20	50471.25
4	2017-12-26 16:13:23.0	Debit	-120.50	50350.75

The specified account, john@gmx.de has been debited by 120.50.
(*) Notice that we could also define simpler requirements, where just the specified account number (Email_ID) is either credited or debited.

4.5 Display Statement

List of the transactions filtered by date for the current logged user.



Project JEE SOA

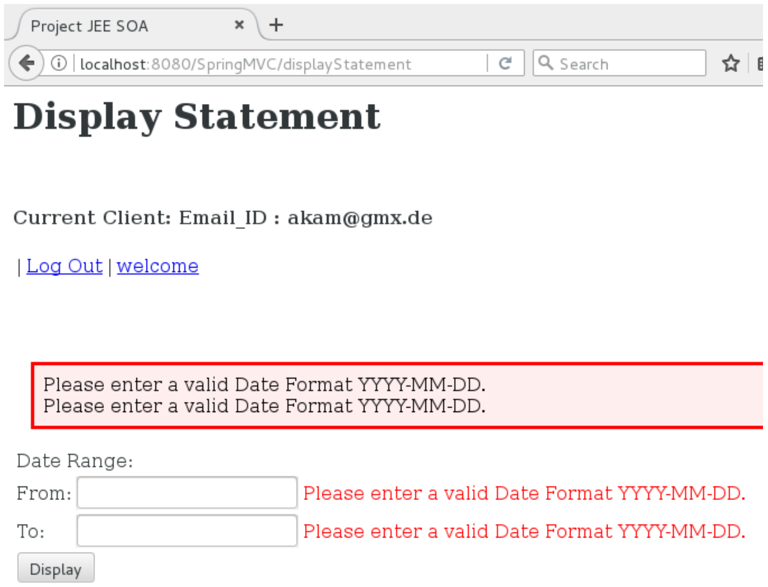
localhost:8080/SpringMVC/displayStatement

Display Statement

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Date Range:
From:
To:
Display



Project JEE SOA

localhost:8080/SpringMVC/displayStatement

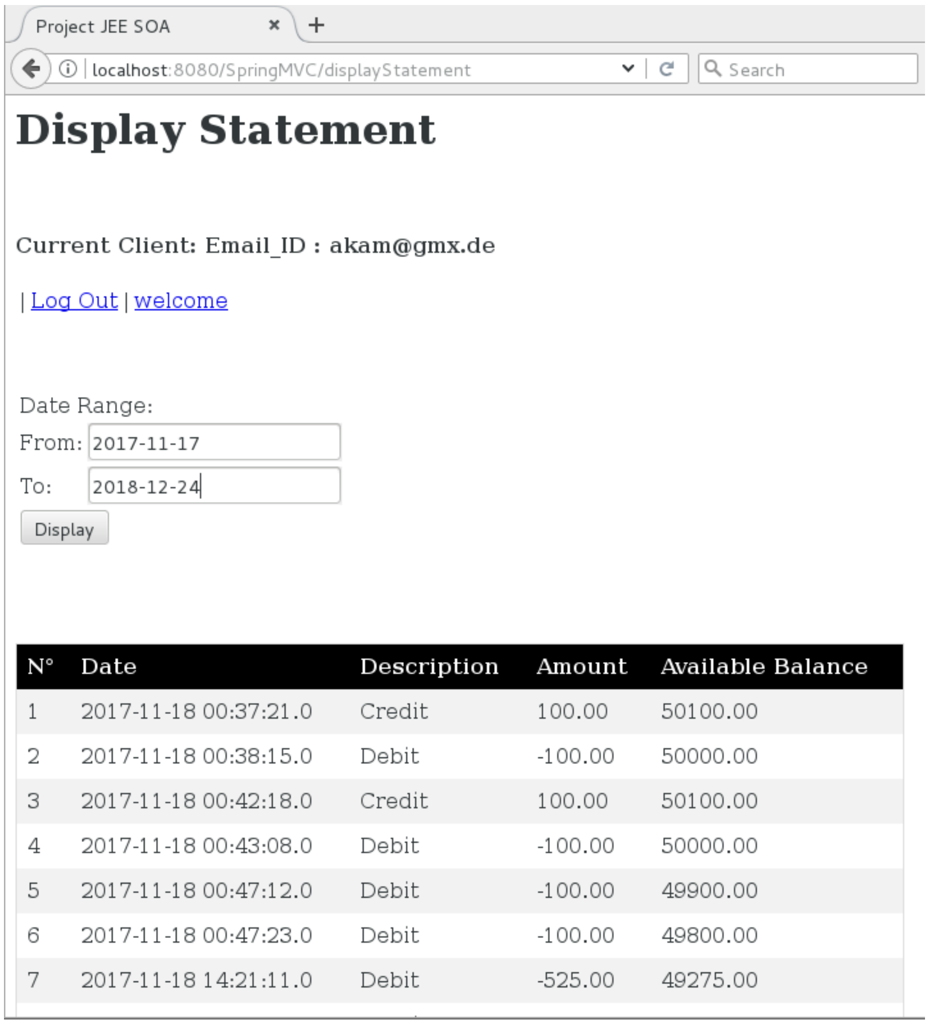
Display Statement

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Please enter a valid Date Format YYYY-MM-DD.
Please enter a valid Date Format YYYY-MM-DD.

Date Range:
From: Please enter a valid Date Format YYYY-MM-DD.
To: Please enter a valid Date Format YYYY-MM-DD.
Display



Project JEE SOA

localhost:8080/SpringMVC/displayStatement

Display Statement

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

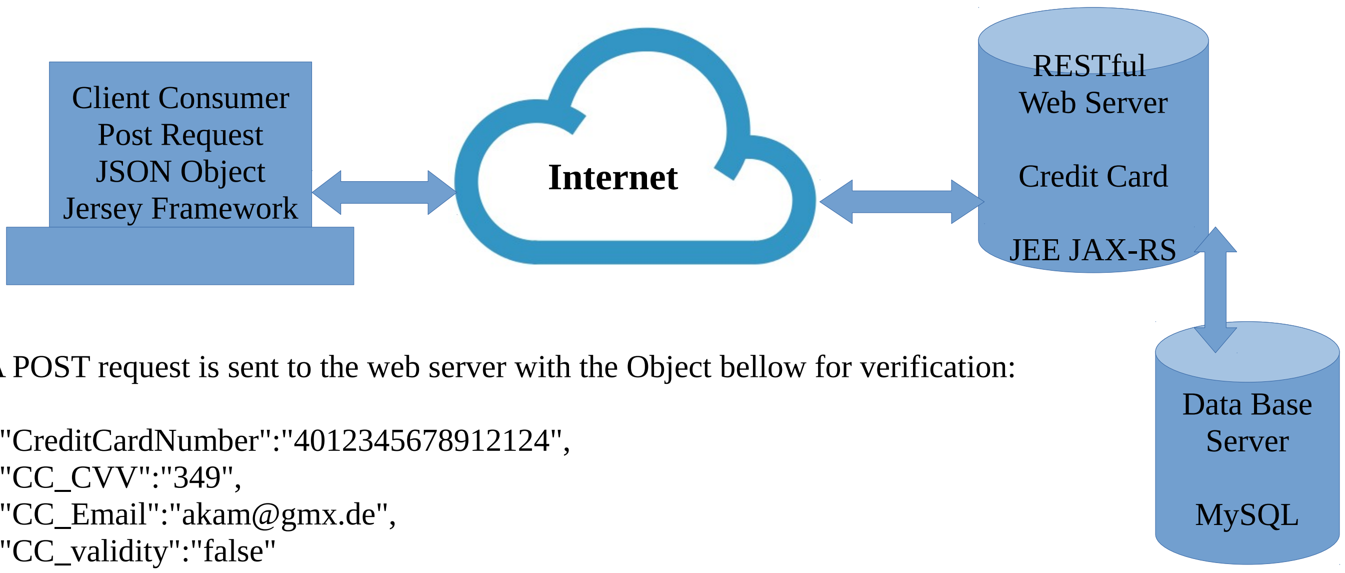
Date Range:
From: 2017-11-17
To: 2018-12-24
Display

