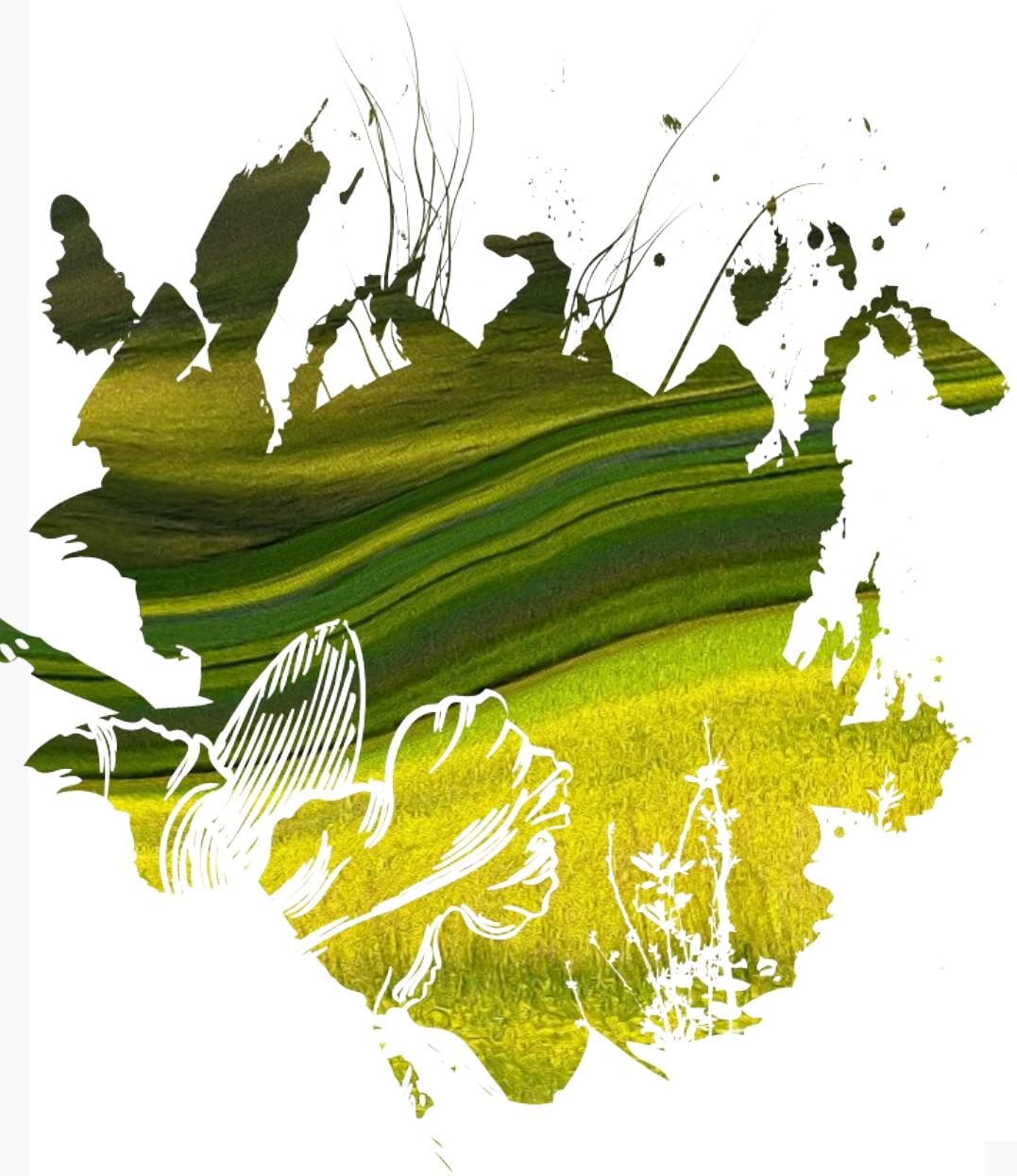


Bộ môn: Kỹ Thuật Phần Mềm

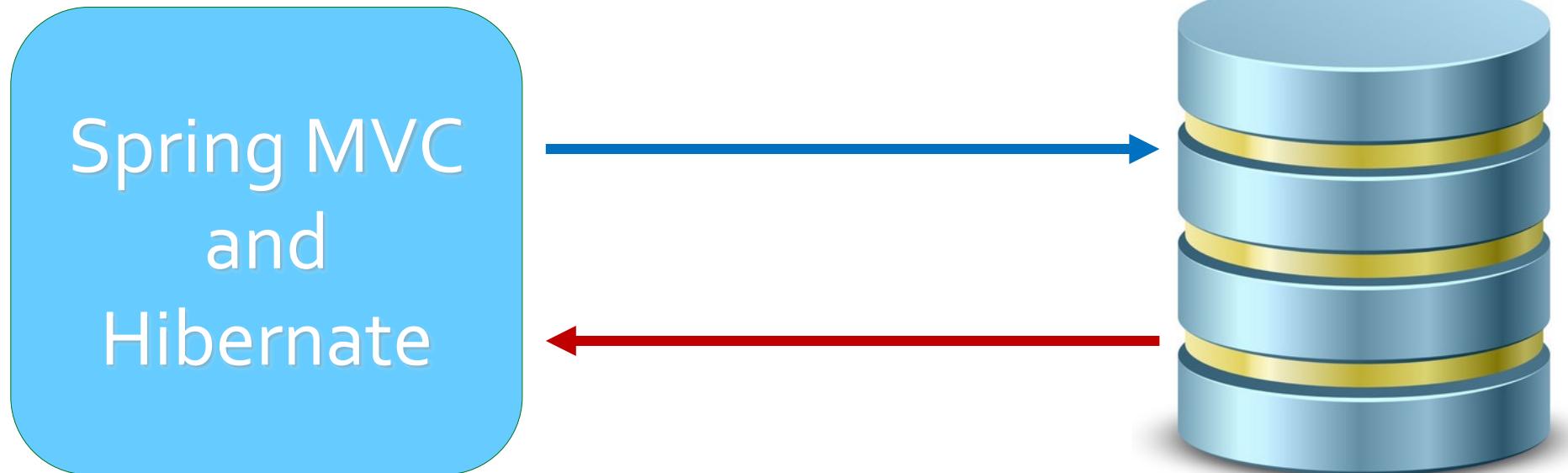
Lập trình www Java



Spring MVC – Web App - Hibernate



Full working Spring MVC and Hibernate app that connects to a database



- » **Setup Database**
- » **Setup Dev Environment**
- » **Listing the customers**
- » **Add a new customer**
- » **Update a customer**
- » **Delete a customer**

