**CUSTOMER APPLICATION**

**(CRUD operations)**

**Prepared by**

**Tumula Mani Kota Rajasekhar (51830158)**

Table Of Contents

[1. Introduction](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#introduction) about the project

[1.1 CRUD Operations](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#crudOperations)

[1.2 CRUD Benefits](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#crudBenefits)

[2. Tools](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#jsfCrudExample) and project structure

[2.1 Tools used](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#tools)

[2.2 Project Structure](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#projectStructure)

[2.3 Project Creation](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#projectCreation)

[3. Application Building](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#applicationBuilding)

[3.1 Database & Table Creation](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#dbAndTableCreation)

[3.2 File Creation](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#fileCreation)

[3.2.1 Implementation of HTML/JSP](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#inputOutputImplementation) pages

[3.3 Java Class Creation](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#javaClassCreation)

[3.3.1 Implementation of DAO Class](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#daoLayerImplementation)

3.3.2 Implementation of Service class

3.3.3 Implementation of Controller class

[4. Project Demo](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#projectDemo)

[5. Conclusion](https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-crud-application-example/#projectConclusion)

## **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.

### 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

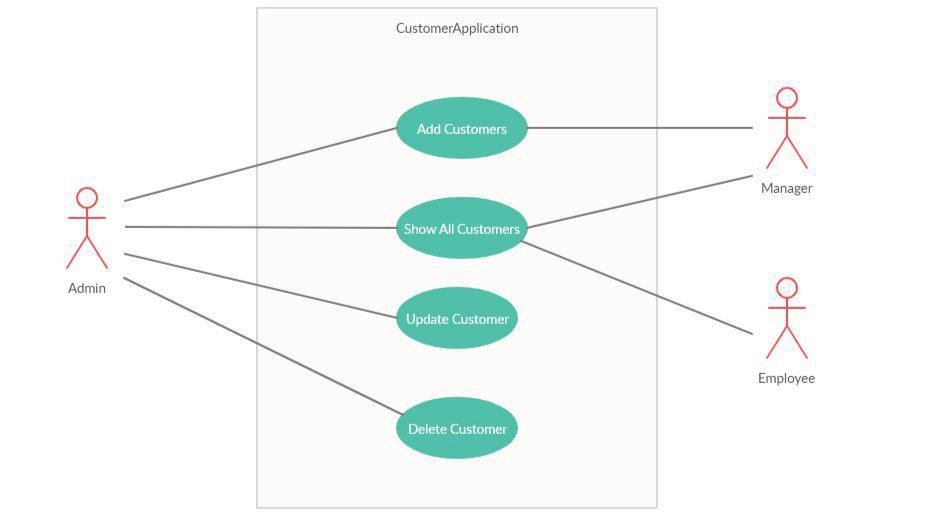
This application lets you manage customer’s database such as add new customer and update/delete customer information. But after making sure that the one who is doing those operations is a valid employee. This can be done by checking his email id and password which are stored in another database table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

### CRUD Benefits

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

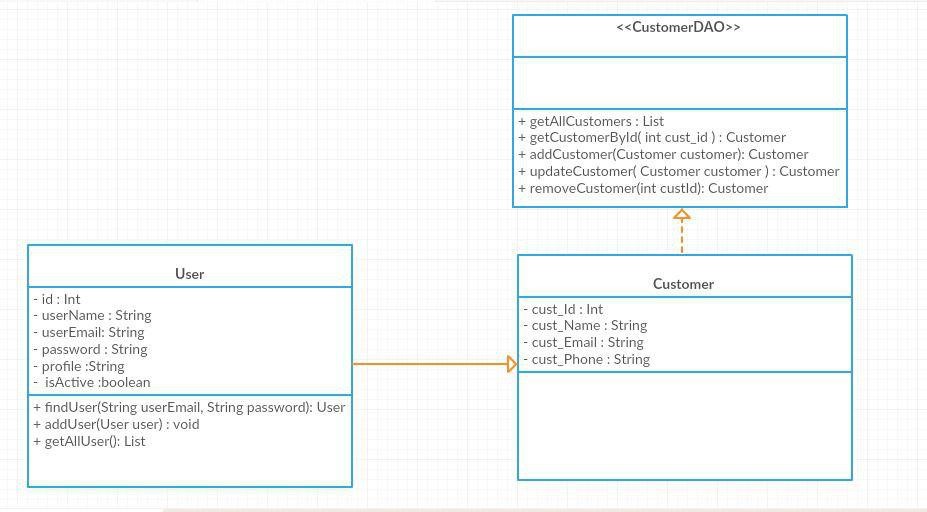
* Improves data security and data access to users by using host and query languages
* Greater data integrity and independence of applications programs
* Improves application performance by reducing the data redundancy