N°	Date	Description	Amount	Available Balance
1	2017-11-18 00:37:21.0	Credit	100.00	50100.00
2	2017-11-18 00:38:15.0	Debit	-100.00	50000.00
3	2017-11-18 00:42:18.0	Credit	100.00	50100.00
4	2017-11-18 00:43:08.0	Debit	-100.00	50000.00
5	2017-11-18 00:47:12.0	Debit	-100.00	49900.00
6	2017-11-18 00:47:23.0	Debit	-100.00	49800.00
7	2017-11-18 14:21:11.0	Debit	-525.00	49275.00

4.6 Credit Card Transaction

Here are the requirements for the credit card transaction:
For the Credit Card transaction, a RESTful Web Service has been created to check the validity of the credit card, such as the Card number, CVV code, the name (card holder or email_id). If the credit card details are found correct, then the transaction is accepted. JSON objects will be exchanged between the main banking system application and the Credit Card Web Service.

Jersey RESTful Web Services framework is used for developing the RESTful Web Services in Java. It provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation.

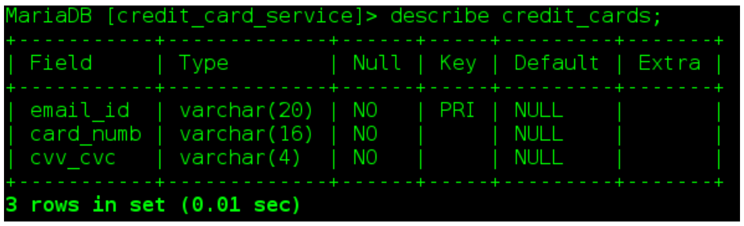


A POST request is sent to the web server with the Object bellow for verification:

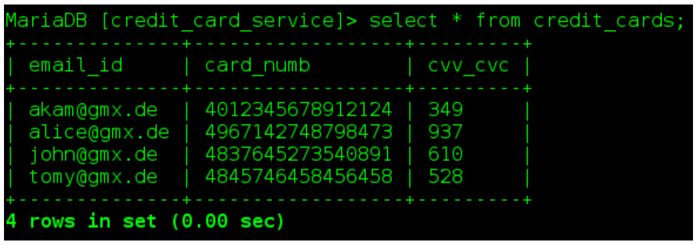
```
{
 "CreditCardNumber":"4012345678912124",
 "CC_CVV":"349",
 "CC_Email":"akam@gmx.de",
 "CC_validity":"false"
}
```

The RESTful web server returns the following Object after check:

```
{
   "CC_CVV": "349",
   "CC_Email": "akam@gmx.de",
   "CC_validity": "true",
   "creditCardNumber": "4012345678912124"
}
```

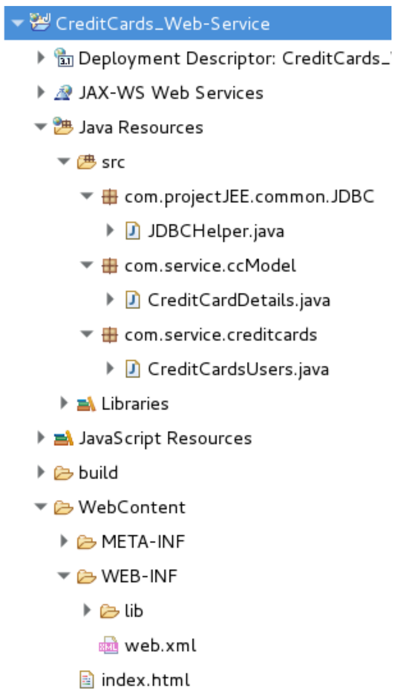


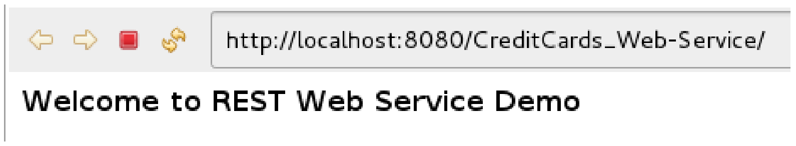
```
MariaDB [credit_card_service]> describe credit_cards;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| email_id  | varchar(20) | NO   | PRI | NULL    |       |
| card_numb | varchar(16) | NO   |     | NULL    |       |
| cvv_cvc   | varchar(4)  | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```



