CUSTOMER APPLICATION
(CRUD operations)

Prepared by

Tumula Mani Kota Rajasekhar (51830158)

Table Of Contents

1. Introduction

The “CUSTOMER APPLICATION” is mainly based on CRUD operations in which the customer’s details are added, deleted, read and updated by a valid employee of the company. In order to view or make any changes in the customer database, the employee should prove his authenticity by completing a login formality, in which he/she is asked to enter his email and password. The employees are segregated into three categories like “ADMIN”, “MANAGER” and “USER”. Each category has some specific powers like admin can perform all the operations on customer database, where the manager can only add a new customer to the database and the user can only view the information about the customers in the database. These types of applications are mainly used in shopping malls and super markets to monitor their growth and customer interaction.

Analysis

- Customer Management System aims to manage the details of the customer. It consists of mainly four functionalities:
- Adding Customer Details:
  Adding the customer details by administrator and manager. Details includes Customer name, address, mobile, email, visited date, visit type.
- Updating the Customer Details:
  Administrator can update the details of customer.
- Deleting the Customer Details:
  Administrator can delete the customer record.
- Viewing the Customer Details:
  Administrator, Manager and employee can view the customer details.

1.1 CRUD Operations

CRUD stands for **Create**, **Read**, **Update** and **Delete**. These functions are the user interfaces to databases, as they permit users to create, view, modify and alter data. CRUD works on entities in databases and manipulates these entities.

For instance, a simple customer database table adds (creates) new customer details, accesses (reads) existing customer details, modifies (updates) existing customer data, and deletes customer details when there is no need.

The commands corresponding to these operations in SQL are INSERT, SELECT, UPDATE and DELETE. **INSERT** adds new records, **SELECT** retrieves or selects existing records based on selection conditions, **UPDATE** modifies existing records and **DELETE** removes tables or records in a table.

CRUD means the basic **operations** to be done in a data base. It directly handles records or data objects; apart from these operations, the records are passive entities.

We will have following components in our database application:

CustomerController.java – Manages the CRUD operation which are performed on customer table

EmpController.java – This class used to verify the authentication of the employee who is going to access the Customer data

EmpFormBean.java – This class is used to validate the required fields in order to perform the login operation

addcustomer.jsp – Page used to add, update and delete a new customer from/to the database table “cust_data1”

all_customer.jsp – Page used to display all the customer details present in the database

login.jsp – Page used to display the login screen where employee can login to view the customer details

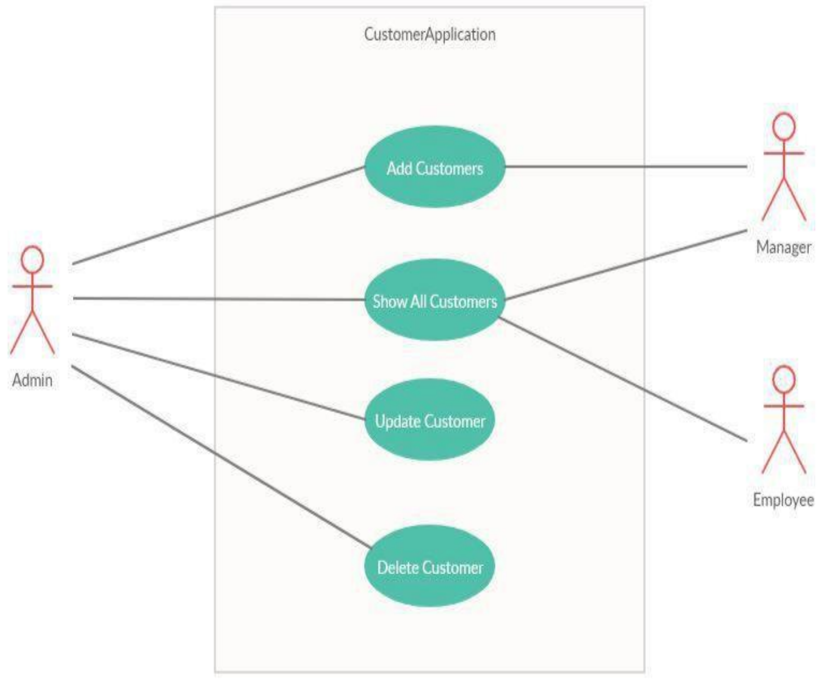
This application lets you manage customer’s database such as add new customer and update/delete customer information. But after making sure that the one who is doing those operations is a valid employee. This can be done by checking his email id and password which are stored in another database table named “emp_table_cont”

1.2 CRUD Benefits

Using the database operations in your applications has some advantages i.e.

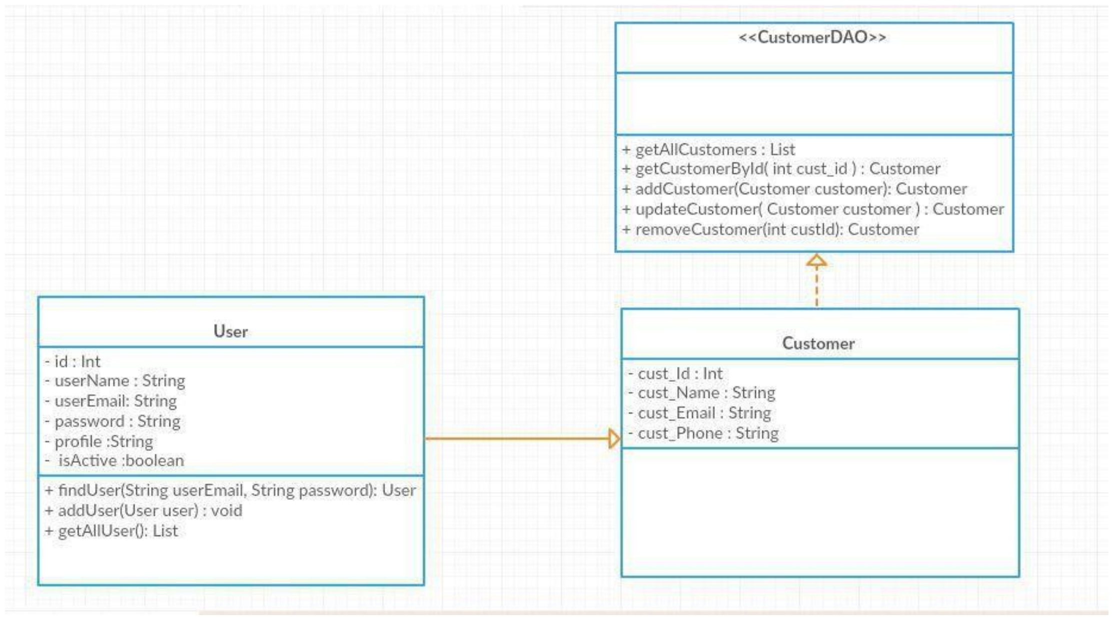
- Improves data security and data access to users by using host and query languages
- Greater data integrity and independence of applications programs
- Improves application performance by reducing the data redundancy

Use Case Diagram:



User Case Diagram

Class Diagram:



Class Diagram

2. Tools and project structure

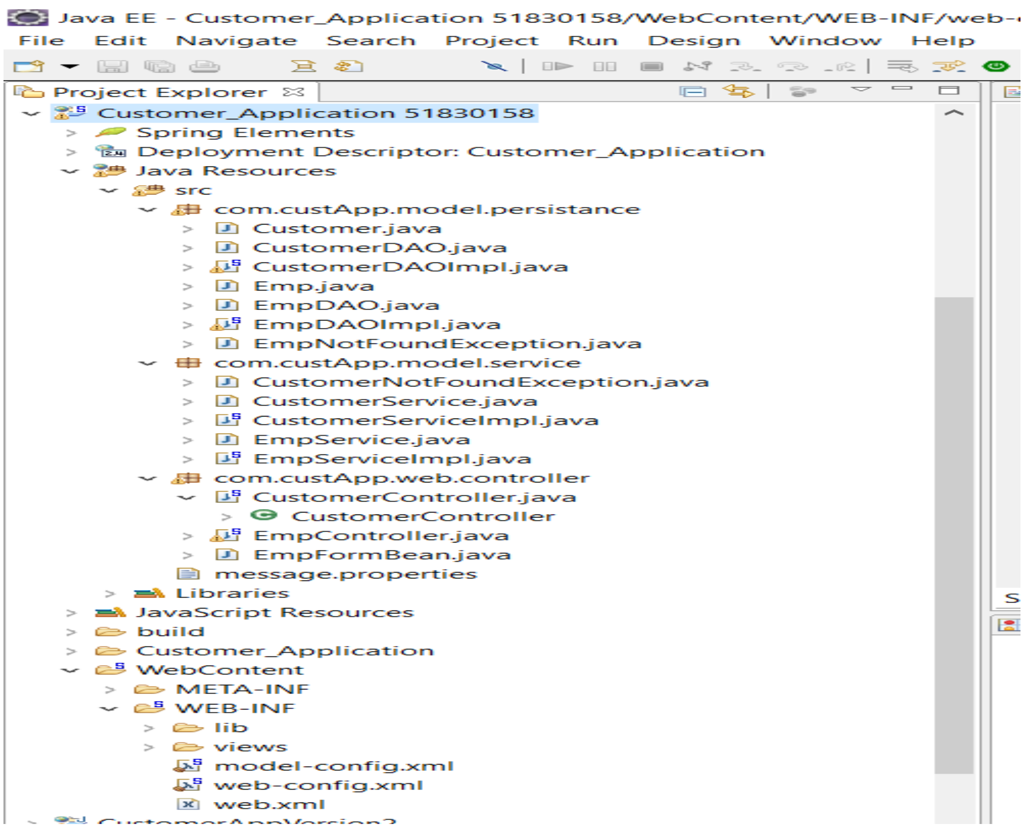
2.1 Tools Used

1. **ECLIPSE**
2. **SPRING 3.8**
3. **MY SQL 5.1.48**
4. **TOMCAT 7.0.57**

2.2 Project Structure

The project structure mainly consists of three layers namely:

1. Web Layer
2. Model Layer
    2.1. Service Layer
    2.2. Persistence Layer


Java EE - Customer_Application 51830158/WebContent/WEB-INF/web-
File Edit Navigate Search Project Run Design Window Help
Project Explorer

```
Customer_Application 51830158
  > Spring Elements
  > Deployment Descriptor: Customer_Application
  Java Resources
    src
      com.custApp.model.persistance
        > Customer.java
        > CustomerDAO.java
        > CustomerDAOImpl.java
        > Emp.java
        > EmpDAO.java
        > EmpDAOImpl.java
        > EmpNotFoundException.java
      com.custApp.model.service
        > CustomerNotFoundException.java
        > CustomerService.java
        > CustomerServiceImpl.java
        > EmpService.java
        > EmpServiceImpl.java
      com.custApp.web.controller
        CustomerController.java
          > CustomerController
        > EmpController.java
        > EmpFormBean.java
      message.properties
    > Libraries
  > JavaScript Resources
  > build
  > Customer_Application
  WebContent
    > META-INF
    WEB-INF
      > lib
      > views
      model-config.xml
      web-config.xml
      web.xml
```

Figure 1: project structure

MODEL LAYER:

It internally consists of two layers namely service and persistence layer.

Persistence Layer:

Here persistence layer mainly deals with the creation of skeleton in database. It consists of DAO interfaces and DAO Implementations which in turn creates the basic structure of the application. In this case Customer class consists of customer details which are to be inserted in the database table. Similarly, in the case of Employee class. The DAO and DAO Implementation layers of Customer and Emp are used to define methods like getEmp() and getCustomer().

Service Layer:

Service layer mainly deals with the business logic in which the service and performance of the DAO layers are written. In the case of this application, interfaces like CustomerService() and EmpService() are created. The implementations of those layers define the service of the Customers and Employees.

WEB LAYER:

It is the main layer which acts as controller layer, which interacts with the HTML/JSP pages. Here in the case of Customer Application the web layer consist of CustomerController and EmpController which is used to interact with the jsp pages and get the info from the user.

2.3 Project Creation:

The project creation starts from choosing Dynamic Web Project from File->New->DynamicWebProject

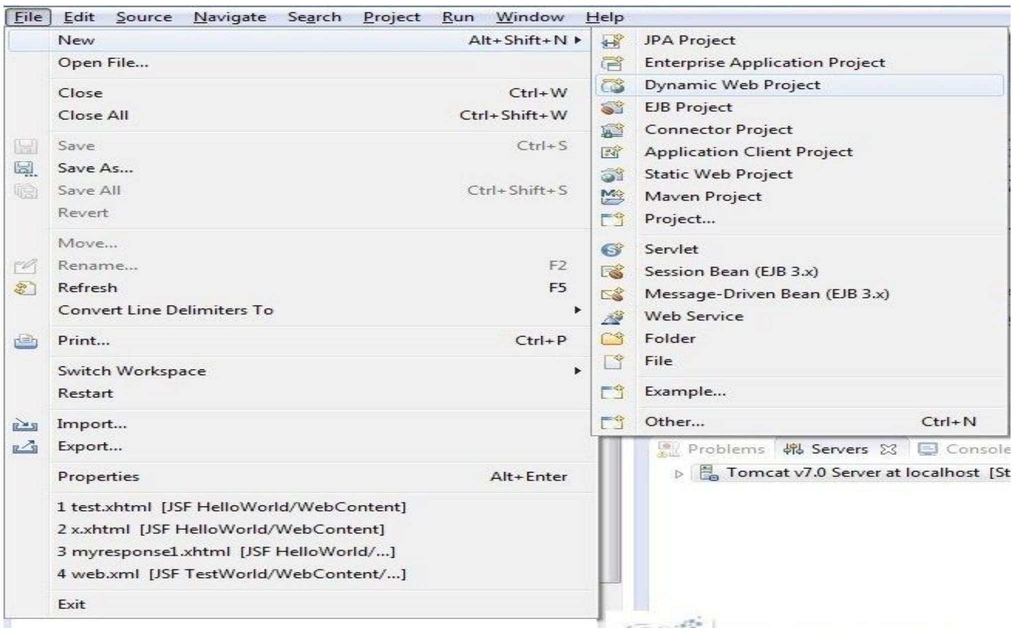
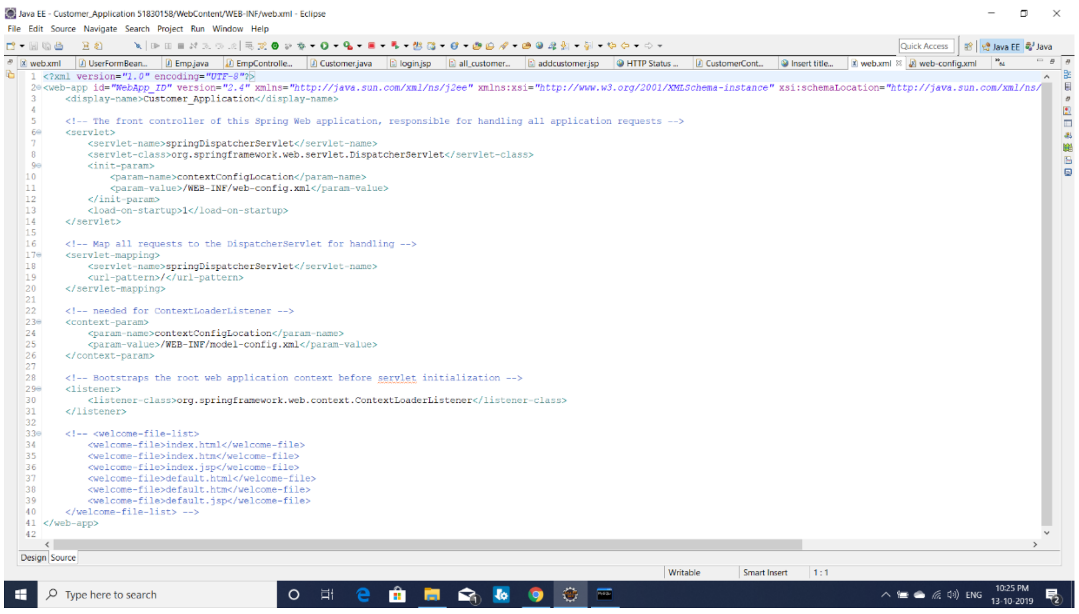


Figure 2: project creation step 1

After the creation of the dynamic web project, the xml file "web.xml" was configured such that the **Spring Dispatcher Servlet** had the xml file "web-config.xml" which further takes care about the web layer and **Context Loader Listener** had the xml file "model-config.xml" which further takes care about the model layer.