**Use Case Diagram:**

****

**User Case Diagram**

**Class Diagram**:



Class Diagram

1. **Tools and project structure**

### Tools Used

### ECLIPSE

### SPRING 3.8

### MY SQL 5.1.48

### TOMCAT 7.0.57

### Project Structure

### The project structure mainly consists of three layers namely:

### Web Layer

### Model Layer

### Service Layer

### Persistence Layer

### 

### 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.

### 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.

### 

### 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"* />

### 

### 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

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

<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>

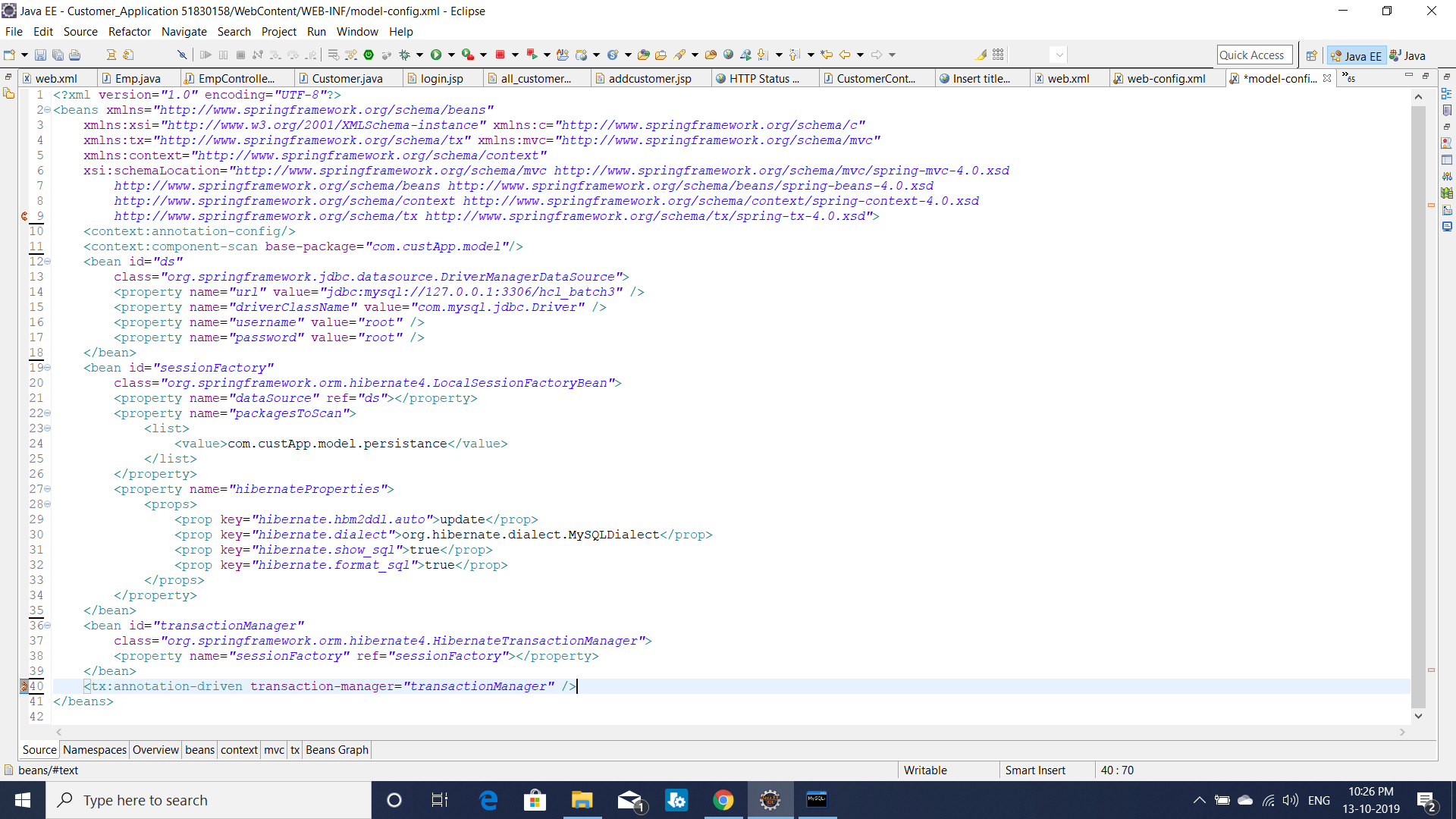


Figure 5: project creation step 4 (model-config.xml)

1. **Application Building**

**3.1.** [**Database & Table Creation**](#dbAndTableCreation)

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

<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.