Application Demo - List



← → ⌂ ⓘ localhost:8081/web-customer-tracker/customer/list

CRM - Customer Relationship Manager

Add Customer 

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
Mary	Public	mary@secode.com	Update Delete
Ajay	Rao	ajaysecode.com	Update Delete

Application Demo – Add new



← → ⌛ ⓘ localhost:8081/web-customer-tracker/customer/showFormForAdd

CRM - Customer Relationship Manager

Save Customer

First name:

Last name:

Email:

Save 

[Back to List](#)

Application Demo - Update



← → ⌂

localhost:8081/web-customer-tracker/customer/list

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
David	Gueta	Gueta@secode.com	Update Delete
Mary	Public	mary@secode.com	Update Delete
Ajay	Rao	ajaysecode.com	Update Delete

Application Demo - Update



← → C ⓘ localhost:8081/web-customer-tracker/customer/showFormForUpdate?customerId=3

CRM - Customer Relationship Manager

Save Customer

First name:

Last name:

Email:

[Back to List](#)

Application Demo - Update



← → ⌂ ⓘ localhost:8081/web-customer-tracker/customer/list

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
David	Gueta	Gueta@secode.com	Update Delete
Mary	Public	mary@secode.com	Update Delete
Ajay	Rao	ajay@secode.com	Update Delete

Application Demo - Delete



← → ⌂ ⓘ localhost:8081/web-customer-tracker/customer/list

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
David	Gueta	Gueta@secode.com	Update Delete
Mary	Public	mary@secode.com	Update Delete
Ajay	Rao	ajay@secode.com	Update Delete 

Application Demo - Delete



← → ⌂ ⓘ localhost:8081/web-customer-tracker/customer/list

CRM - Customer

Add Customer

localhost:8081 says
Are you sure you want to delete this customer?

OK Cancel

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
David	Gueta	Gueta@secode.com	Update Delete
Mary	Public	mary@secode.com	Update Delete
Ajay	Rao	ajay@secode.com	Update Delete

Application Demo - Delete



← → ⌂ ⓘ localhost:8081/web-customer-tracker/customer/list

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
David	Gueta	Gueta@secode.com	Update Delete
Mary	Public	mary@secode.com	Update Delete



Setup Database

1. Create a new sql server user for application:

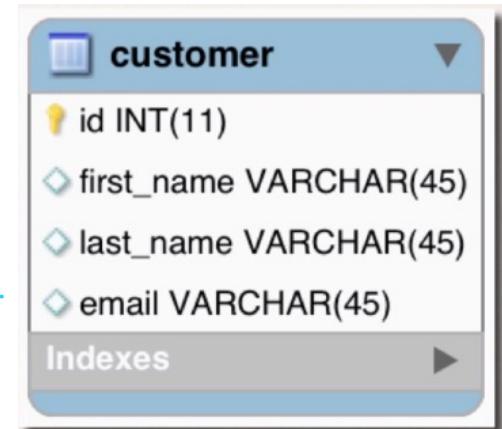
- Id: **springstudent**
- Pass: **springstudent**

2. Create database: **web_customer_tracker**

3. Create table: **customer**

4. Input sample data

	id	first_name	last_name	email
1	1	David	Adams	david@secode.com
2	2	John	Doe	john@secode.com
3	3	Ajay	Rao	ajaysecode.com
4	4	Mary	Public	mary@secode.com
5	5	Maxwell	Dixon	max@secode.com





Setup Dev Environment

1. Setup pom.xml

2. Setup config files:

- ***spring-mvc-crud-demo-servlet.xml;***
- ***web.xml***

Setup Dev Environment - pom.xml



```
<!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <version>8.4.1.jre11</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.16.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.2</version>
</dependency>
```

MS SQL Server
connection
driver

Hibernate
driver

Connection
Pool driver

Setup Dev Environment - pom.xml



```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-c3p0 -->
```

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-c3p0</artifactId>
    <version>5.2.12.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.mchange/mchange-commons-java -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>mchange-commons-java</artifactId>
    <version>0.2.11</version>
</dependency>
```

Connection
Pool driver

Connection
Pool driver

Setup Dev Environment - pom.xml



```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.spring
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
```

Spring MVC

Spring Transaction

Spring Object
Relational
Mapping

Setup Dev Environment - pom.xml



```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet:jstl:1.2 -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet-api:3.0.1 -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.1</version>
</dependency>
```

Setup Dev Environment - **spring-mvc-crud-demo-servlet.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:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx.xsd">
```

List of Name Spaces

Setup Dev Environment - **spring-mvc-crud-demo-servlet.xml**



<!-- Define Spring MVC view resolver -->

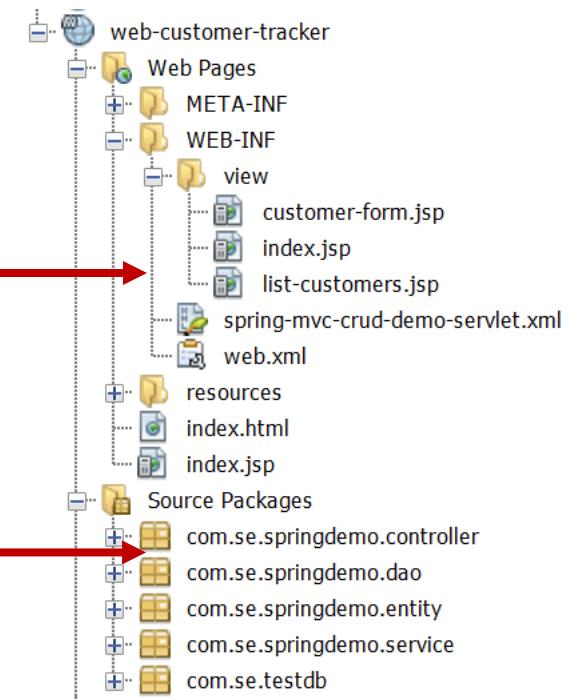
```
<bean  
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
    <property name="prefix" value="/WEB-INF/view/" />  
    <property name="suffix" value=".jsp" />  
</bean>
```