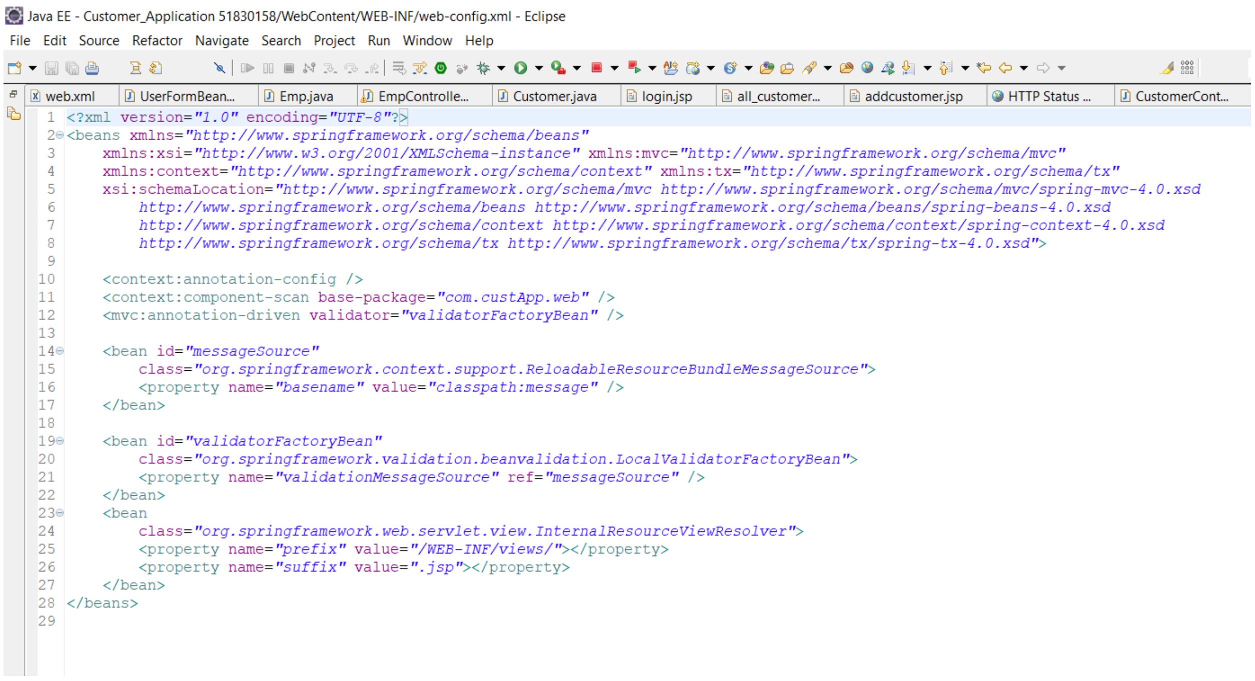


```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/
    <display-name>Customer_Application</display-name>

    <!-- The front controller of this Spring Web application, responsible for handling all application requests -->
    <servlet>
        <servlet-name>springDispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/web-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Map all requests to the DispatcherServlet for handling -->
    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- needed for ContextLoaderListener -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/model-config.xml</param-value>
    </context-param>

    <!-- Bootstraps the root web application context before servlet initialization -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!-- <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list> -->
</web-app>
```

Figure 3: project creation step 2 (configuring xml)

web-config.xml:

It is the layer which interacts with web layer and also the HTML/JSP pages by using
`<context:component-scan base-package="com.custApp.web" />`


Java EE - Customer_Application 51830158/WebContent/WEB-INF/web-config.xml - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
web.xml
UserFormBean...
Emp.java
EmpControlle...
Customer.java
login.jsp
all_customer...
addcustomer.jsp
HTTP Status ...
CustomerCont...

```xml
<?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:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <context:annotation-config />
    <context:component-scan base-package="com.custApp.web" />
    <mvc:annotation-driven validator="validatorFactoryBean" />

    <bean id="messageSource"
        class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <property name="basename" value="classpath:message" />
    </bean>

    <bean id="validatorFactoryBean"
        class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
        <property name="validationMessageSource" ref="messageSource" />
    </bean>
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>

```

Figure 4: project creation step 3 (web-config.xml)

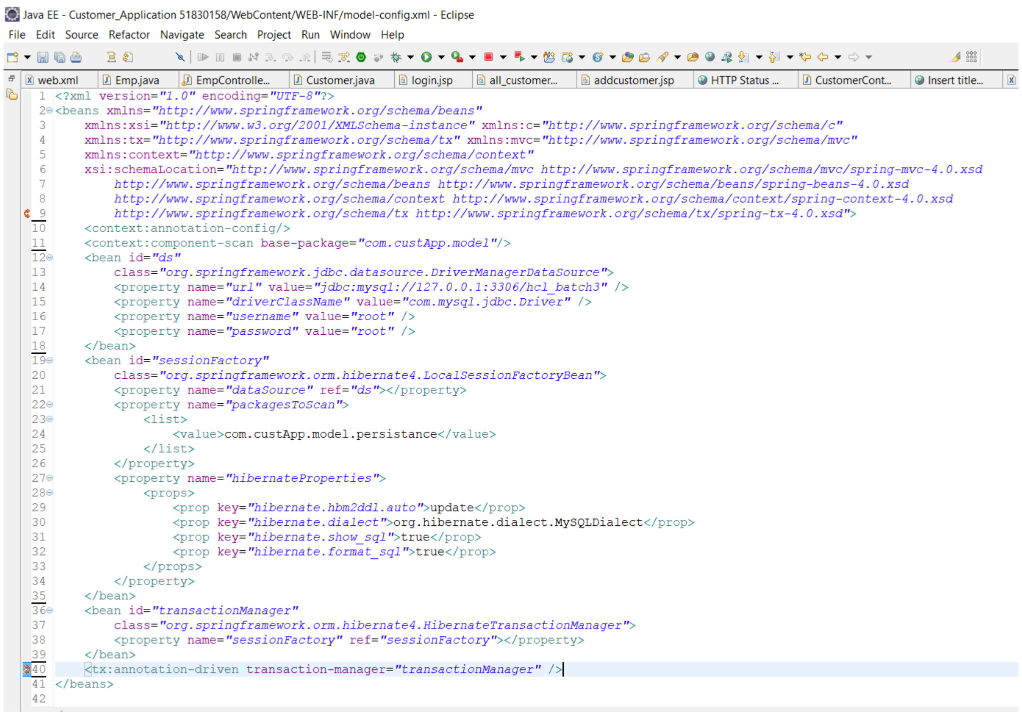
model-config.xml:

This is the layer which interacts with model layer by using
`<context:component-scan base-package="com.custApp.model"/>`
This is also the layer which is used to interact with database using jdbc

```xml
class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="url" value="jdbc:mysql://127.0.0.1:3306/hcl_batch3" />
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="username" value="root" />
<property name="password" value="root" />
```

This is also the layer which is responsible for configuring hibernate by using the following lines

```xml
<property name="hibernateProperties">
      <props>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
      </props>
</property>
```