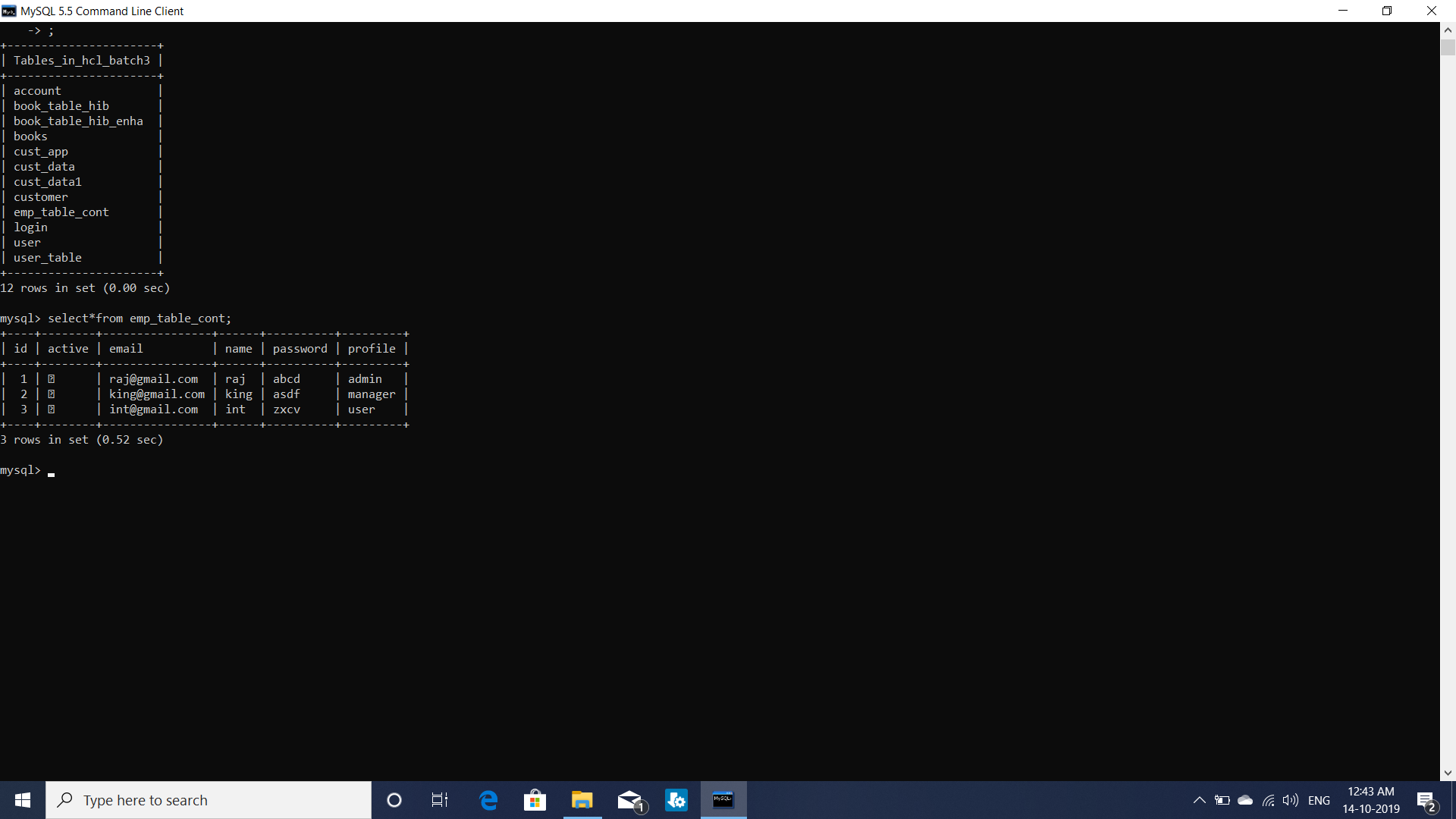


Figure 6: employee table

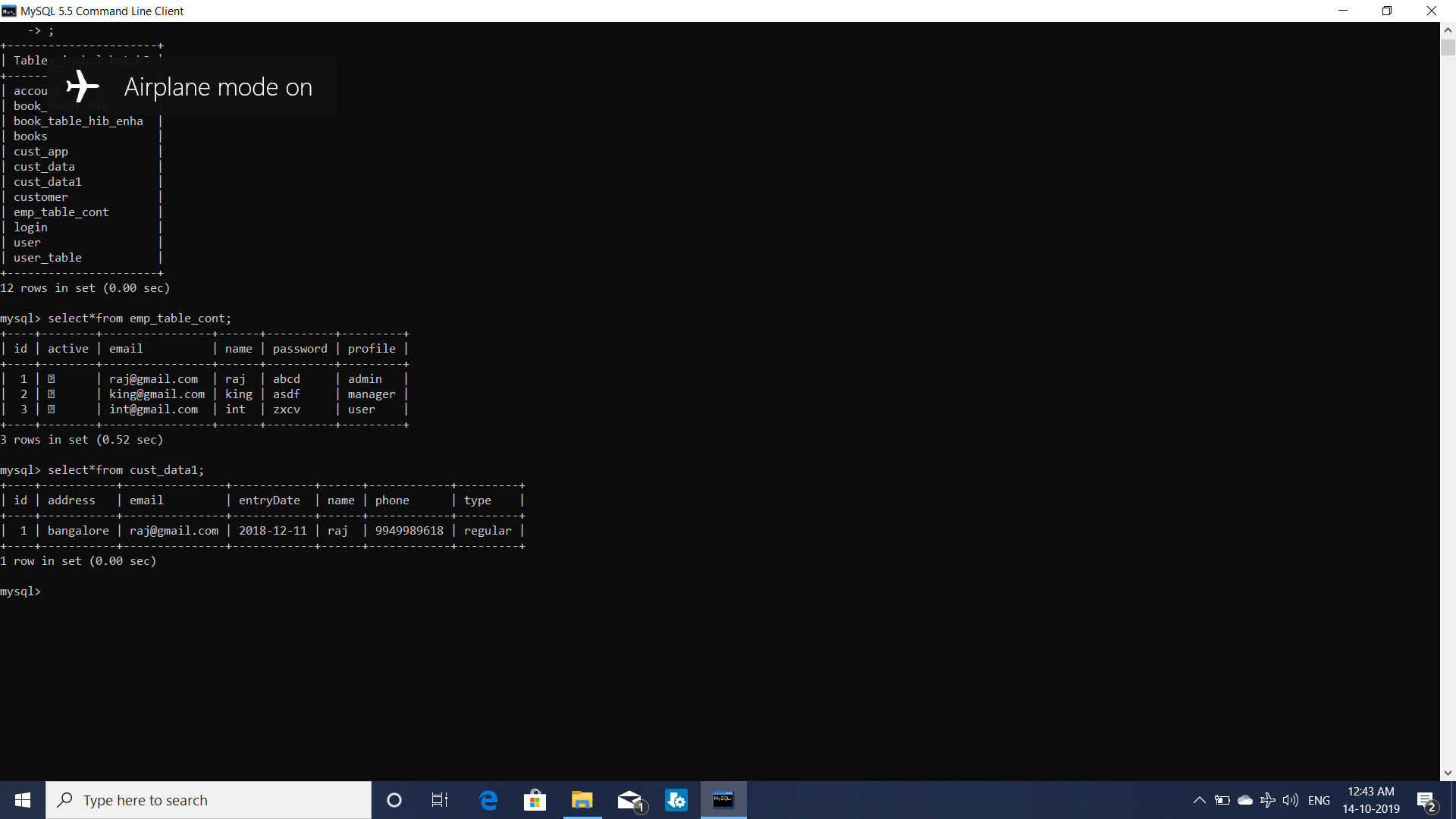


Figure 7: customer table