<!-- Add support for component scanning -->

```
<context:component-scan base-package="com.se.springdemo" />
```

<!-- Add support for conversion, formatting and validation support -->

```
<mvc:annotation-driven/>
```



Setup Dev Environment - **spring-mvc-crud-demo-servlet.xml**



<!-- Step 1: Define Database DataSource / connection pool -->

```
<bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
  destroy-method="close">
  <property name="driverClass" value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
  <property name="jdbcUrl" value="jdbc:sqlserver://localhost:1433;databaseName=web_customer_tracker" />
  <property name="user" value="springstudent" />
  <property name="password" value="springstudent" />
  <!-- these are connection pool properties for C3P0 -->
  <property name="minPoolSize" value="5" />
  <property name="maxPoolSize" value="20" />
  <property name="maxIdleTime" value="30000" />
</bean>
```

<!-- Step 2: Setup Hibernate session factory -->

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
  <property name="dataSource" ref="myDataSource" />
  <property name="packagesToScan" value="com.se.springdemo.entity" />
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.SQLServer2012Dialect</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
```

Setup Dev Environment - **spring-mvc-crud-demo-servlet.xml**



```
<!-- Step 3: Setup Hibernate transaction manager -->
<bean id="myTransactionManager"
      class="org.springframework.orm.hibernate5.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

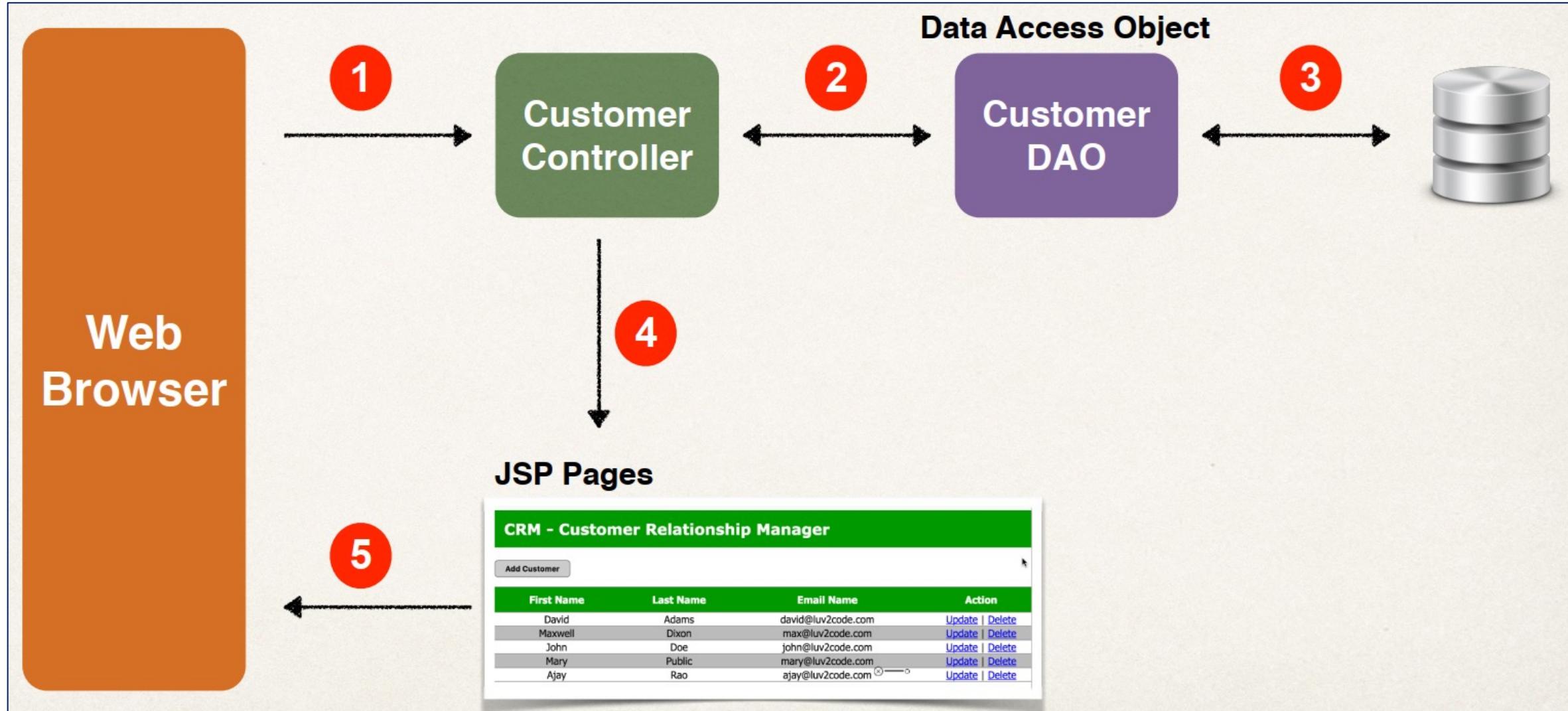
Defined
in step 2

```
<!-- Step 4: Enable configuration of transactional behavior based on annotations -->
<tx:annotation-driven proxy-target-class="true" transaction-manager="myTransactionManager" />
<!-- Add support for reading web resources: css, images, js, etc ... -->
<mvc:resources location="/resources/" mapping="/resources/**"></mvc:resources>
```