```
MariaDB [credit_card_service]> select * from credit_cards;
+--------------+------------------+---------+
| email_id     | card_numb        | cvv_cvc |
+--------------+------------------+---------+
| akam@gmx.de  | 4012345678912124 | 349     |
| alice@gmx.de | 4967142748798473 | 937     |
| john@gmx.de  | 4837645273540891 | 610     |
| tomy@gmx.de  | 4845746458456458 | 528     |
+--------------+------------------+---------+
4 rows in set (0.00 sec)
```

First start the RESTful Web Service

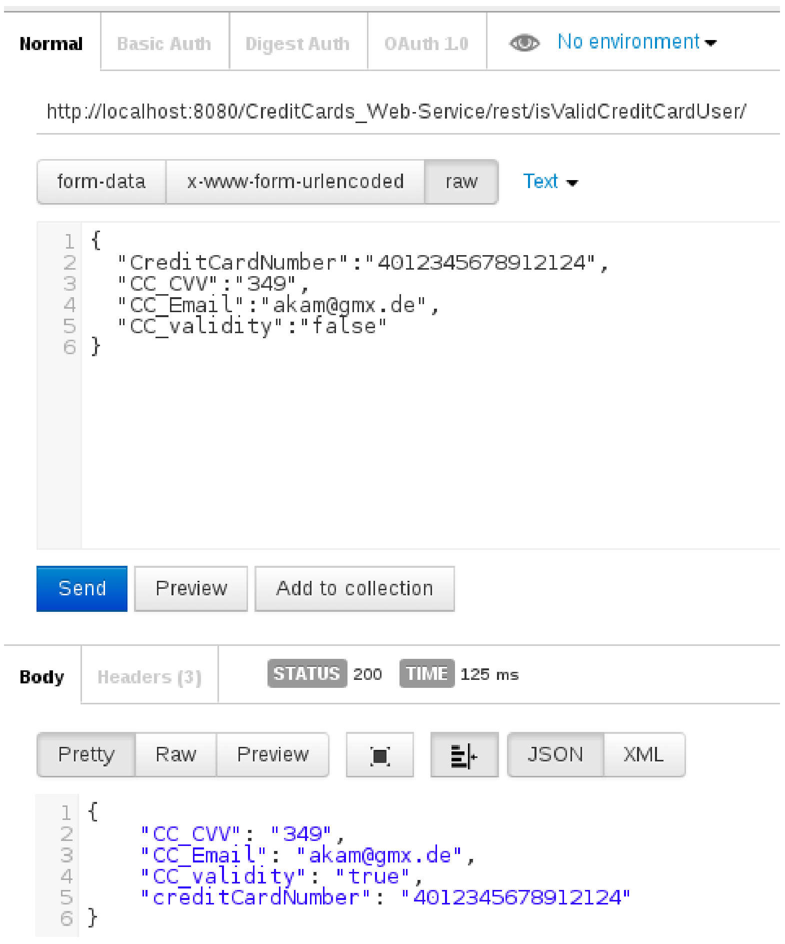




http://localhost:8080/CreditCards_Web-Service/

Welcome to REST Web Service Demo

Next we can test the RESTful Web Service with REST Client tool like POSTMAN



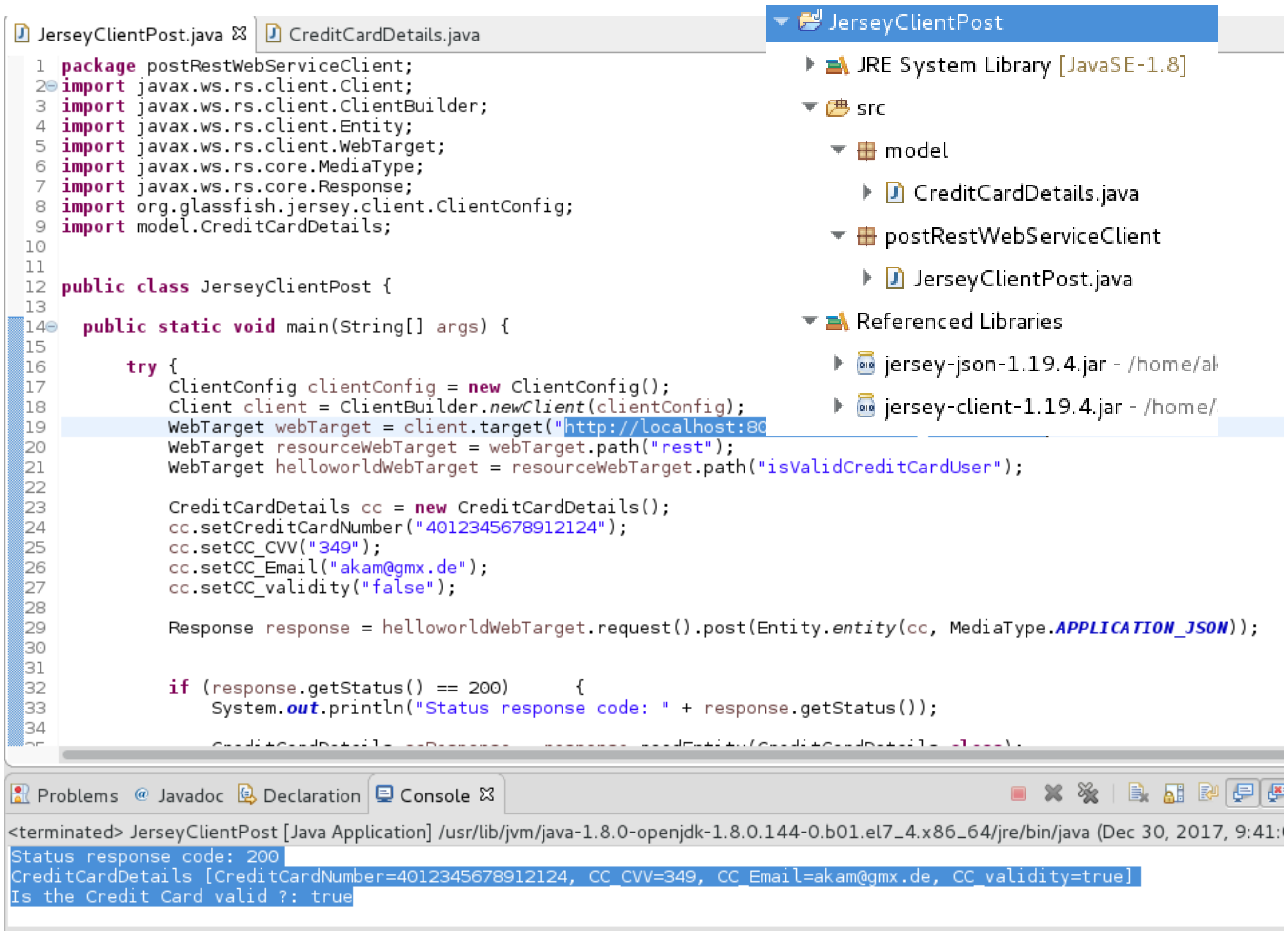
http://localhost:8080/CreditCards_Web-Service/rest/isValidCreditCardUser/

```
{
  "CreditCardNumber":"4012345678912124",
  "CC_CVV":"349",
  "CC_Email":"akam@gmx.de",
  "CC_validity":"false"
}
```

STATUS 200 TIME 125 ms

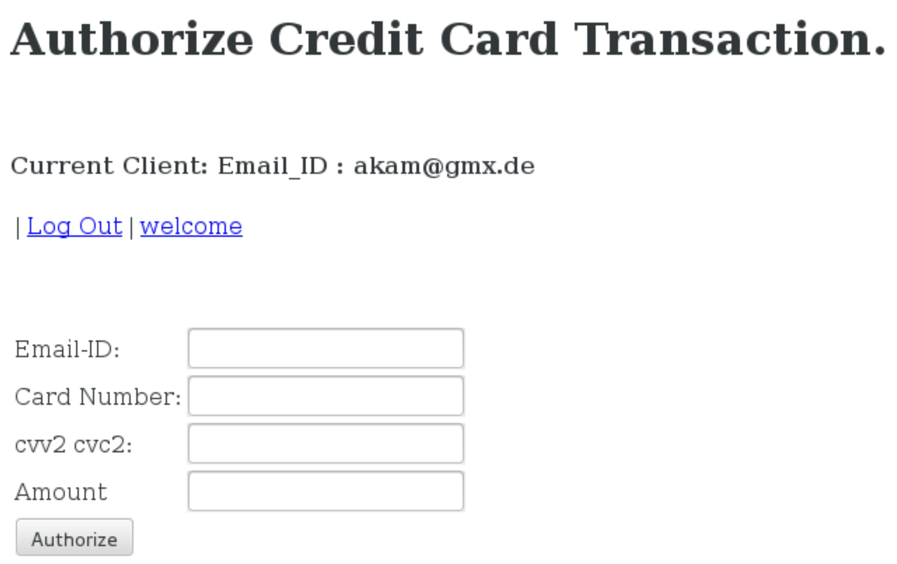
```
{
    "CC_CVV": "349",
    "CC_Email": "akam@gmx.de",
    "CC_validity": "true",
    "creditCardNumber": "4012345678912124"
}
```

We can also develop our own RESTful client application as shown here bellow. The Jersey JAX-RS jars are require. The jersey-media-moxy jar file is also required. It allows us to convert POJOs to JSONs.