**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.

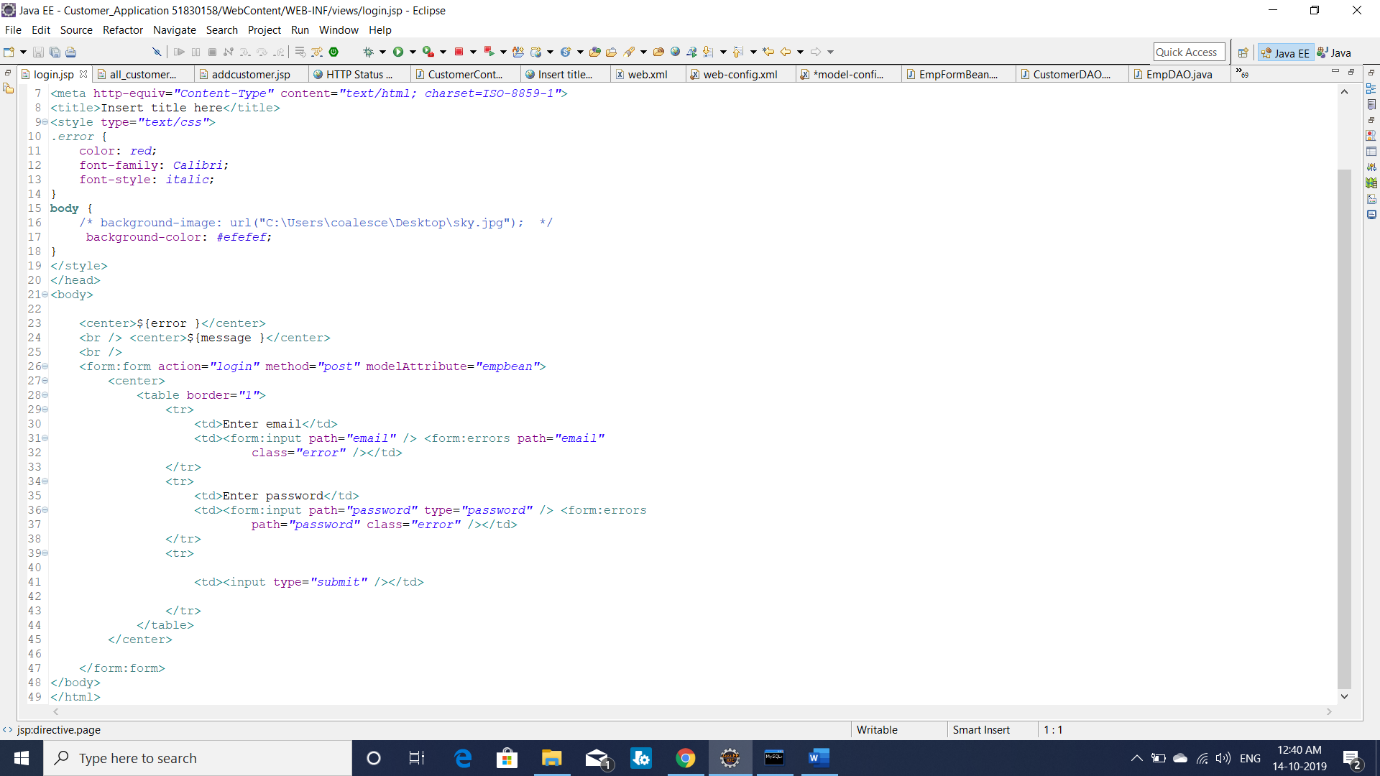


Figure 8: home page (login.jsp)