Java EE - Customer_Application 51830158/WebContent/WEB-INF/model-config.xml - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
web.xml
Emp.java
EmpControlle...
Customer.java
login.jsp
all_customer...
addcustomer.jsp
HTTP Status ...
CustomerCont...
Insert title...

```xml
<?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:c="http://www.springframework.org/schema/c"
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
    <context:annotation-config/>
    <context:component-scan base-package="com.custApp.model"/>
    <bean id="ds"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="url" value="jdbc:mysql://127.0.0.1:3306/hcl_batch3" />
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="ds"></property>
        <property name="packagesToScan">
            <list>
                <value>com.custApp.model.persistance</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
            </props>
        </property>
    </bean>
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"></property>
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager" />
</beans>
```

Figure 5: project creation step 4 (model-config.xml)

3. Application Building

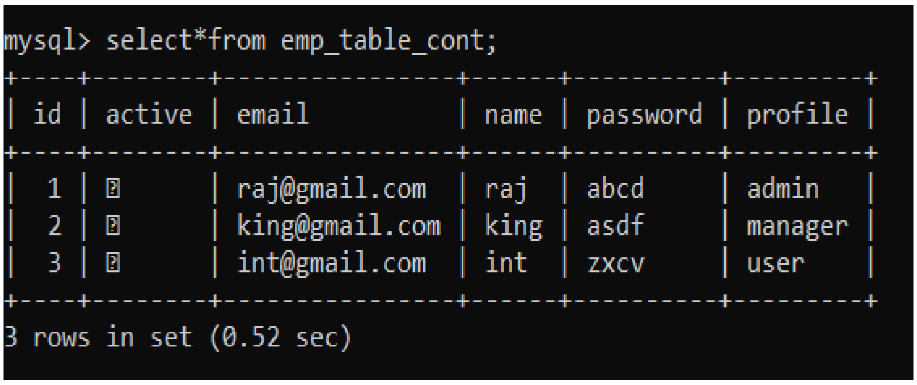
3.1. Database & Table Creation

Here in this case we are using MY SQL as database and inorder to create a table in database we are using hibernate quieries. Before that we are going to configure the hibernate properties in “model-config.xml” as

```xml
<property name="hibernateProperties">

        <props>

                <prop key="hibernate.hbm2ddl.auto">update</prop>

                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

                <prop key="hibernate.show_sql">true</prop>

                <prop key="hibernate.format_sql">true</prop>

        </props>

</property>
```

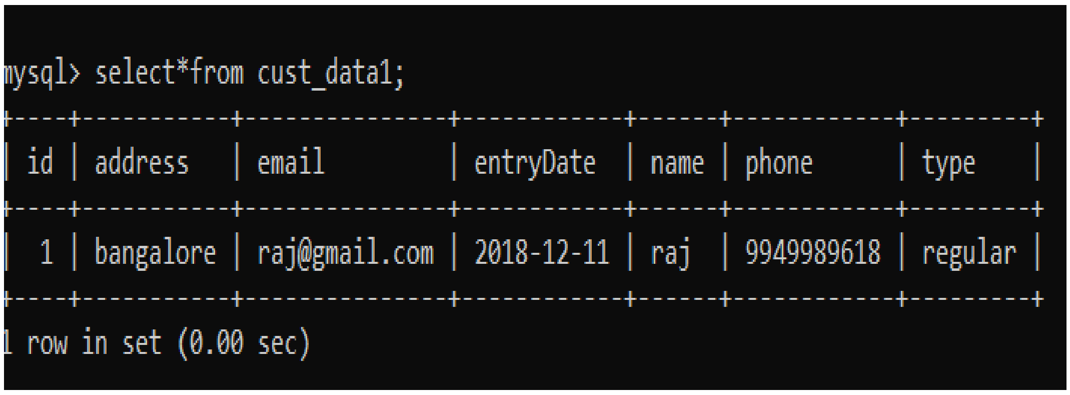
After configuring the hibernate properties we can access the database by using HQL quiries instead of using SQL quiries.

Two database tables are created named with `cust_data1` which contains the customer info and `emp_table_cont` which contails the employee info who can operate the customer table.



```
mysql> select*from emp_table_cont;
+----+--------+----------------+------+----------+---------+
| id | active | email          | name | password | profile |
+----+--------+----------------+------+----------+---------+
|  1 | ⍰      | raj@gmail.com  | raj  | abcd     | admin   |
|  2 | ⍰      | king@gmail.com | king | asdf     | manager |
|  3 | ⍰      | int@gmail.com  | int  | zxcv     | user    |
+----+--------+----------------+------+----------+---------+
3 rows in set (0.52 sec)
```

Figure 6: employee table



```
mysql> select*from cust_data1;
+----+-----------+---------------+------------+------+------------+---------+
| id | address   | email         | entryDate  | name | phone      | type    |
+----+-----------+---------------+------------+------+------------+---------+
|  1 | bangalore | raj@gmail.com | 2018-12-11 | raj  | 9949989618 | regular |
+----+-----------+---------------+------------+------+------------+---------+
1 row in set (0.00 sec)
```

Figure 7: customer table

3.2. File Creation

3.2.1. Implementation of HTML/JSP pages:

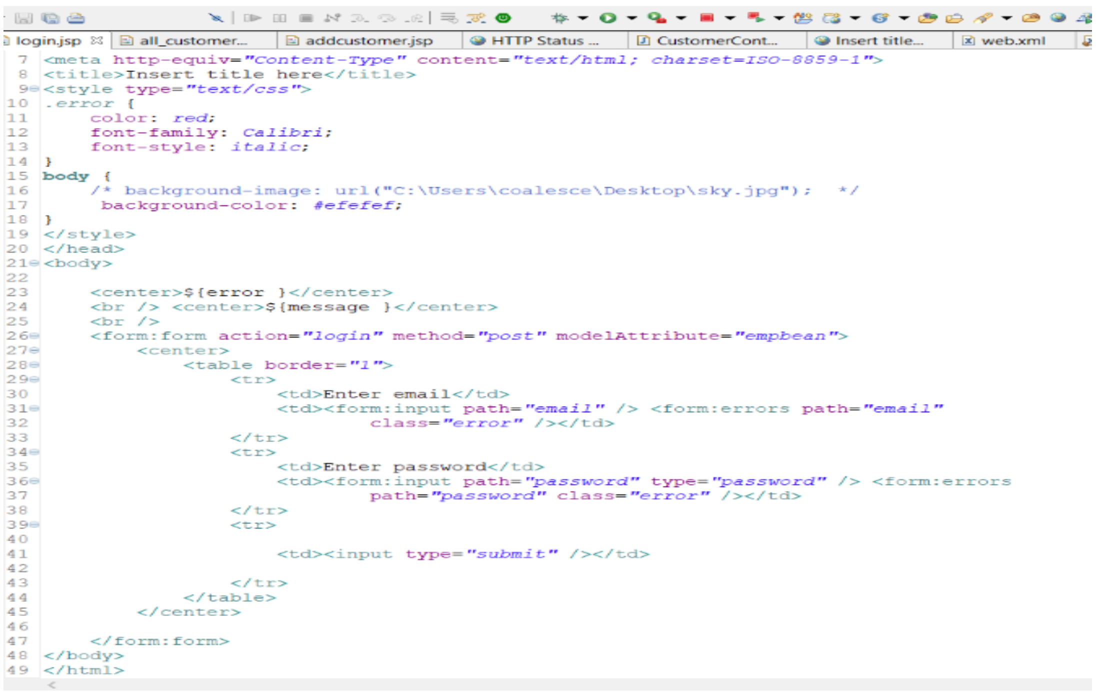
Totally in this project three JSP pages are created to enhance the User Interface such that it will be easy for the user to interact.

The home page which will welcome the user is “login.jsp” where the employee ahd to provide his details like email id and password in order to prove his authentication. The email and password field are validated using annotations like @Email (for the validation of email id) and @NotEmpty in the case of password. Once the details are entered and submitted by the employee they were verified and re-directs the employee to the Customer table where he can make the CRUD operation like add new customer/delete existing customer. Here there is also another validation followed in order to limit the operations performed. The profiles of the employees were divided into three categories like “admin”, “manager” and “user”.