```
package postRestWebServiceClient;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.glassfish.jersey.client.ClientConfig;
import model.CreditCardDetails;


public class JerseyClientPost {

  public static void main(String[] args) {

      try {
          ClientConfig clientConfig = new ClientConfig();
          Client client = ClientBuilder.newClient(clientConfig);
          WebTarget webTarget = client.target("http://localhost:80
          WebTarget resourceWebTarget = webTarget.path("rest");
          WebTarget helloworldWebTarget = resourceWebTarget.path("isValidCreditCardUser");

          CreditCardDetails cc = new CreditCardDetails();
          cc.setCreditCardNumber("4012345678912124");
          cc.setCC_CVV("349");
          cc.setCC_Email("akam@gmx.de");
          cc.setCC_validity("false");

          Response response = helloworldWebTarget.request().post(Entity.entity(cc, MediaType.APPLICATION_JSON));


          if (response.getStatus() == 200)      {
              System.out.println("Status response code: " + response.getStatus());

              CreditCardDetails ccResponse = response.readEntity(CreditCardDetails.class);
```

```
<terminated> JerseyClientPost [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.144-0.b01.el7_4.x86_64/jre/bin/java (Dec 30, 2017, 9:41:
Status response code: 200
CreditCardDetails [CreditCardNumber=4012345678912124, CC_CVV=349, CC_Email=akam@gmx.de, CC_validity=true]
Is the Credit Card valid ?: true
```

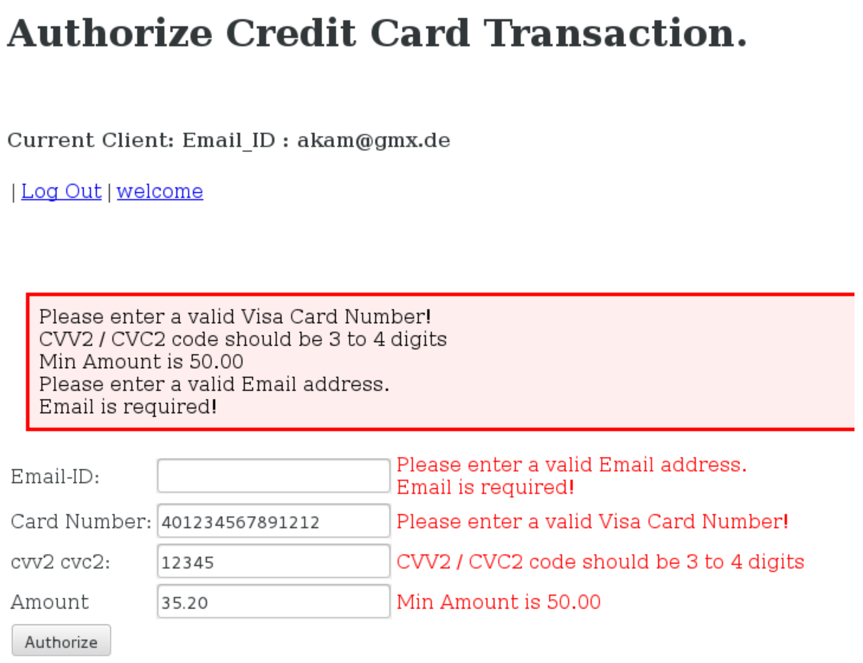


Authorize Credit Card Transaction.

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Email-ID:	
Card Number:	
cvv2 cvc2:	
Amount	



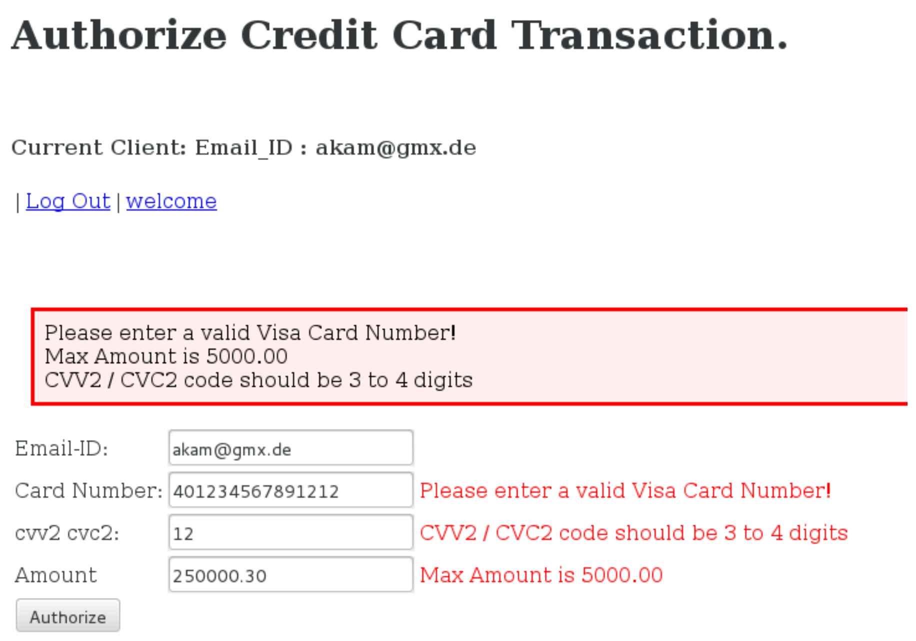
Authorize Credit Card Transaction.

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Please enter a valid Visa Card Number!
CVV2 / CVC2 code should be 3 to 4 digits
Min Amount is 50.00
Please enter a valid Email address.
Email is required!

Email-ID:		Please enter a valid Email address. Email is required!
Card Number:	401234567891212	Please enter a valid Visa Card Number!
cvv2 cvc2:	12345	CVV2 / CVC2 code should be 3 to 4 digits
Amount	35.20	Min Amount is 50.00

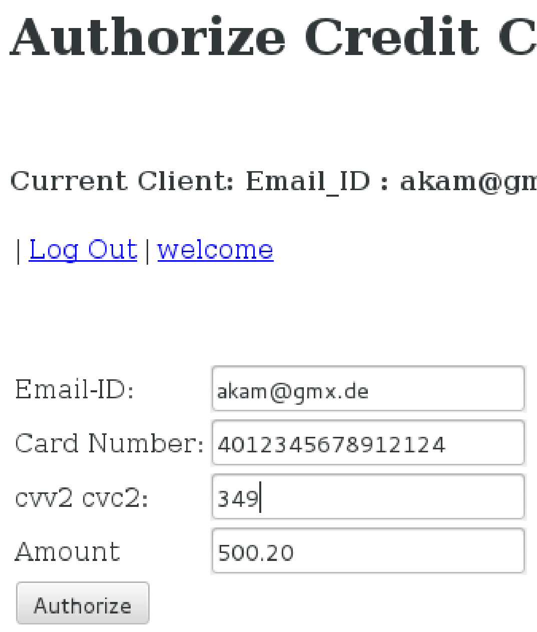

Authorize Credit Card Transaction.

Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Please enter a valid Visa Card Number!
Max Amount is 5000.00
CVV2 / CVC2 code should be 3 to 4 digits

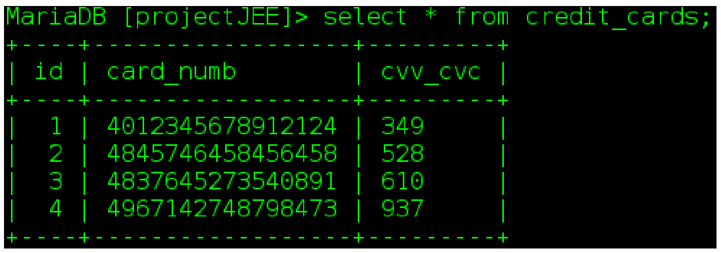
Email-ID:	akam@gmx.de	
Card Number:	401234567891212	Please enter a valid Visa Card Number!
cvv2 cvc2:	12	CVV2 / CVC2 code should be 3 to 4 digits
Amount	250000.30	Max Amount is 5000.00


Authorize Credit Card

Current Client: Email_ID : akam@gmx.de

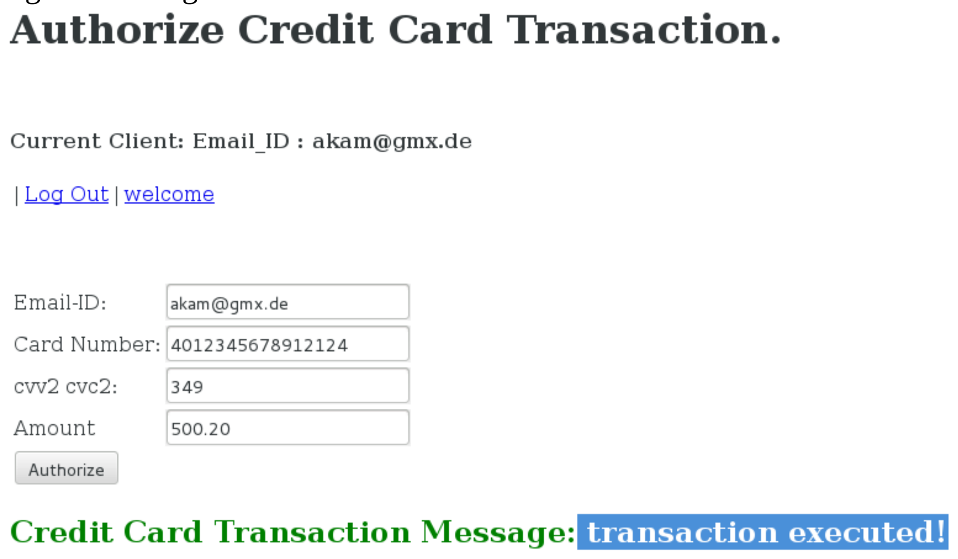
| Log Out | welcome

Email-ID:	akam@gmx.de
Card Number:	4012345678912124
cvv2 cvc2:	349
Amount	500.20