**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.

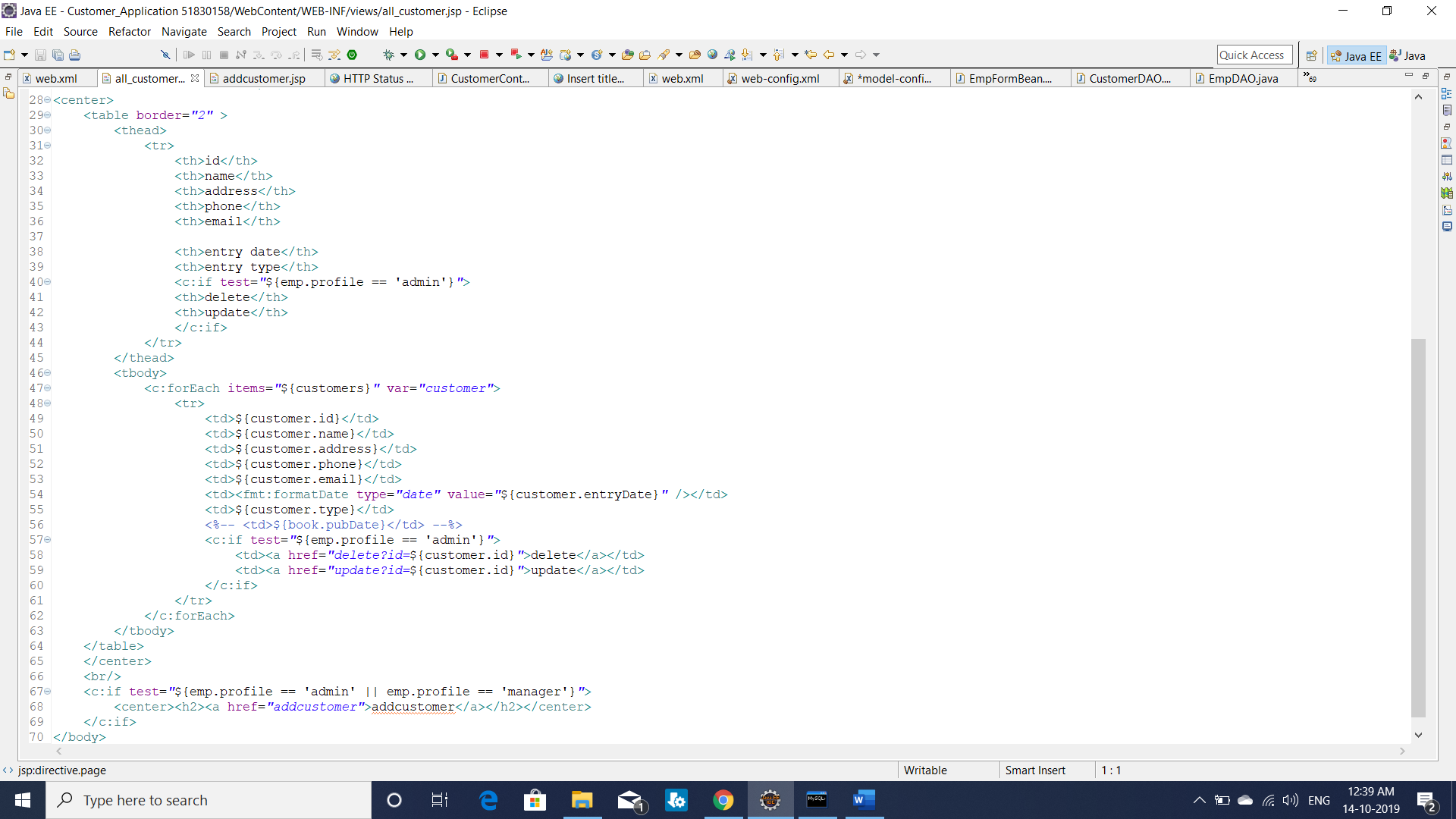


Figure 9: all\_customers.jsp

**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.

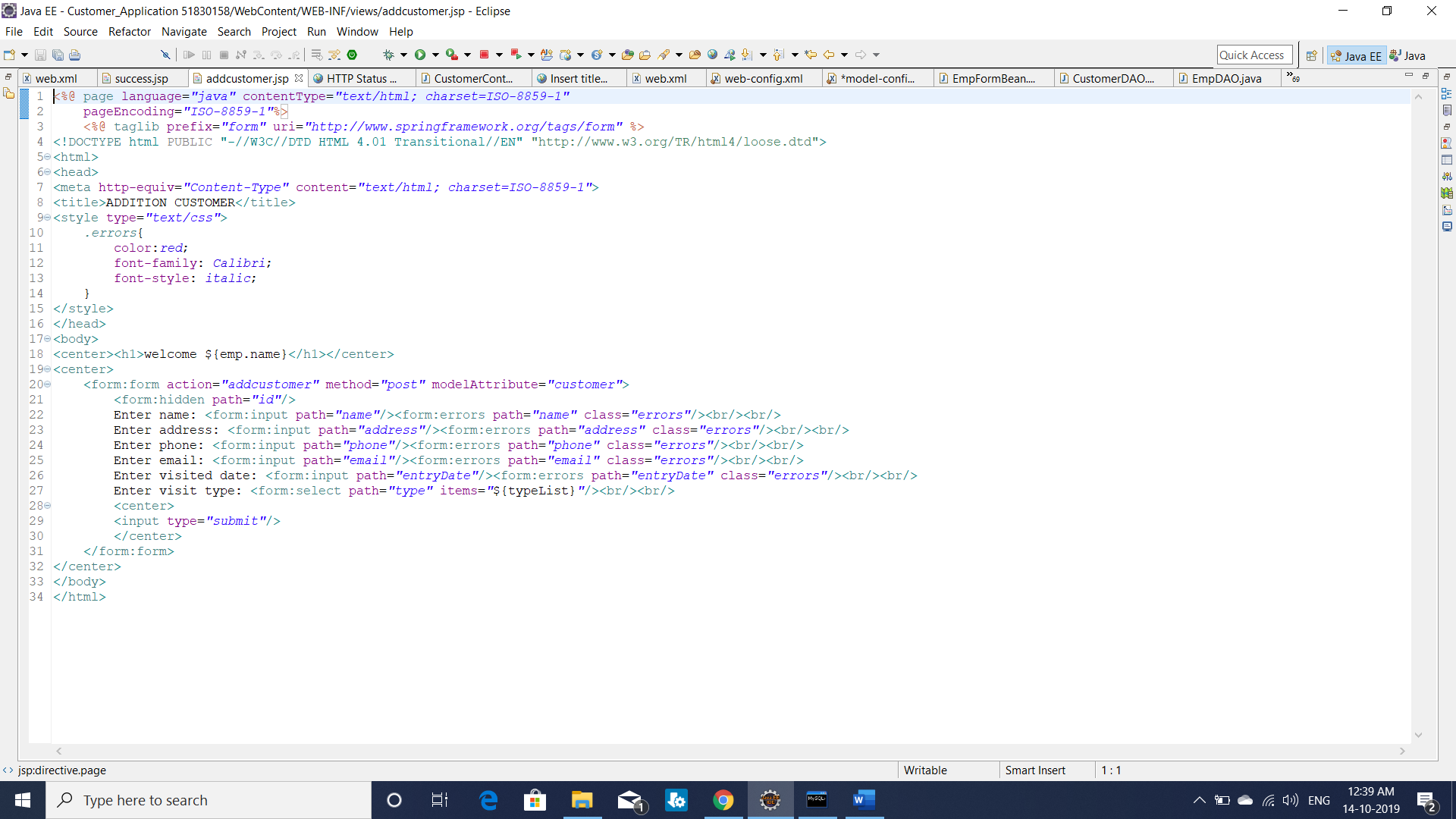


Figure 10: addcustomer.jsp

**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.

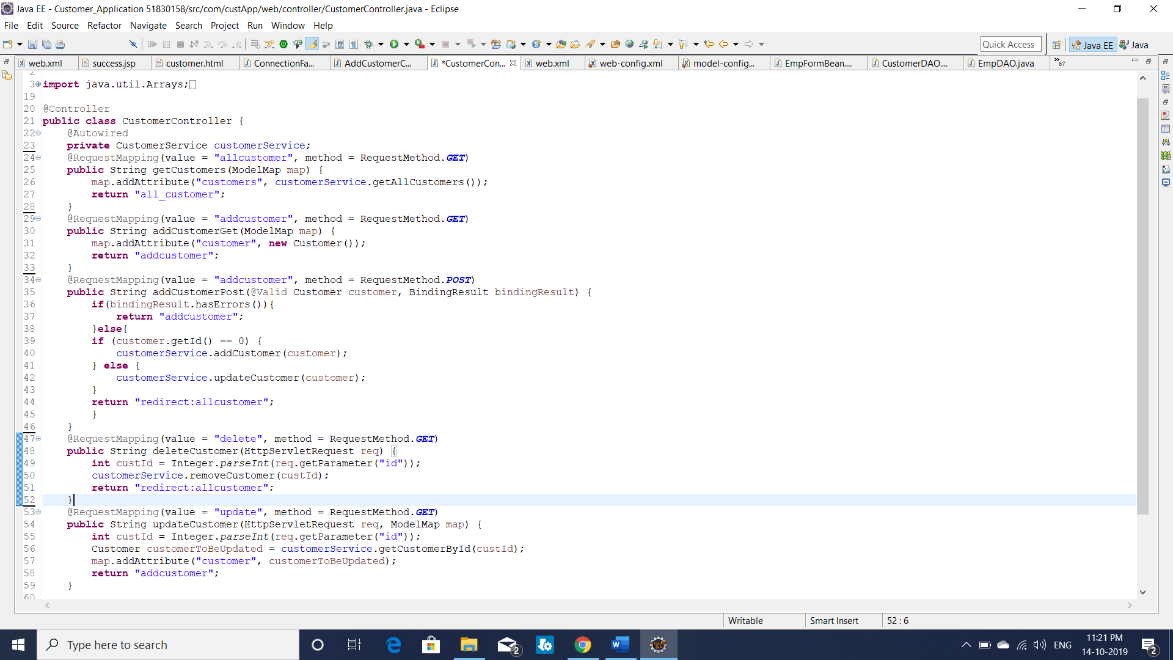