The admin can make all the crud operations where manager can only add the customer and the user had access to view the information in the customer table. This validation can be performed by using the EmpController.java.

login.jsp:

“login.jsp” will be the home page where it asks the credentials of the employees in order to log them into the application.



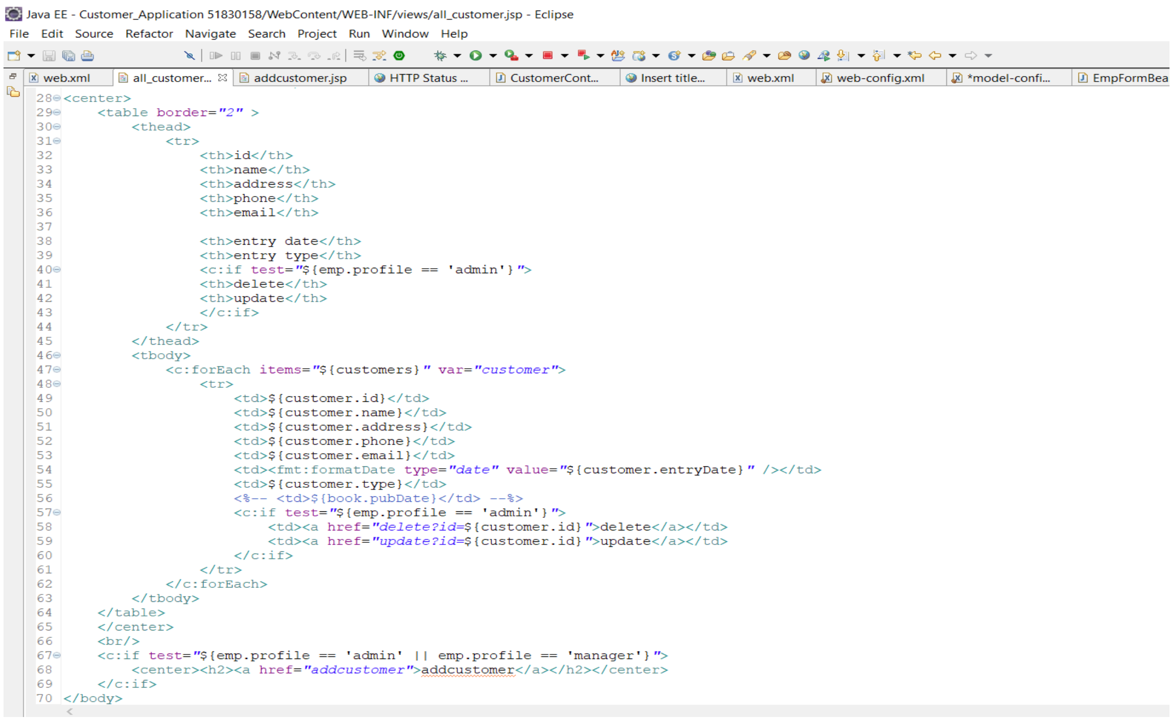
```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
<style type="text/css">
.error {
    color: red;
    font-family: Calibri;
    font-style: italic;
}
body {
    /* background-image: url("C:\Users\coalesce\Desktop\sky.jpg");  */
     background-color: #efefef;
}
</style>
</head>
<body>

    <center>${error }</center>
    <br /> <center>${message }</center>
    <br />
    <form:form action="login" method="post" modelAttribute="empbean">
        <center>
            <table border="1">
                <tr>
                    <td>Enter email</td>
                    <td><form:input path="email" /> <form:errors path="email"
                            class="error" /></td>
                </tr>
                <tr>
                    <td>Enter password</td>
                    <td><form:input path="password" type="password" /> <form:errors
                            path="password" class="error" /></td>
                </tr>
                <tr>

                    <td><input type="submit" /></td>

                </tr>
            </table>
        </center>

    </form:form>
</body>
</html>
```

Figure 8: home page (login.jsp)

all_customers.jsp:

The successful login into the application takes the employee to the all_customers.jsp page where the total number of customers with their details like name, phone number and email are there. In that page according to the profile of the employee the options like “delete”, “update” and “add customer” are shown where they can make the crud operations.

Css styles can be applied such that the customer table is nicely alligned and also looks in an attactive way.


Java EE - Customer_Application 51830158/WebContent/WEB-INF/views/all_customer.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
web.xml
all_customer...
addcustomer.jsp
HTTP Status ...
CustomerCont...
Insert title...
web.xml
web-config.xml
*model-confi...
EmpFormBea

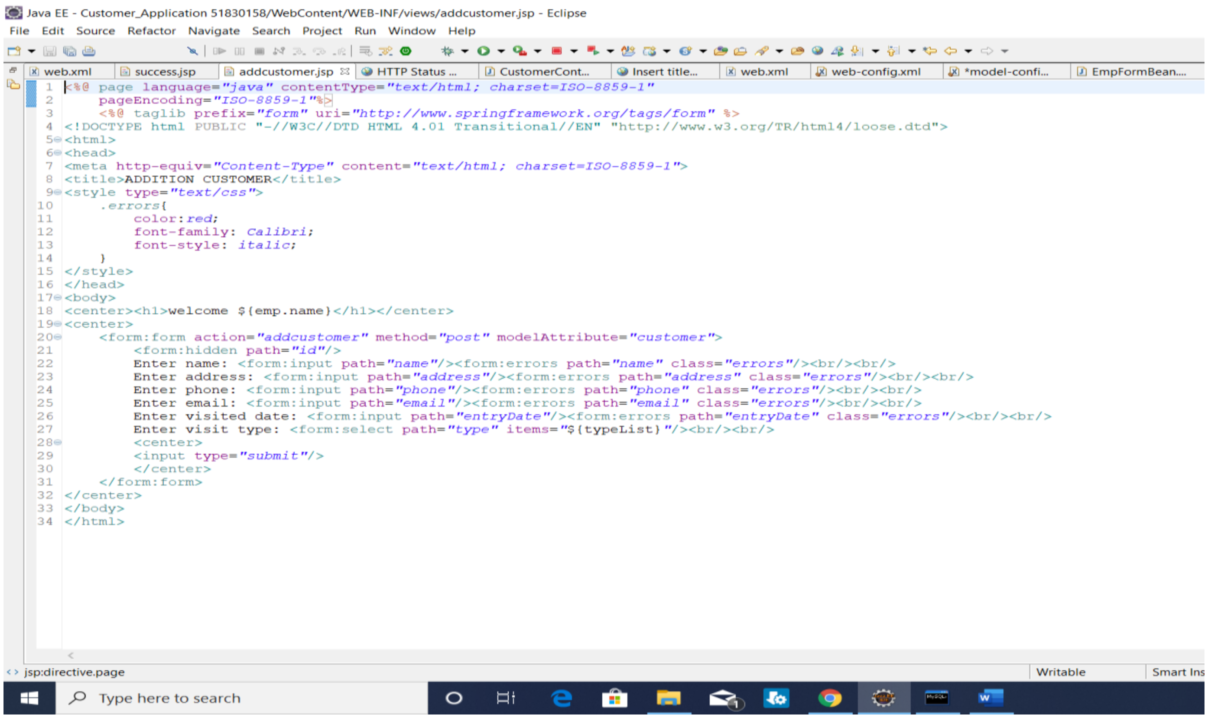
```
<center>
    <table border="2" >
        <thead>
            <tr>
                <th>id</th>
                <th>name</th>
                <th>address</th>
                <th>phone</th>
                <th>email</th>

                <th>entry date</th>
                <th>entry type</th>
                <c:if test="${emp.profile == 'admin'}">
                <th>delete</th>
                <th>update</th>
                </c:if>
            </tr>
        </thead>
        <tbody>
            <c:forEach items="${customers}" var="customer">
                <tr>
                    <td>${customer.id}</td>
                    <td>${customer.name}</td>
                    <td>${customer.address}</td>
                    <td>${customer.phone}</td>
                    <td>${customer.email}</td>
                    <td><fmt:formatDate type="date" value="${customer.entryDate}" /></td>
                    <td>${customer.type}</td>
                    <%-- <td>${book.pubDate}</td> --%>
                    <c:if test="${emp.profile == 'admin'}">
                        <td><a href="delete?id=${customer.id}">delete</a></td>
                        <td><a href="update?id=${customer.id}">update</a></td>
                    </c:if>
                </tr>
            </c:forEach>
        </tbody>
    </table>
    </center>
    <br/>
    <c:if test="${emp.profile == 'admin' || emp.profile == 'manager'}">
        <center><h2><a href="addcustomer">addcustomer</a></h2></center>
    </c:if>
</body>
```

Figure 9: all_customers.jsp

addcustomer.jsp:

The employees who had the profile “admin” and “manager” are able to perform the add new customer operation in which the details of the customer are filled and submitted in order to create a new customer in the database.