```
MariaDB [projectJEE]> select * from credit_cards;
+----+------------------+---------+
| id | card_numb        | cvv_cvc |
+----+------------------+---------+
|  1 | 4012345678912124 | 349     |
|  2 | 4845746458456458 | 528     |
|  3 | 4837645273540891 | 610     |
|  4 | 4967142748798473 | 937     |
+----+------------------+---------+
```

Above the user has entered a valid card number and the associated email account. So the user should be able to execute the transaction.

A green message will confirm to the user that the transaction has been executed as shown below.



Authorize Credit Card Transaction.

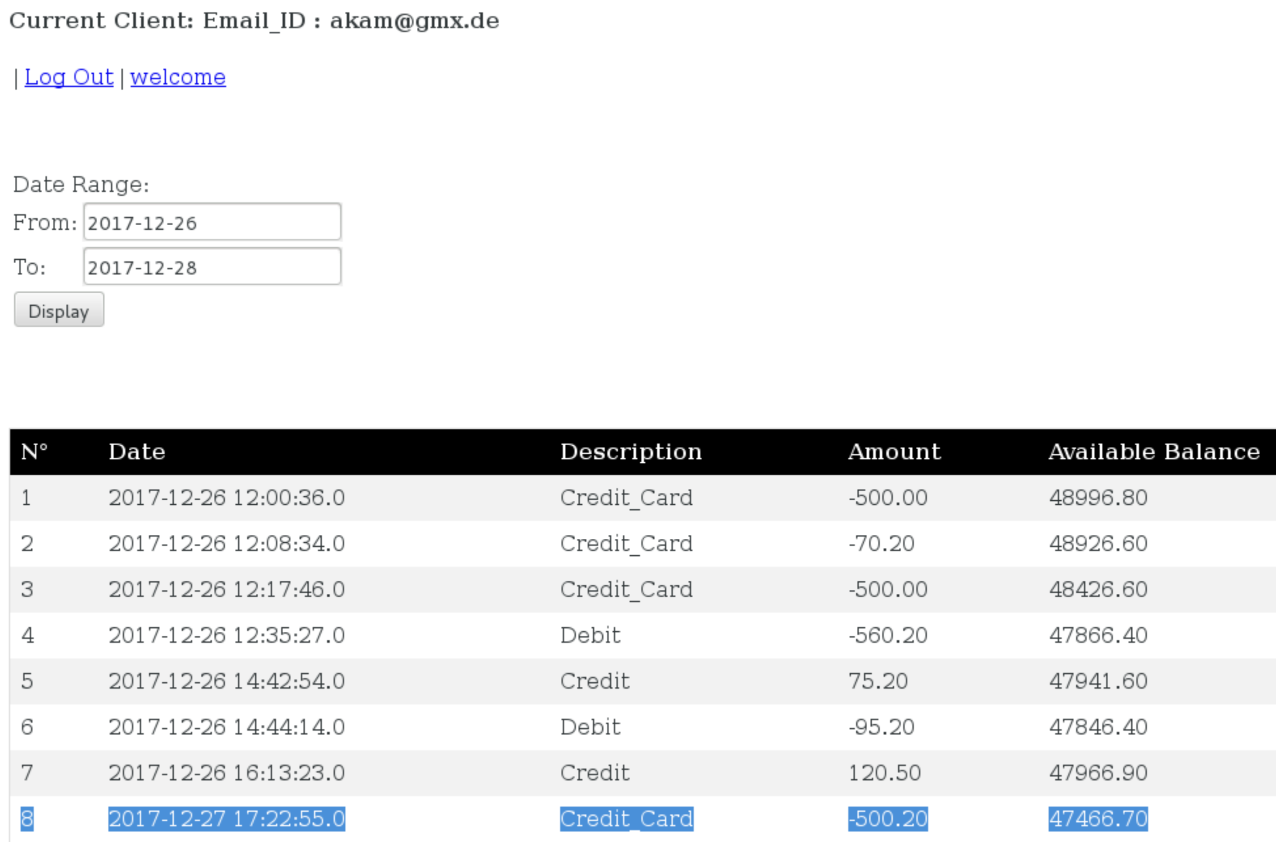
Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Email-ID:	akam@gmx.de
Card Number:	4012345678912124
cvv2 cvc2:	349
Amount	500.20

Credit Card Transaction Message: transaction executed!

Now we can list the transactions to verify that indeed the credit card transaction has been executed.



Current Client: Email_ID : akam@gmx.de

| Log Out | welcome

Date Range:

From:	2017-12-26
To:	2017-12-28

N°	Date	Description	Amount	Available Balance
1	2017-12-26 12:00:36.0	Credit_Card	-500.00	48996.80
2	2017-12-26 12:08:34.0	Credit_Card	-70.20	48926.60
3	2017-12-26 12:17:46.0	Credit_Card	-500.00	48426.60
4	2017-12-26 12:35:27.0	Debit	-560.20	47866.40
5	2017-12-26 14:42:54.0	Credit	75.20	47941.60
6	2017-12-26 14:44:14.0	Debit	-95.20	47846.40
7	2017-12-26 16:13:23.0	Credit	120.50	47966.90
8	2017-12-27 17:22:55.0	Credit_Card	-500.20	47466.70

5 Database

5.1 JDBC (Java database connectivity)

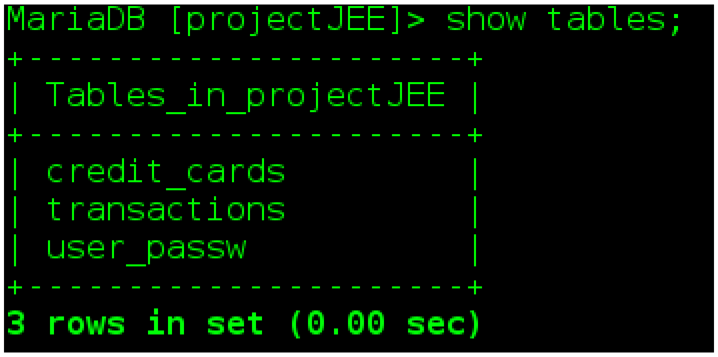
I have decided to use ***JDBC (Java database connectivity)***, simply because I need more time to master Hibernate and integrated it efficiently with Spring MVC. This is more a JSE practice rather than JEE and as so, I still need to write lot of boilerplate code.