Figure 11: CustomerController.java

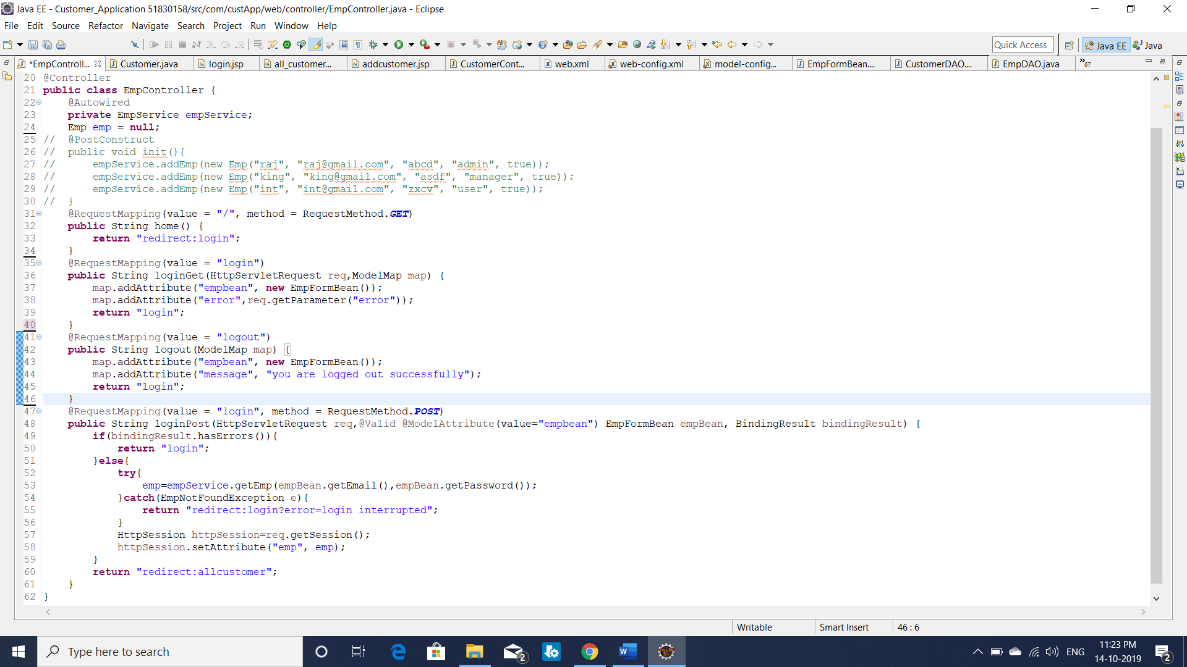


Figure 12: EmpController.java

1. **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”.

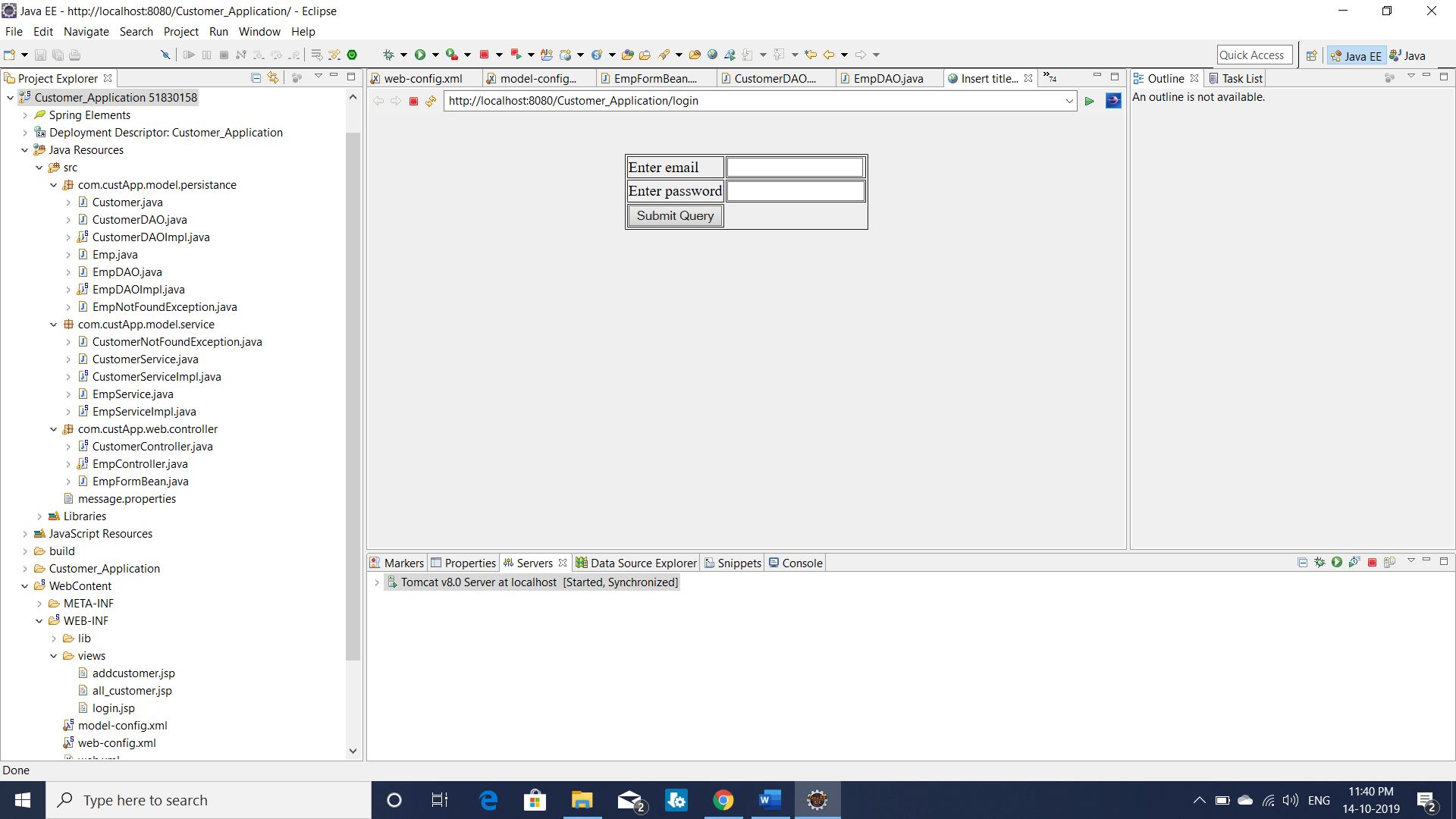


Figure 13: login page

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.