List of Customers

List Customers – App Process



List Customers Development Process



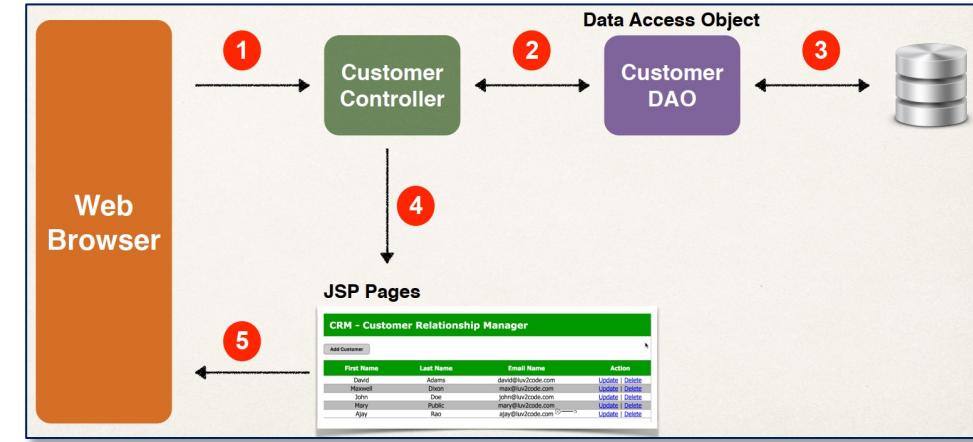
1. Create **Customer.java**

2. Create:

- Interface **CustomerDAO.java**
- Class **CustomerDAOImpl.java**

3. Create controller **CustomerController.java**

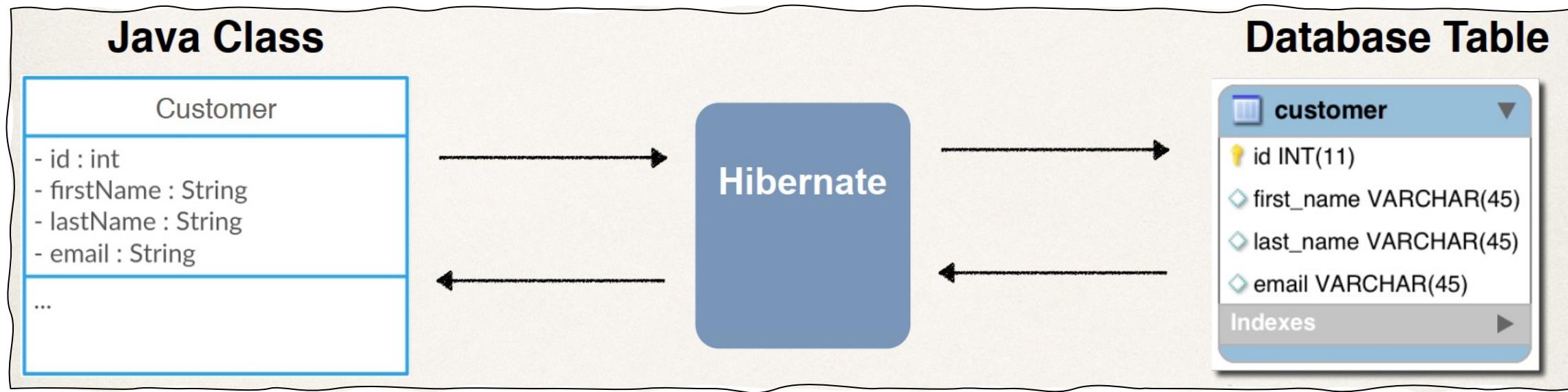
4. Create view **list-customer.jsp**



Entity Class

Java class that is mapped to a database table

List Customers – Object Relation Mapping (ORM)