JDBC API is the industry standard for database-independent connectivity and the driver is provided by the database vendor. JDBC offers Transaction management which crucial for a banking system (debit, credit, transfer from one account to another account). When multiple operations are performed on the database, and if one or more transaction fails then it is required to bring the database to the original state. To maintain the correct state of the data, transaction management is required. By using Commit(), changes done in the DB will be made permanent. By using RollBack() changes done in the DB will be removed.

Batch Processing
When required, to increase the performance, all queries are placed in one batch and is transferred to the Database as one batch and executed on the database.

Stored procedures could also be used.

5.2 DataBase Design, Tables



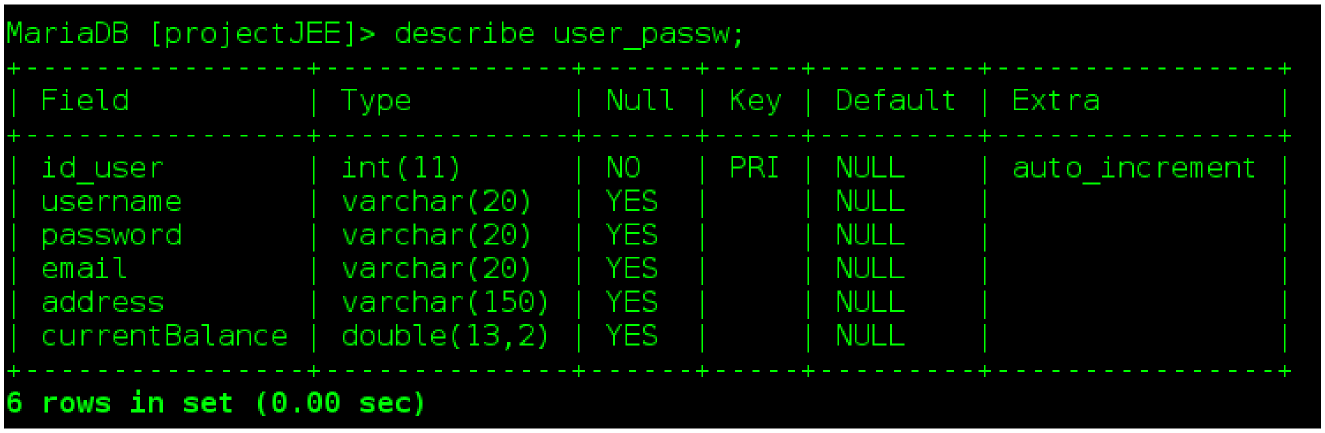
```
MariaDB [projectJEE]> show tables;
+----------------------+
| Tables_in_projectJEE |
+----------------------+
| credit_cards         |
| transactions         |
| user_passw           |
+----------------------+
3 rows in set (0.00 sec)
```

Database name: ***projectJEE***
3 tables are stored in the projectJEE data base:

- ***user_passw***
- ***transactions***
- ***credit_cards***

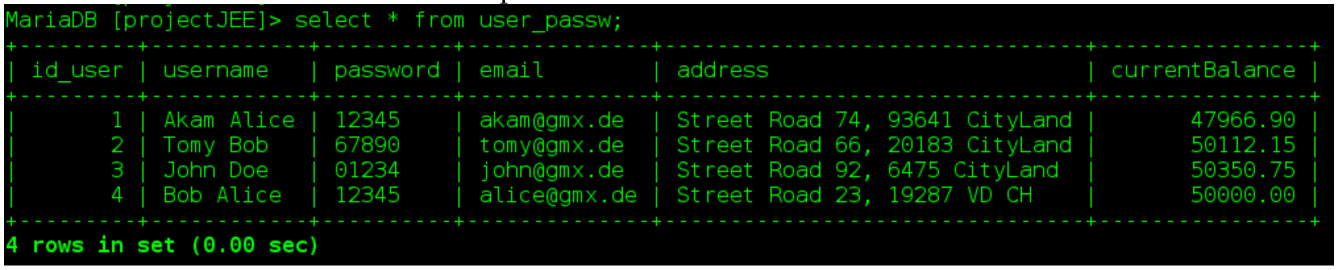
The database is designed to reduce the duplication of data and ensure referential integrity(3NF Third normal form). 3NF is designed to improve database processing while minimizing storage costs.

(*) Note that the credit_cards table in the projectJEE database was use in the first version of the program, before implementing the RESTful Web Service.



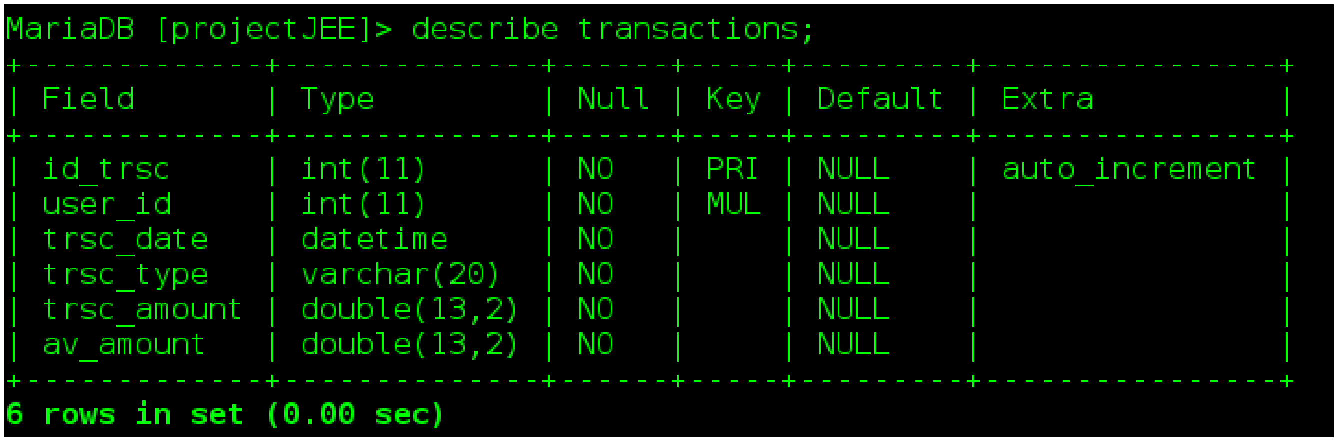
```
MariaDB [projectJEE]> describe user_passw;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id_user        | int(11)      | NO   | PRI | NULL    | auto_increment |
| username       | varchar(20)  | YES  |     | NULL    |                |
| password       | varchar(20)  | YES  |     | NULL    |                |
| email          | varchar(20)  | YES  |     | NULL    |                |
| address        | varchar(150) | YES  |     | NULL    |                |
| currentBalance | double(13,2) | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