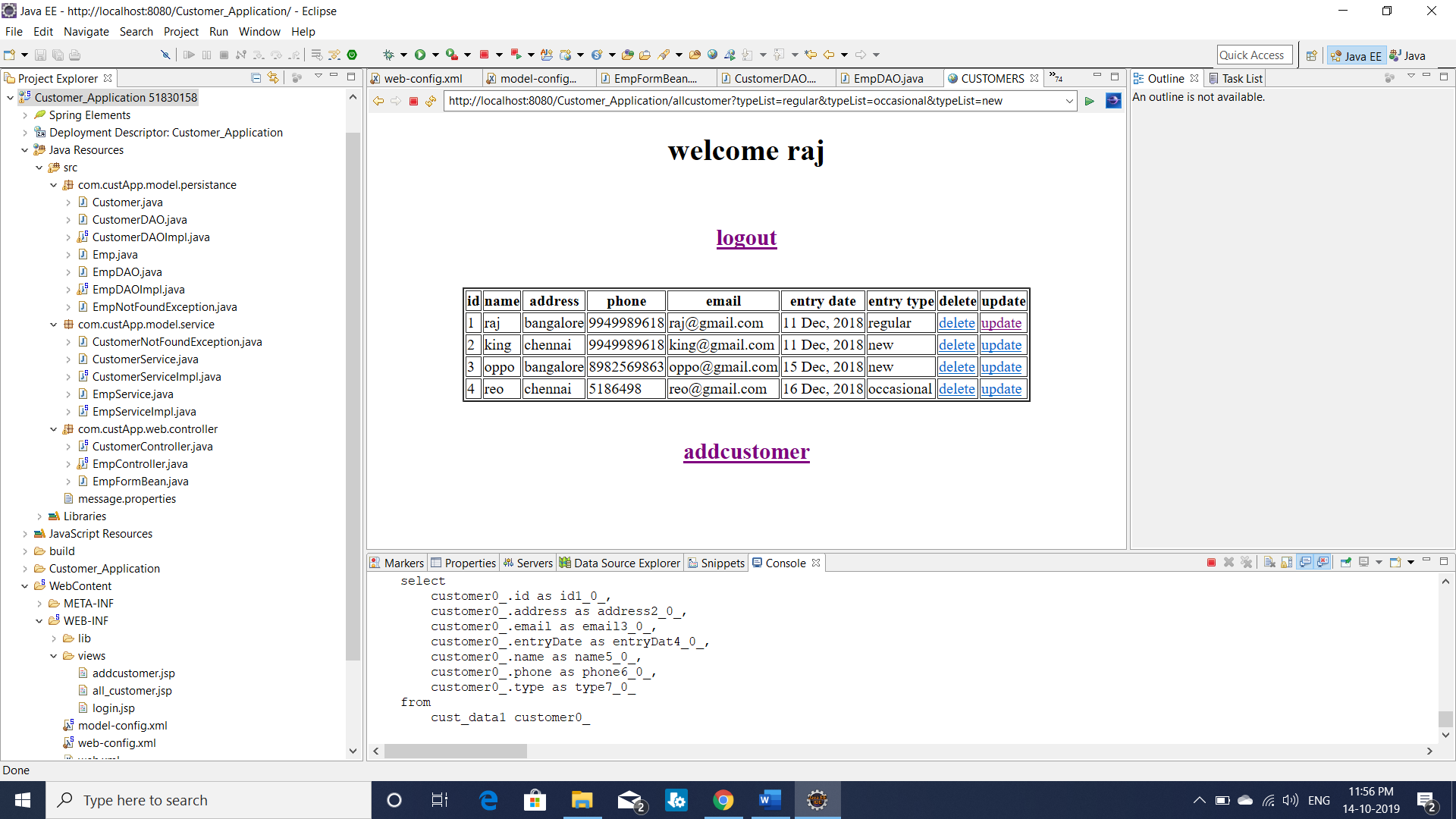


Figure 14: display page for admin

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.