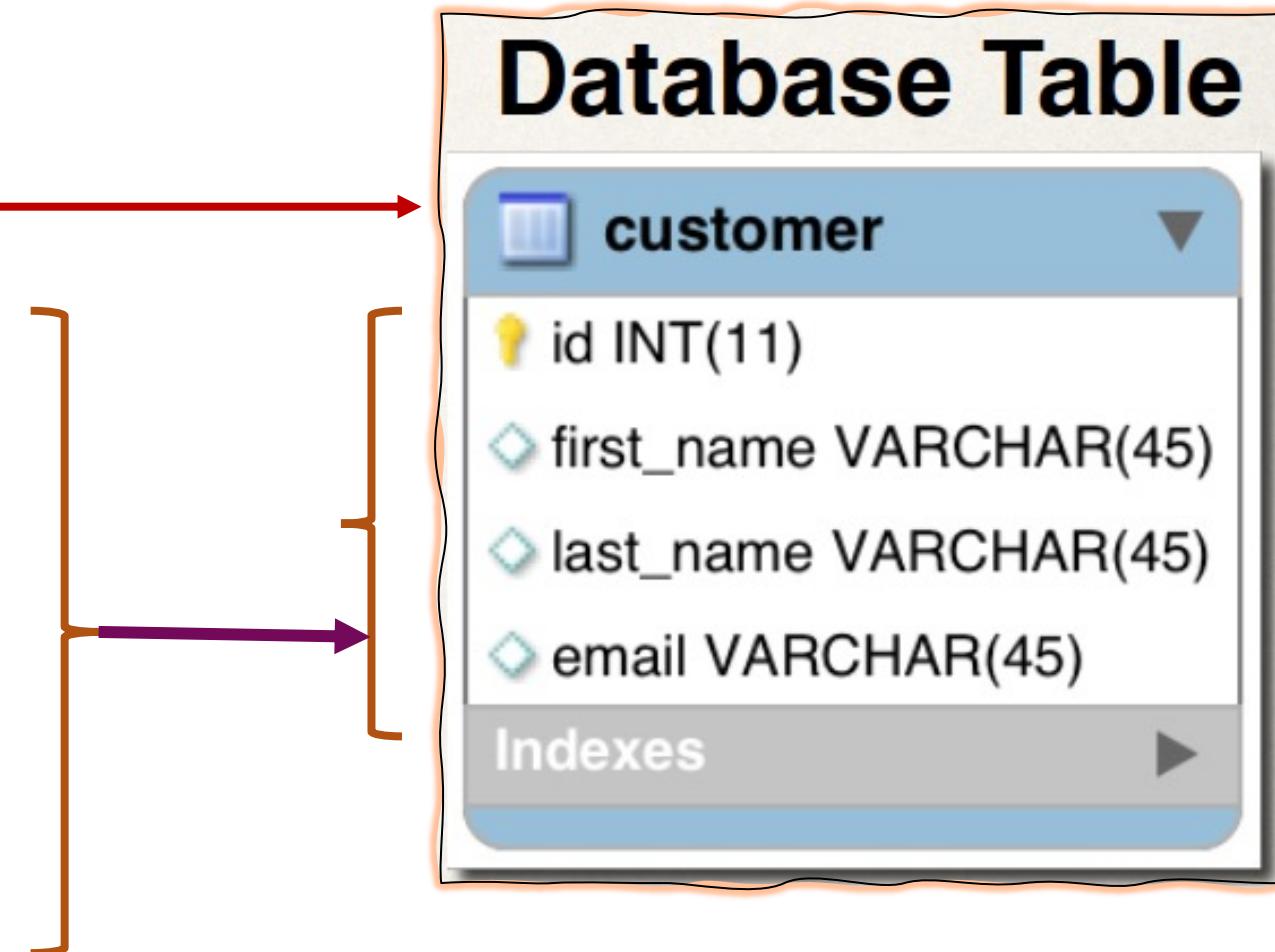


Step 1: Create Customer.java Mapping



File: com.se.springdemo.entity
Customer.java

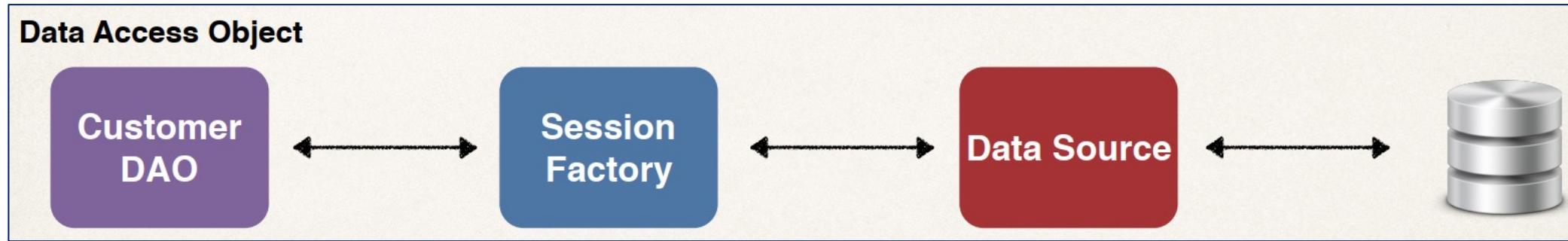
```
@Entity  
@Table(name="customer")  
public class Customer {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    @Column(name="id")  
    private int id;  
  
    @Column(name="first_name")  
    private String firstName;  
  
    @Column(name="last_name")  
    private String lastName;  
  
    @Column(name="email")  
    private String email;
```



Step 2: Create Customer Data Access Object (DAO)



- » For Hibernate, our **DAO** needs a **Hibernate SessionFactory**
- » **Hibernate Session Factory** needs a **Data Source** (defines database connection info)



These are all dependencies!

We will wire them together
with Dependency Injection (DI)

Step 2: Create Customer Data Access Object (DAO)

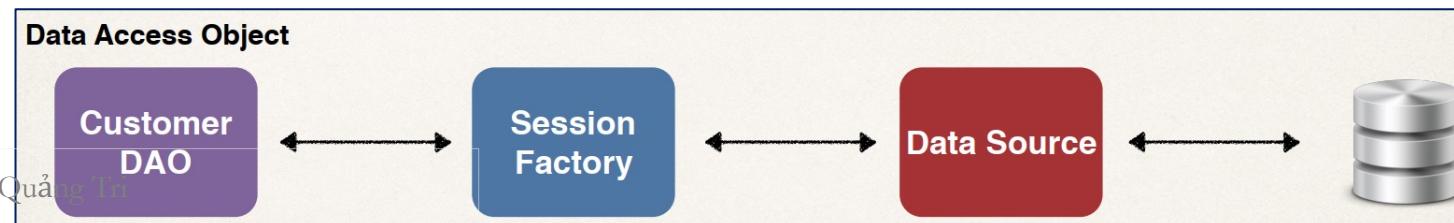


Data source

```
<bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"  
      destroy-method="close">  
    ...  
</bean>
```

Session Factory

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">  
  <property name="dataSource" ref="myDataSource" />  
  <property name="packagesToScan" value="com.se.springdemo.entity" />  
  ...  
</bean>
```

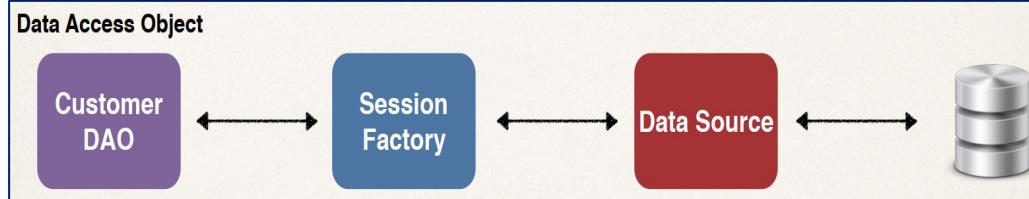


Step 2: Create Customer Data Access Object (DAO)



DAO Interface – CustomerDAO.java

```
package com.se.springdemo.dao;  
import ...3 lines  
public interface CustomerDAO {  
    public List<Customer> getCustomers();  
    public void saveCustomer(Customer theCustomer);  
    public Customer getCustomer(int theId);  
    public void deleteCustomer(int theId);  
}
```



CustomerDAO Implement- CustomerDAOImpl.java