In this application the details of the customer are given as name, address, phone number, email, visited date and type of visit in a drop down list.


Java EE - Customer_Application 51830158/WebContent/WEB-INF/views/addcustomer.jsp - Eclipse

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>ADDITION CUSTOMER</title>
<style type="text/css">
    .errors{
        color:red;
        font-family: Calibri;
        font-style: italic;
    }
</style>
</head>
<body>
<center><h1>welcome ${emp.name}</h1></center>
<center>
    <form:form action="addcustomer" method="post" modelAttribute="customer">
        <form:hidden path="id"/>
        Enter name: <form:input path="name"/><form:errors path="name" class="errors"/><br/><br/>
        Enter address: <form:input path="address"/><form:errors path="address" class="errors"/><br/><br/>
        Enter phone: <form:input path="phone"/><form:errors path="phone" class="errors"/><br/><br/>
        Enter email: <form:input path="email"/><form:errors path="email" class="errors"/><br/><br/>
        Enter visited date: <form:input path="entryDate"/><form:errors path="entryDate" class="errors"/><br/><br/>
        Enter visit type: <form:select path="type" items="${typeList}"/><br/><br/>
        <center>
        <input type="submit"/>
        </center>
    </form:form>
</center>
</body>
</html>
```

Figure 10: addcustomer.jsp

3.3. Java Class Creation

3.3.1. Implementation of DAO Class

There are two DAO interfaces used in this application, they are CustomerDAO and EmpDAO and their implementations where the required methods are defined in order to perform the CRUD operations are defined.

In the case of CustomerDAOImpl.java the methods implemented are:

- `public List<Customer> getAllCustomers();`
- `public Customer getCustomerById(int custId);`
- `public Customer addCustomer(Customer customer);`
- `public Customer updateCustomer(Customer customer);`
- `public Customer removeCustomer(int custId);`

In the case of EmpDAOImpl.java the following methods are implemented:

- `public Emp getEmp(String email , String password);`
- `public void addEmp(Emp emp);`
- `public List<Emp> getAllEmp();`

3.3.2. Implementation of Service class

There are two Service interfaces used in this application, they are CustomerService and EmpService and their implementations where the required methods are defined in order to perform the service operations or simply the business logic.

In the case of CustomerServiceImpl.java the methods implemented are:

- `public List<Customer> getAllCustomers();`
- `public Customer getCustomerById(int custId);`
- `public Customer addCustomer(Customer customer);`
- `public Customer updateCustomer(Customer customer);`
- `public Customer removeCustomer(int custId);`

In the case of EmpServiceImpl.java the following methods are implemented:

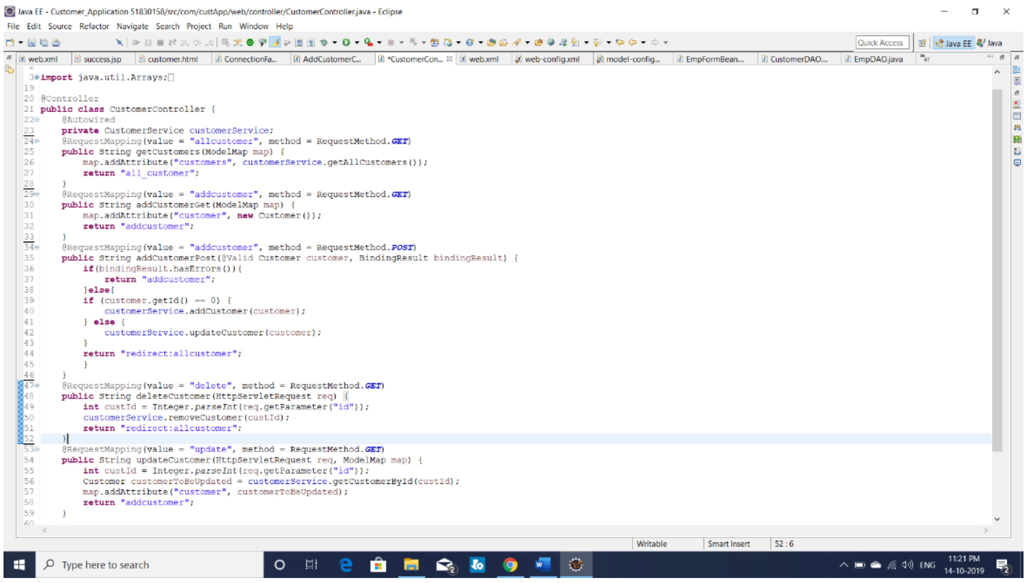
- `public Emp getEmp(String email , String password);`
- `public void addEmp(Emp emp);`
- `public List<Emp> getAllEmp();`

3.3.3. Implementation of Controller class

In this application two controller classes are used, they are CustomerController.java and EmpController.java. In the CustomerController.java there are various GET and POST methods which provides pages to the user and takes the data from the user through the HTML/JSP pages in order to perform various CRUD operation on the customer database.

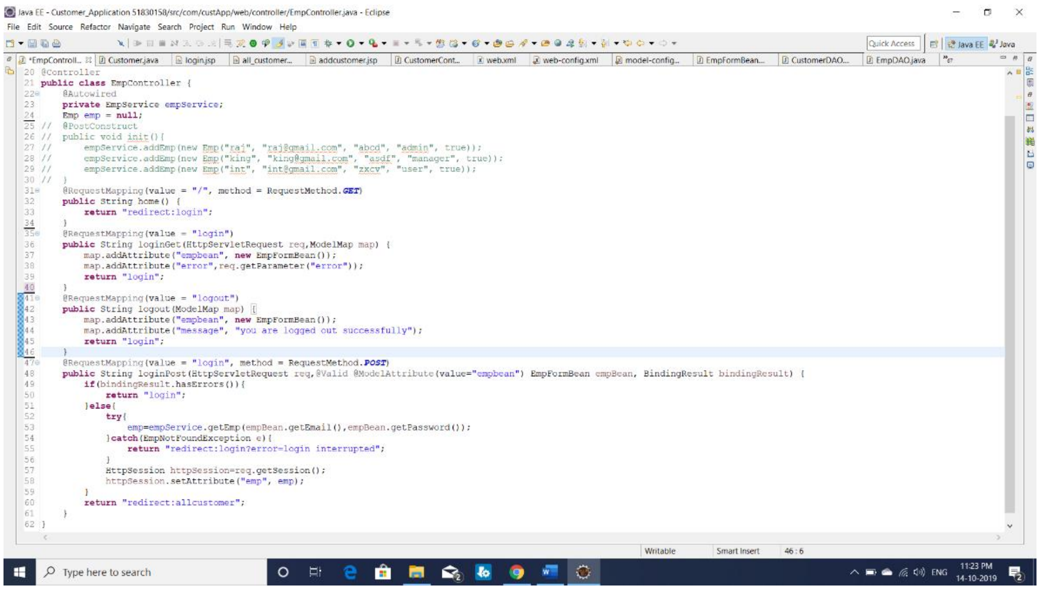
The other controller class was EmpController.java which consists of the home page .i.e. “login.jsp” which is used to validate the employee based on their profile. Here this controller plays the major role in order to verify the login details of the employee and provide him the controls on the customer database.

Logout operation is also provided which makes the employee to log out after the required operation was performed.