We can list the users in the table “user pass” table as shown below.



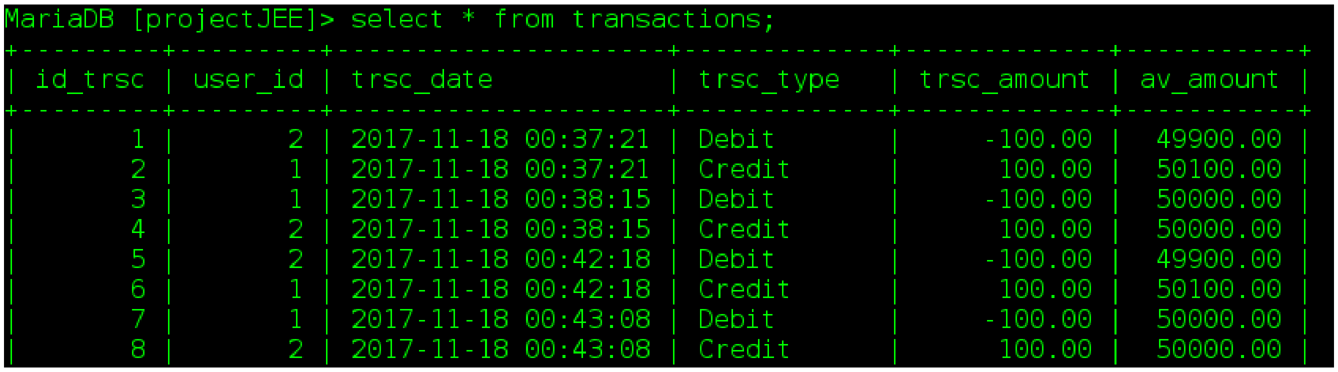
```
MariaDB [projectJEE]> select * from user_passw;
+---------+------------+----------+--------------+--------------------------------+----------------+
| id_user | username   | password | email        | address                        | currentBalance |
+---------+------------+----------+--------------+--------------------------------+----------------+
|       1 | Akam Alice | 12345    | akam@gmx.de  | Street Road 74, 93641 CityLand |       47966.90 |
|       2 | Tomy Bob   | 67890    | tomy@gmx.de  | Street Road 66, 20183 CityLand |       50112.15 |
|       3 | John Doe   | 01234    | john@gmx.de  | Street Road 92, 6475 CityLand  |       50350.75 |
|       4 | Bob Alice  | 12345    | alice@gmx.de | Street Road 23, 19287 VD CH    |       50000.00 |
+---------+------------+----------+--------------+--------------------------------+----------------+
4 rows in set (0.00 sec)
```

```
create table user_passw (
      id_user INT NOT NULL auto_increment,
      username VARCHAR(20) default NULL,
      password  VARCHAR(20) NOT NULL,
      email VARCHAR(20),
      address VARCHAR(150),
      currentBalance DOUBLE(13,2) SIGNED default 50000,
      PRIMARY KEY (id_user)
   );

INSERT INTO user_passw (username, password, email, address, currentBalance)
            VALUES ('Akam Alice' ,'12345', 'akam@gmx.de'  , 'Street Road 74, 93641 CityLand');
```



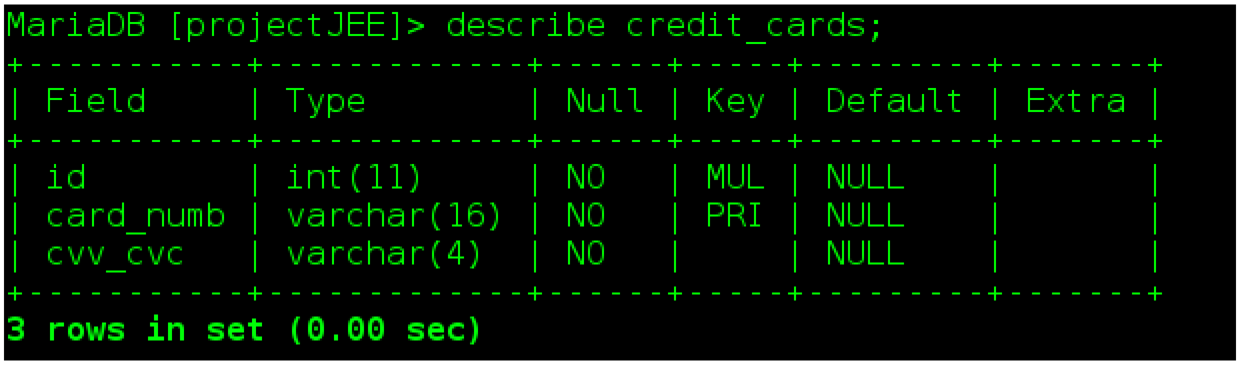
```
MariaDB [projectJEE]> describe transactions;
+--------------+---------------+------+-----+---------+----------------+
| Field        | Type          | Null | Key | Default | Extra          |
+--------------+---------------+------+-----+---------+----------------+
| id_trsc      | int(11)       | NO   | PRI | NULL    | auto_increment |
| user_id      | int(11)       | NO   | MUL | NULL    |                |
| trsc_date    | datetime      | NO   |     | NULL    |                |
| trsc_type    | varchar(20)   | NO   |     | NULL    |                |
| trsc_amount  | double(13,2)  | NO   |     | NULL    |                |
| av_amount    | double(13,2)  | NO   |     | NULL    |                |
+--------------+---------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Here below are listed the transactions extracted from the “transactions” table.



```
MariaDB [projectJEE]> select * from transactions;
+---------+---------+---------------------+-------------+-------------+-----------+
| id_trsc | user_id | trsc_date           | trsc_type   | trsc_amount | av_amount |
+---------+---------+---------------------+-------------+-------------+-----------+
|       1 |       2 | 2017-11-18 00:37:21 | Debit       |     -100.00 |  49900.00 |
|       2 |       1 | 2017-11-18 00:37:21 | Credit      |      100.00 |  50100.00 |
|       3 |       1 | 2017-11-18 00:38:15 | Debit       |     -100.00 |  50000.00 |
|       4 |       2 | 2017-11-18 00:38:15 | Credit      |      100.00 |  50000.00 |
|       5 |       2 | 2017-11-18 00:42:18 | Debit       |     -100.00 |  49900.00 |
|       6 |       1 | 2017-11-18 00:42:18 | Credit      |      100.00 |  50100.00 |
|       7 |       1 | 2017-11-18 00:43:08 | Debit       |     -100.00 |  50000.00 |
|       8 |       2 | 2017-11-18 00:43:08 | Credit      |      100.00 |  50000.00 |
```

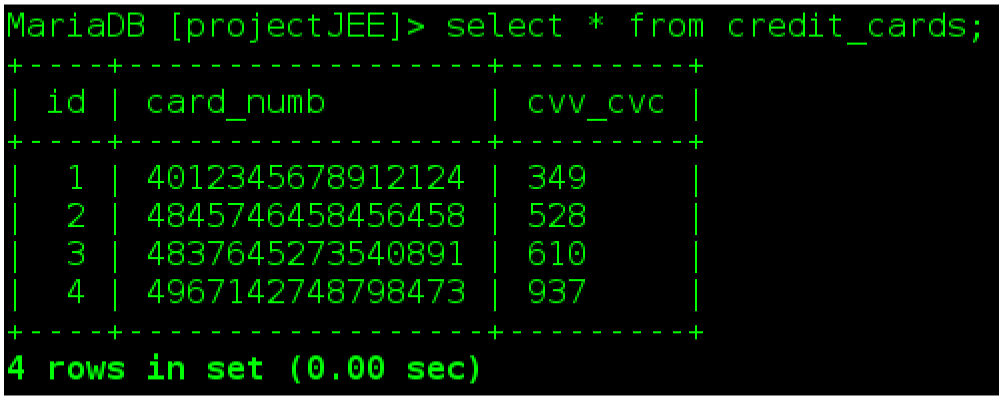
```sql
create table transactions (
      id_trsc INT NOT NULL auto_increment,
      user_id INT NOT NULL,
      trsc_date  DATETIME NOT NULL,
      trsc_type  VARCHAR(20) NOT NULL,
      trsc_amount  DOUBLE(13,2) SIGNED NOT NULL,
      av_amount  DOUBLE(13,2) SIGNED NOT NULL,