```
@Repository  
public class CustomerDAOImpl implements CustomerDAO {  
    // need to inject the session factory  
    @Autowired  
    private SessionFactory sessionFactory;  
    @Transactional  
    public List<Customer> getCustomers() {  
        // get the current hibernate session  
        Session currentSession = sessionFactory.getCurrentSession();  
        // create a query ... sort by last name  
        Query<Customer> theQuery =  
            currentSession.createQuery("from Customer order by lastName",  
                Customer.class);  
        // execute query and get result list  
        List<Customer> customers = theQuery.getResultList();  
        // return the results  
        return customers;  
    }
```

Spring @Transaction:

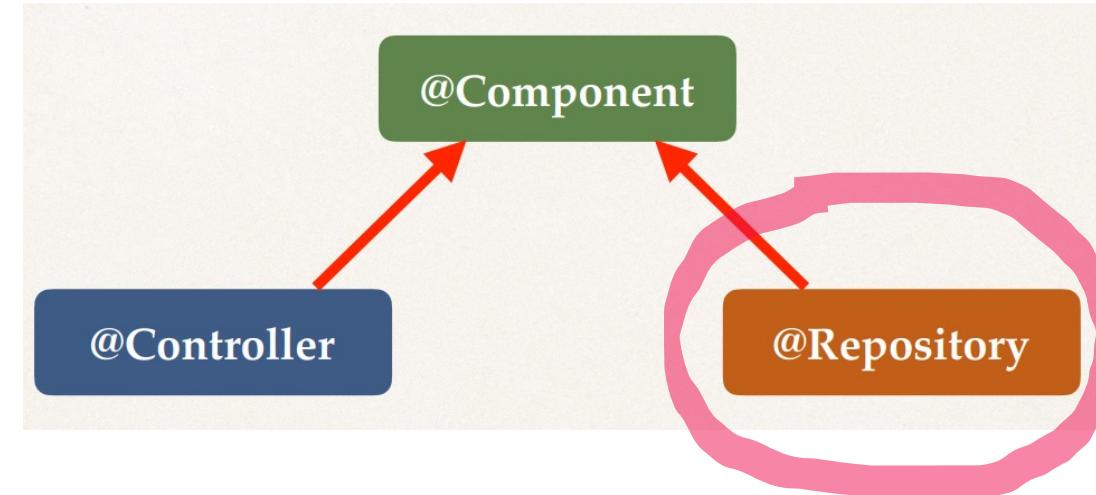
- » **Automagically** begin and end a transaction for your Hibernate code
- » **No need** for you to explicitly do this in your code

Step 2: Create Customer Data Access Object (DAO)

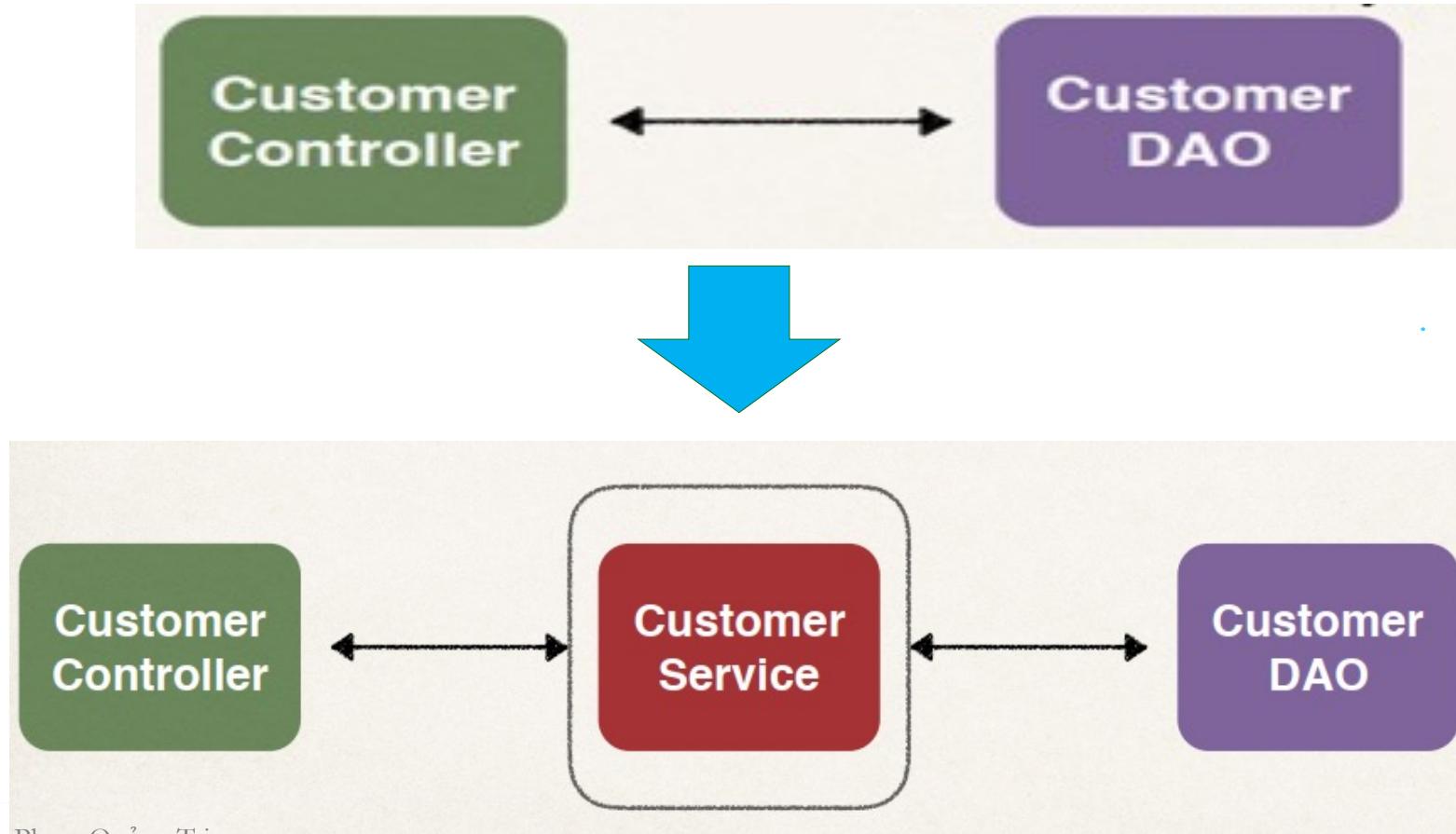