```java
import java.util.Arrays;

@Controller
public class CustomerController {
    @Autowired
    private CustomerService customerService;
    @RequestMapping(value = "allcustomer", method = RequestMethod.GET)
    public String getCustomers(ModelMap map) {
        map.addAttribute("customers", customerService.getAllCustomers());
        return "all_customer";
    }
    @RequestMapping(value = "addcustomer", method = RequestMethod.GET)
    public String addCustomerGet(ModelMap map) {
        map.addAttribute("customer", new Customer());
        return "addcustomer";
    }
    @RequestMapping(value = "addcustomer", method = RequestMethod.POST)
    public String addCustomerPost(@Valid Customer customer, BindingResult bindingResult) {
        if(bindingResult.hasErrors()){
            return "addcustomer";
        }else{
        if (customer.getId() == 0) {
            customerService.addCustomer(customer);
        } else {
            customerService.updateCustomer(customer);
        }
        return "redirect:allcustomer";
        }
    }
    @RequestMapping(value = "delete", method = RequestMethod.GET)
    public String deleteCustomer(HttpServletRequest req) {
        int custId = Integer.parseInt(req.getParameter("id"));
        customerService.removeCustomer(custId);
        return "redirect:allcustomer";
    }
    @RequestMapping(value = "update", method = RequestMethod.GET)
    public String updateCustomer(HttpServletRequest req, ModelMap map) {
        int custId = Integer.parseInt(req.getParameter("id"));
        Customer customerToBeUpdated = customerService.getCustomerById(custId);
        map.addAttribute("customer", customerToBeUpdated);
        return "addcustomer";
    }
```

Figure 11: CustomerController.java