      PRIMARY KEY(id_trsc),
      FOREIGN KEY (user_id)
      REFERENCES USER(id_user)
      ON UPDATE CASCADE ON DELETE CASCADE
 );
```



```
MariaDB [projectJEE]> describe credit_cards;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| id        | int(11)      | NO   | MUL | NULL    |       |
| card_numb | varchar(16)  | NO   | PRI | NULL    |       |
| cvv_cvc   | varchar(4)   | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

In the table *credit_cards above*, the *card number* field is unique (visa pattern) and is used as primary key. The *id* field is a foreign key referencing the *id_user* in the table u*ser_passw* table. (onupdate delete cascade). Therefore a user could have many Credit Cards.

As shown below, the table credit_cards contains the credit cards details associated with each user.

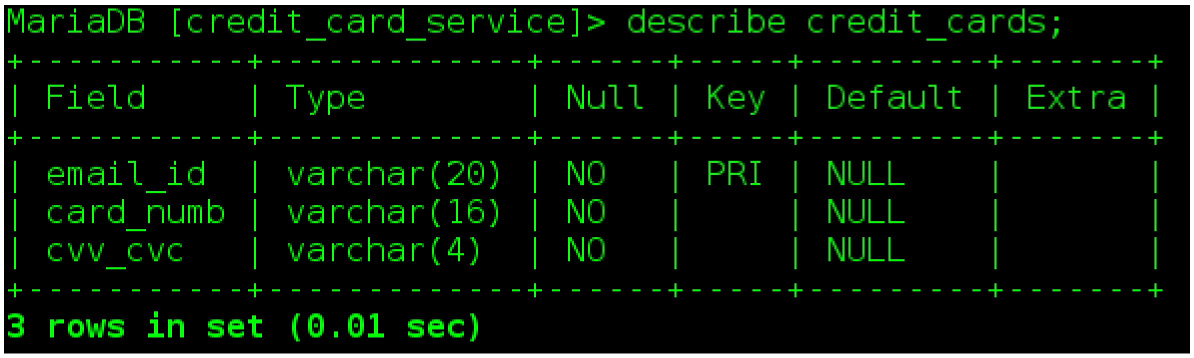


```
MariaDB [projectJEE]> select * from credit_cards;
+----+------------------+---------+
| id | card_numb        | cvv_cvc |
+----+------------------+---------+
|  1 | 4012345678912124 | 349     |
|  2 | 4845746458456458 | 528     |
|  3 | 4837645273540891 | 610     |
|  4 | 4967142748798473 | 937     |
+----+------------------+---------+
4 rows in set (0.00 sec)
```

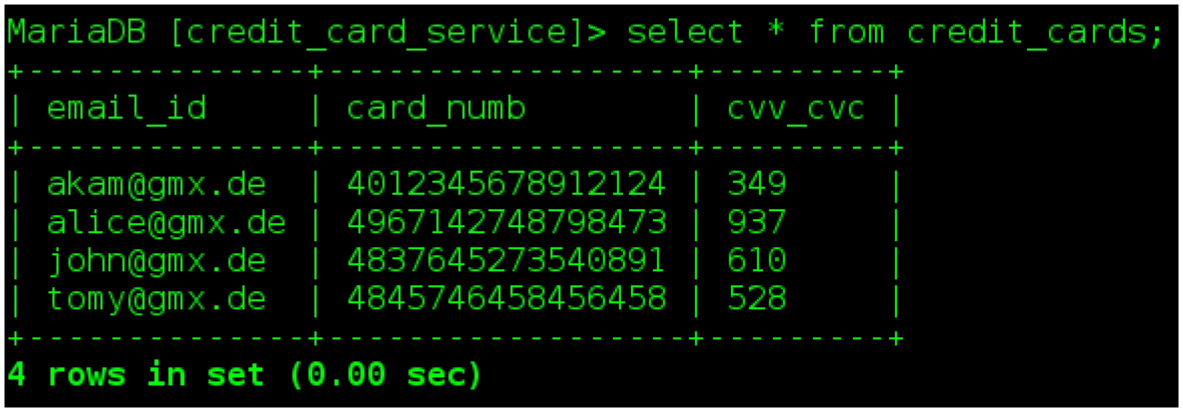
```
create table credit_cards (
      id INT NOT NULL,
      card_numb VARCHAR(16) NOT NULL,
      cvv_cvc VARCHAR(4) NOT NULL,

      PRIMARY KEY(card_numb),
      FOREIGN KEY (id)
      REFERENCES user_passw(id_user)
      ON UPDATE CASCADE ON DELETE CASCADE
 );

insert into credit_cards values
('1', '4012345678912124', '349' ),
('2', '4845746458456458', '528' ),
('3', '4837645273540891', '610' ),
('4', '4967142748798473', '937' ) ;
```

Database name: ***credit_card_service***
1 table is stored in this data base: ***credit_cards***
This database is associated with the credit cards *RESTful Web Service*.

```
create database credit_card_service;

use credit_card_service;

create table credit_cards (
      email_id VARCHAR(20) NOT NULL,
      card_numb VARCHAR(16) NOT NULL,
      cvv_cvc VARCHAR(4) NOT NULL,

      PRIMARY KEY(email_id)
 );

insert into credit_cards values
('akam@gmx.de', '4012345678912124', '349' ),
('tomy@gmx.de', '4845746458456458', '528' ),
('john@gmx.de', '4837645273540891', '610' ),
('alice@gmx.de', '4967142748798473', '937' ) ;
```



```
MariaDB [credit_card_service]> describe credit_cards;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| email_id  | varchar(20)  | NO   | PRI | NULL    |       |
| card_numb | varchar(16)  | NO   |     | NULL    |       |
| cvv_cvc   | varchar(4)   | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```



```
MariaDB [credit_card_service]> select * from credit_cards;
+--------------+------------------+---------+
| email_id     | card_numb        | cvv_cvc |
+--------------+------------------+---------+
| akam@gmx.de  | 4012345678912124 | 349     |
| alice@gmx.de | 4967142748798473 | 937     |
| john@gmx.de  | 4837645273540891 | 610     |
| tomy@gmx.de  | 4845746458456458 | 528     |
+--------------+------------------+---------+
4 rows in set (0.00 sec)
```

6 Bibliography

Java Enterprise Edition
http://www.oracle.com/technetwork/java/javaee/overview/index.html

Apache Web Server
http://tomcat.apache.org/

Maven Repository:
https://maven.apache.org/
https://mvnrepository.com/

MVC Spring Framework:
https://spring.io/

DataBase:
https://mariadb.org/
https://dev.mysql.com/

RESTful Web Service
https://jersey.github.io/

JSP Java Server Pages Standard Tag Library JSTL
http://www.oracle.com/technetwork/java/index-jsp-135995.html