Specialized Annotation for DAOs:

- » Applied to DAO implementation
- » Spring will **automatically register** the **DAO implementation**
- » Spring also **provides translation** of any **JDBC related exceptions**

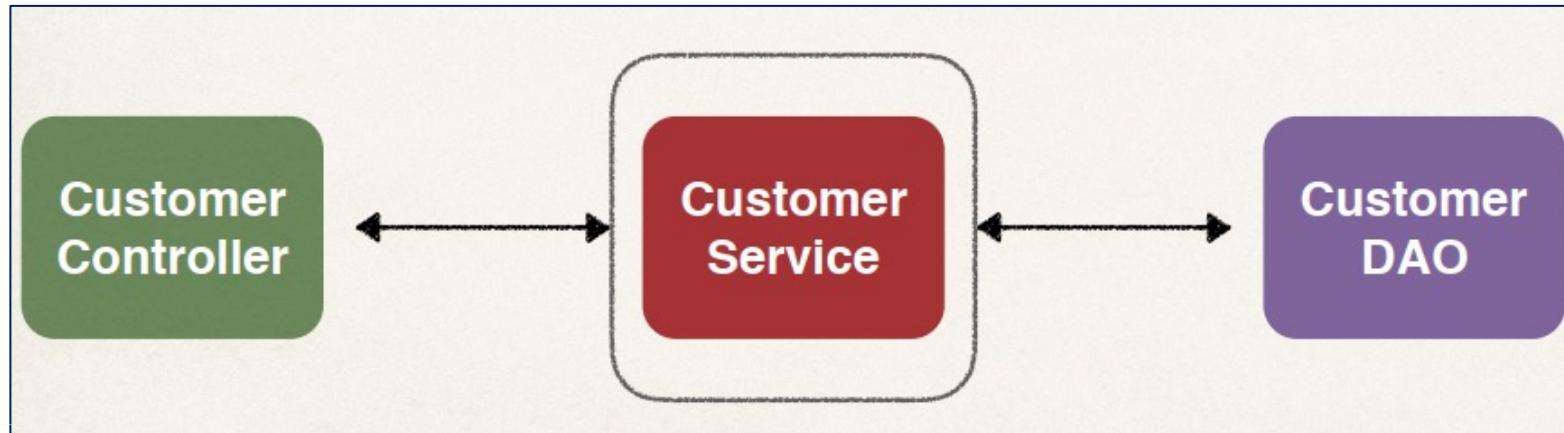


Step 3: Create CustomerController - Service Layer



Purpose of Service Layer:

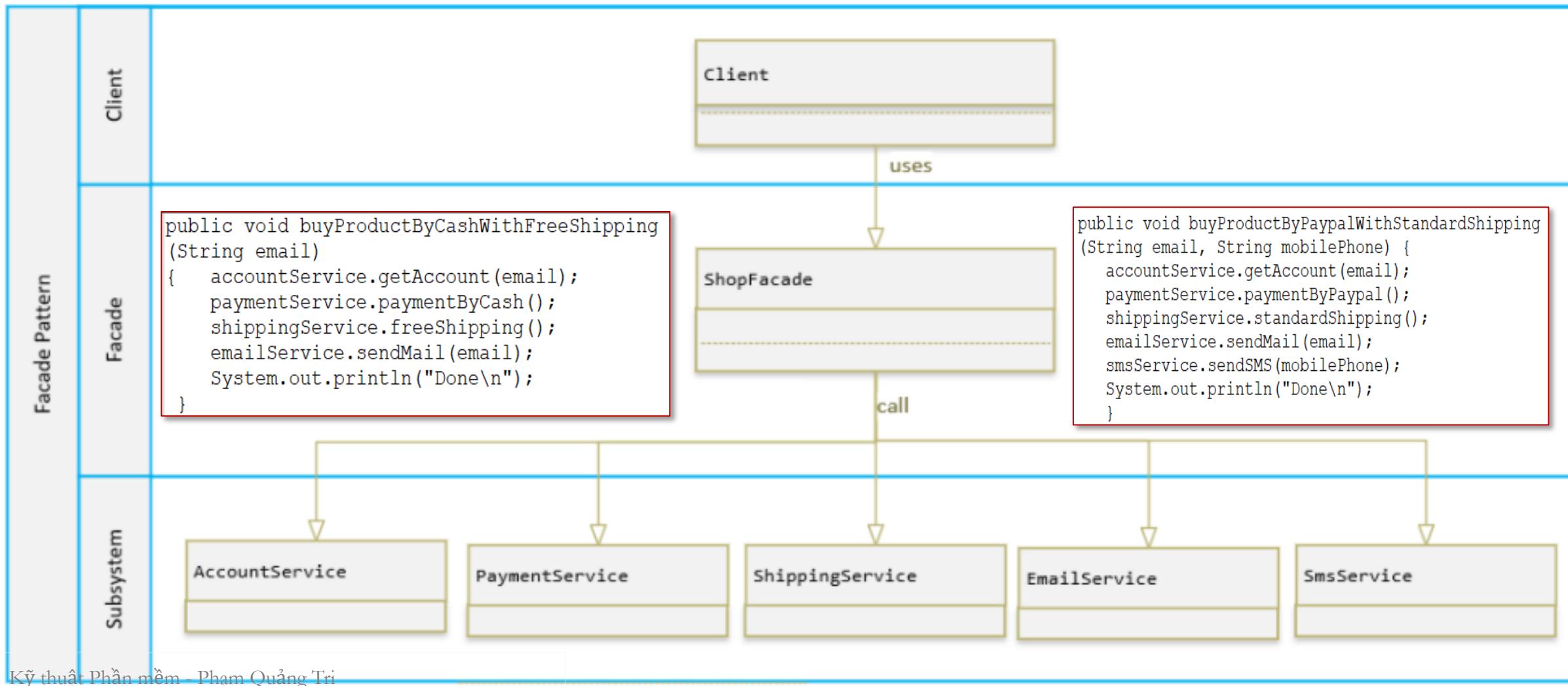
- » **Service Facade design pattern**
- » **Intermediate layer for custom business logic**
- » **Integrate data from multiple sources (DAO/repositories)**



Step 3: Create CustomerController - Service Layer



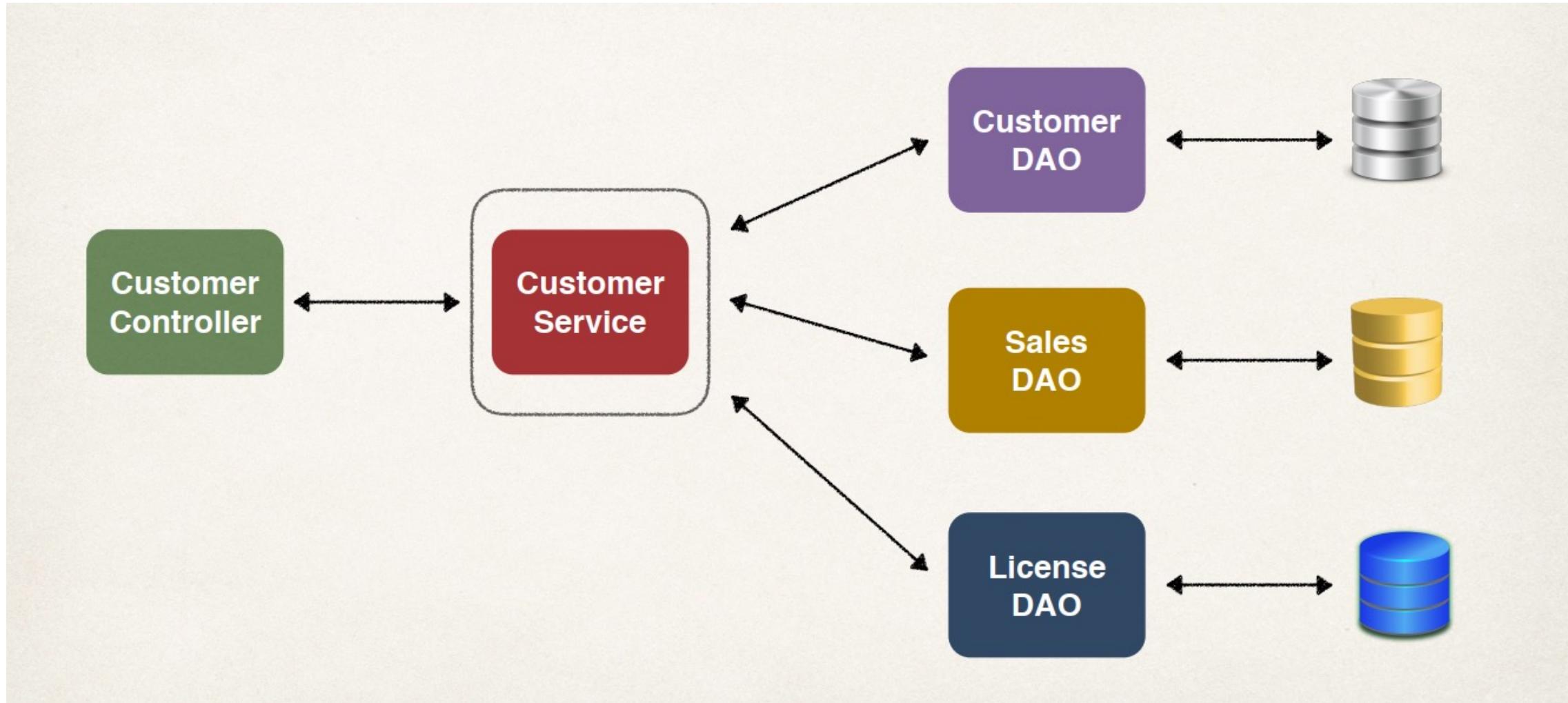
Facade design pattern:



Step 3: Create CustomerController - Service Layer

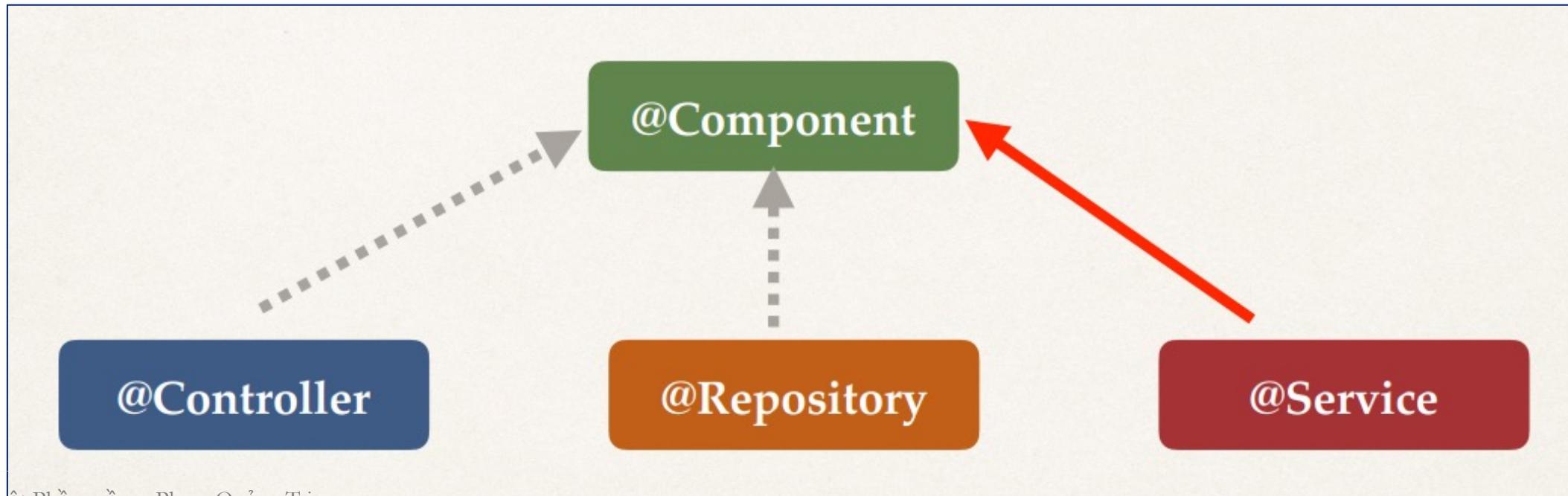


Integrate Multiple Data sources



Specialized Annotation for Services:

- » Applied to **Service implementation**
- » Spring will **automatically register** the **Services implementation**



Step 3: Create CustomerController - Service Layer



Interface CustomerService.java