```java
@Controller
public class EmpController {
    @Autowired
    private EmpService empService;
    Emp emp = null;
//  @PostConstruct
//  public void init(){
//      empService.addEmp(new Emp("raj", "raj@gmail.com", "abcd", "admin", true));
//      empService.addEmp(new Emp("king", "king@gmail.com", "asdf", "manager", true));
//      empService.addEmp(new Emp("int", "int@gmail.com", "zxcv", "user", true));
//  }
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home() {
        return "redirect:login";
    }
    @RequestMapping(value = "login")
    public String loginGet(HttpServletRequest req,ModelMap map) {
        map.addAttribute("empbean", new EmpFormBean());
        map.addAttribute("error",req.getParameter("error"));
        return "login";
    }
    @RequestMapping(value = "logout")
    public String logout(ModelMap map) {
        map.addAttribute("empbean", new EmpFormBean());
        map.addAttribute("message", "you are logged out successfully");
        return "login";
    }
    @RequestMapping(value = "login", method = RequestMethod.POST)
    public String loginPost(HttpServletRequest req,@Valid @ModelAttribute(value="empbean") EmpFormBean empBean, BindingResult bindingResult) {
        if(bindingResult.hasErrors()){
            return "login";
        }else{
            try{
                emp=empService.getEmp(empBean.getEmail(),empBean.getPassword());
            }catch(EmpNotFoundException e){
                return "redirect:login?error=login interrupted";
            }
            HttpSession httpSession=req.getSession();
            httpSession.setAttribute("emp", emp);
        }
        return "redirect:allcustomer";
    }
}
```

Figure 12: EmpController.java

4. **Project Demo**

The home page “login” screen will be displayed once the project was made to run on server. Here the Tomcat was used as the server in order to run the application. The login screen consists of two fields which askes the user to enter email and password which validated preciously such that they can not be left. Once the details are filled and submitted the POST method of EmpController was activated and compares the login details with the existing table (emp table) and proceeds if the details provided are correct. If the details were incorrect the page redirects to login page and displays the message “Login Interrupted”.

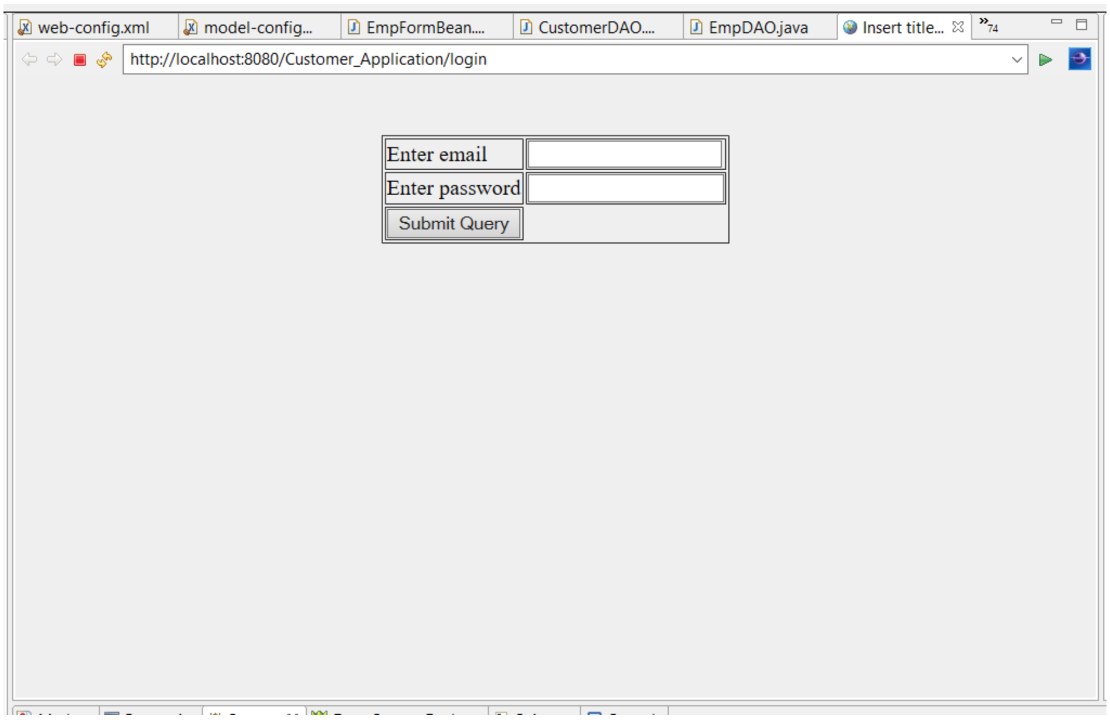
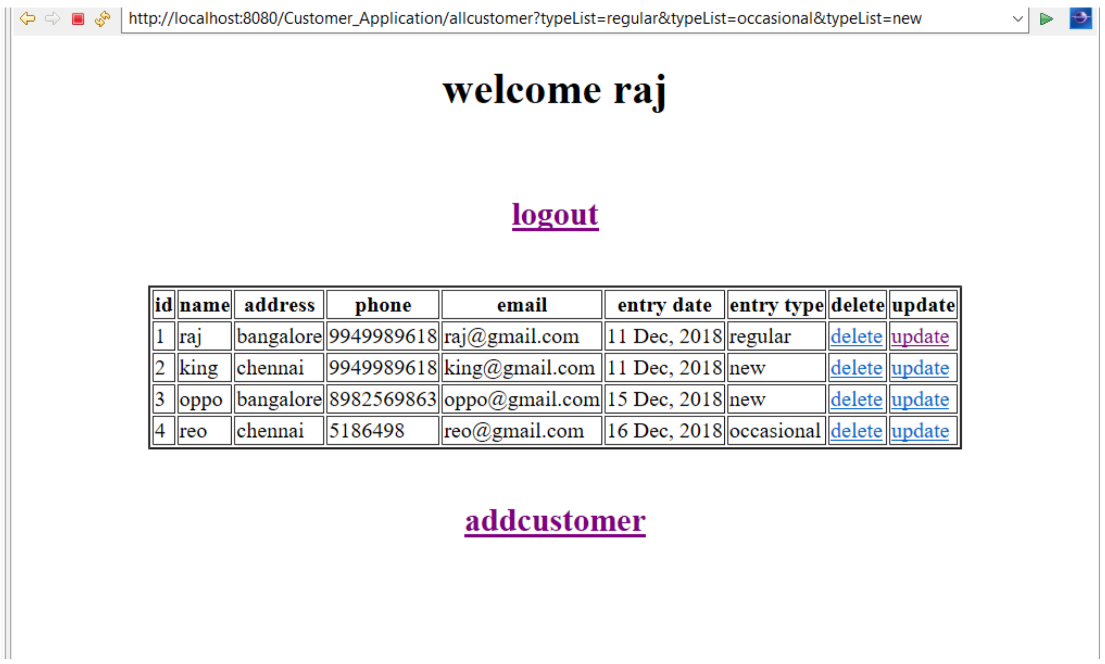


Figure 13: login page

Once the details are entered correctly and submitted the page redirects to display the customers details and displays the links where the crud operations are performed based on the profile of the employee. Here in this case I am logged in as admin where we can see all the provisions like delete, update and add customer links for the crud operations.


http://localhost:8080/Customer_Application/allcustomer?typeList=regular&typeList=occasional&typeList=new

welcome raj

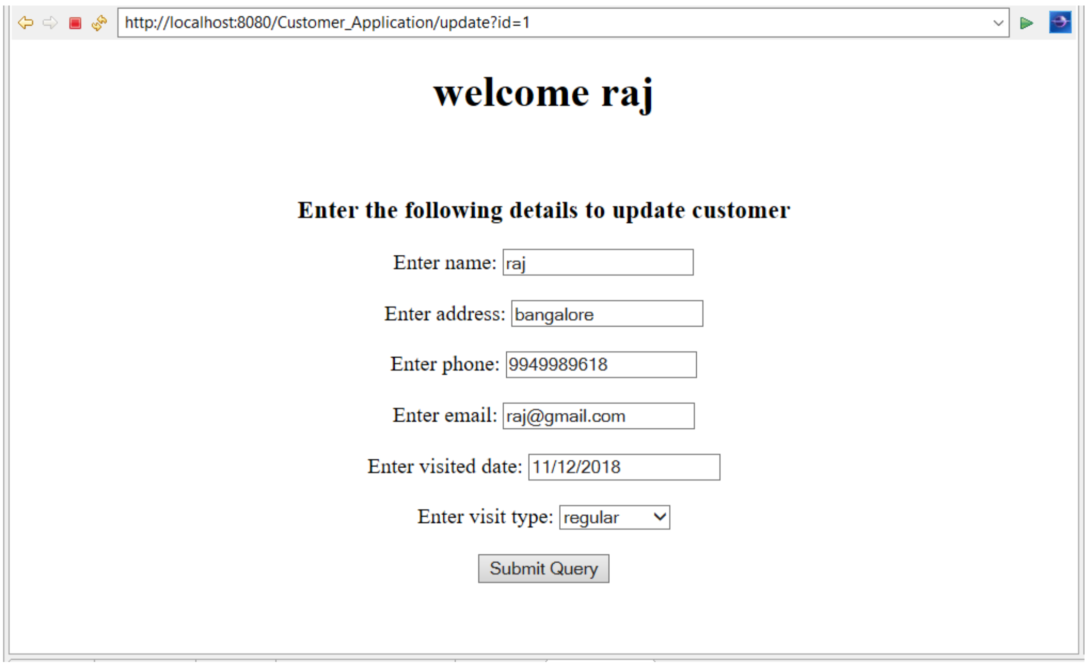
logout

id	name	address	phone	email	entry date	entry type	delete	update
1	raj	bangalore	9949989618	raj@gmail.com	11 Dec, 2018	regular	delete	update
2	king	chennai	9949989618	king@gmail.com	11 Dec, 2018	new	delete	update
3	oppo	bangalore	8982569863	oppo@gmail.com	15 Dec, 2018	new	delete	update
4	reo	chennai	5186498	reo@gmail.com	16 Dec, 2018	occasional	delete	update

addcustomer

Figure 14: display page for admin

If the delete link was clicked the specific attribute gets deleted and if the update link was clicked the prefilled form was displayed where the emp can edit the customer information and submit it in order to make changes in database.


http://localhost:8080/Customer_Application/update?id=1

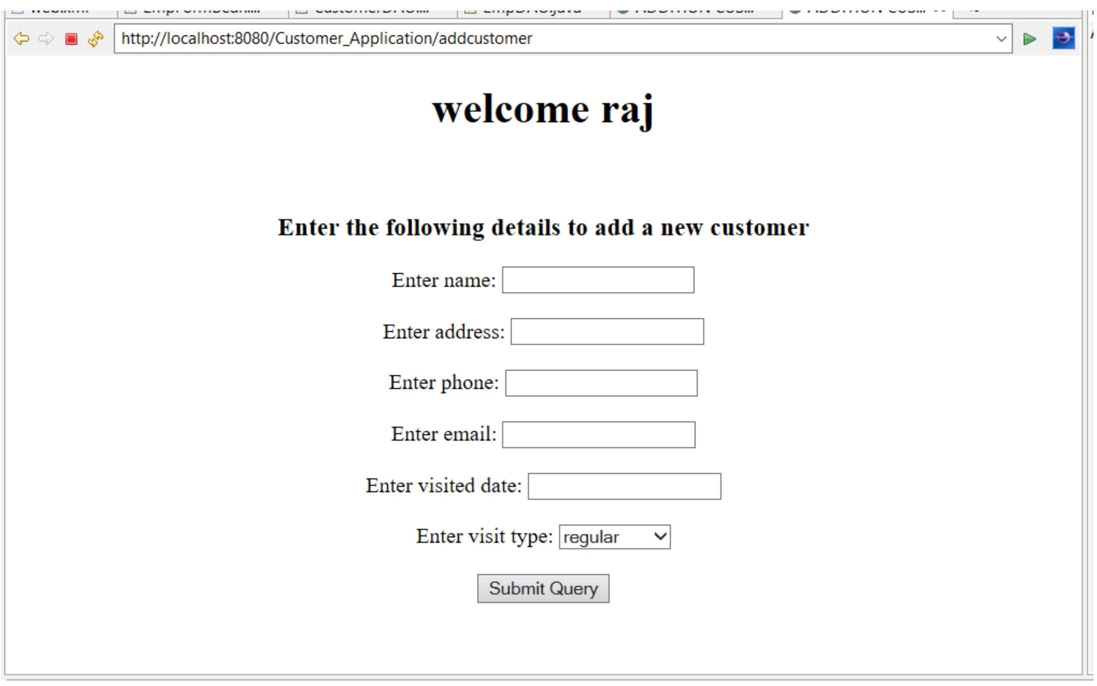
welcome raj

Enter the following details to update customer

Enter name:	raj
Enter address:	bangalore
Enter phone:	9949989618
Enter email:	raj@gmail.com
Enter visited date:	11/12/2018
Enter visit type:	regular

Figure 15: update the customer page

If the addcustomer link was clicked which can seen in the figure 14, it redirects to a new page where the emp is able to enter the customer details like name, address, phone number, email, visited date and visit type as shown in the following figure.


http://localhost:8080/Customer_Application/addcustomer

welcome raj

Enter the following details to add a new customer

Enter name:	
Enter address:	
Enter phone:	
Enter email:	
Enter visited date:	
Enter visit type:	regular

Submit Query

Figure 16: add new customer page

5. Conclusion

The "CUSTOMER APPLICATION" is mainly based on CRUD operations in which the customer's details are added, deleted, read and updated by a valid employee of the company. In order to view or make any changes in the customer database, the employee should prove his authenticity by completing a login formality, in which he/she is asked to enter his email and password. The employees are segregated into three categories like "ADMIN", "MANAGER" and "USER". Each category has some specific powers like admin can perform all the operations on customer database, where the manager can only add a new customer to the database and the user can only view the information about the customers in the database.

The requirements mentioned above can be implemented using different tools like eclipse, spring and database like MySQL. The results of the CUSTOMER APPLICATION are given in the project demo where the crud operations performed can be viewed in the form of screen shots attached. These types of applications are mainly used in shopping malls and super markets to monitor their growth and customer interaction.