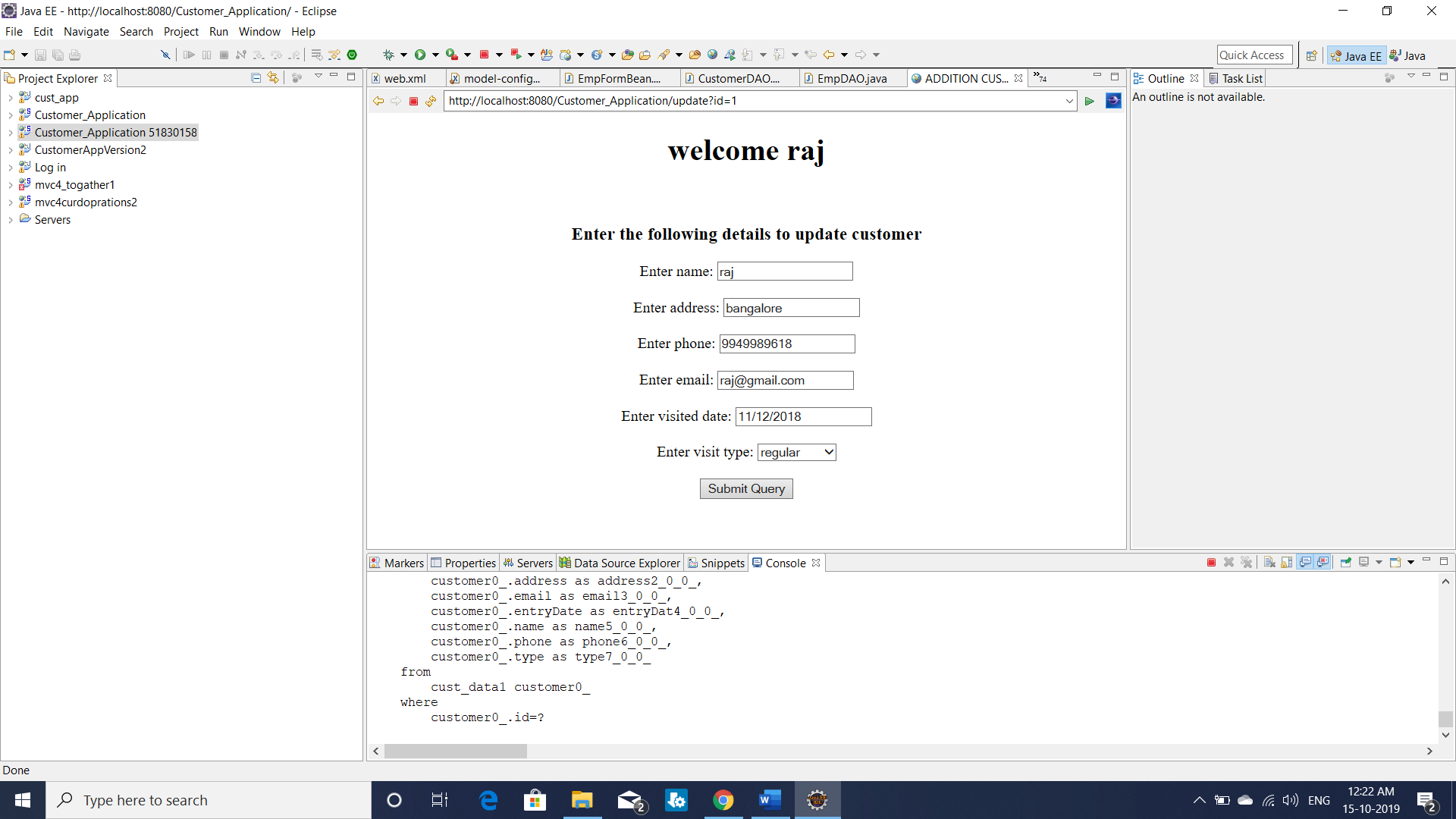


Figure 15: update the customer page

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.

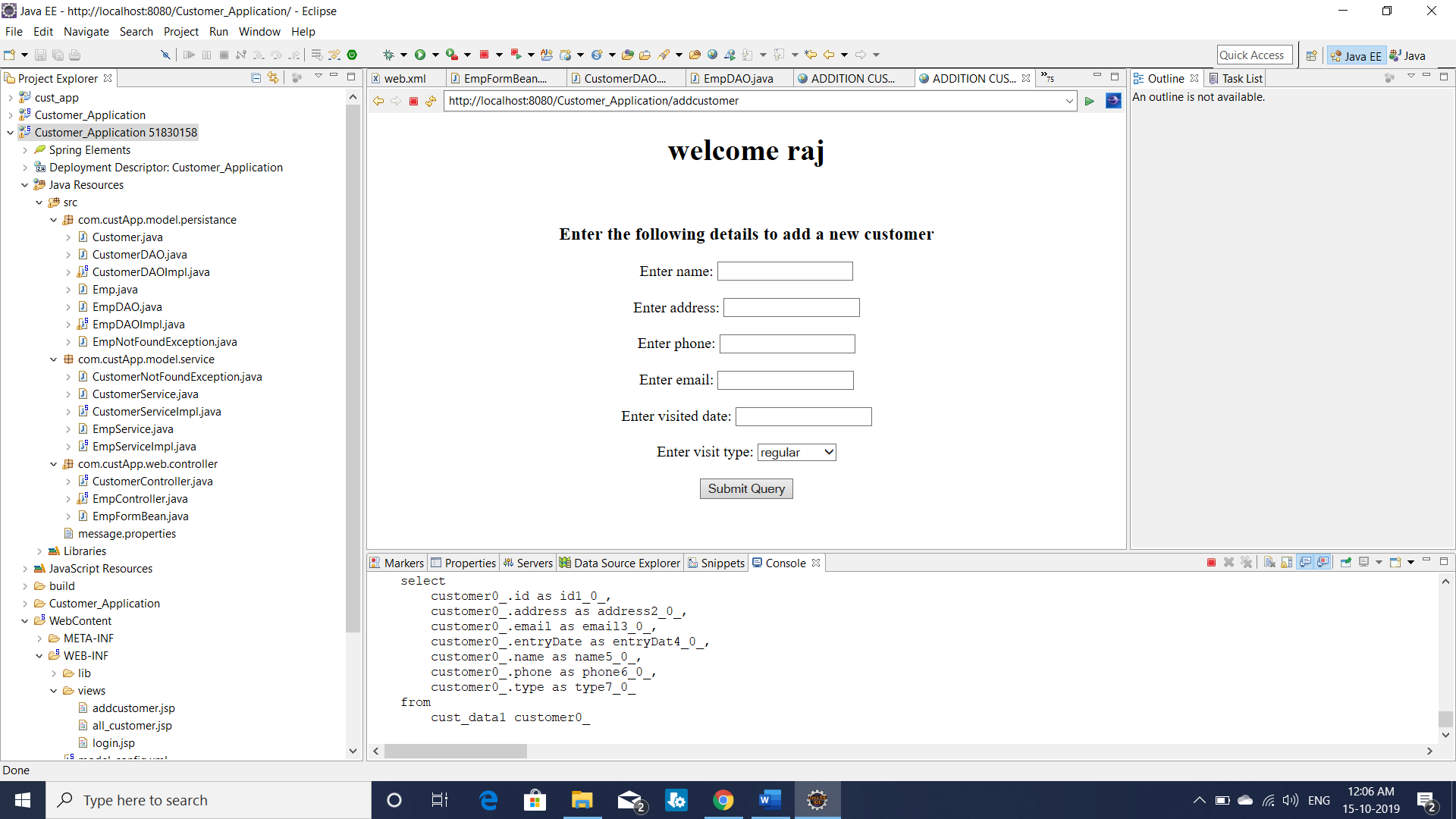


Figure 16: add new customer page

1. **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.