```
public interface CustomerService {  
    public List<Customer> getCustomers();  
    public void saveCustomer(Customer theCustomer);  
    public Customer getCustomer(int theId);  
    public void deleteCustomer(int theId);  
}
```

Class CustomerController.java

```
@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
    @Autowired  
    // need to inject our customer service  
    private CustomerService customerService;  
    @GetMapping("/list")  
    public String listCustomers(Model theModel) {  
        // get customers from the service  
        List<Customer> theCustomers = customerService.getCustomers();  
        // add the customers to the model  
        theModel.addAttribute("customers", theCustomers);  
        return "list-customers";  
    }  
}
```

Class CustomerServiceImpl.java

```
@Service  
public class CustomerServiceImpl implements CustomerService {  
    // need to inject customer dao  
    @Autowired  
    private CustomerDAO customerDAO;  
    @Override  
    @Transactional  
    public List<Customer> getCustomers() {...3 lines}  
    @Override  
    @Transactional  
    public void saveCustomer(Customer theCustomer) {...4 lines}  
    @Override  
    @Transactional  
    public Customer getCustomer(int theId) {...4 lines}  
    @Override  
    @Transactional  
    public void deleteCustomer(int theId) {...4 lines}  
}
```

Step 4: Create view list-customer.jsp

File: list-customers.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
    <title>List Customers</title>
    <!-- reference our style sheet -->
    <link type="text/css"
          rel="stylesheet"
          href="${pageContext.request.contextPath}/resources/css/style.css" />
</head>
```

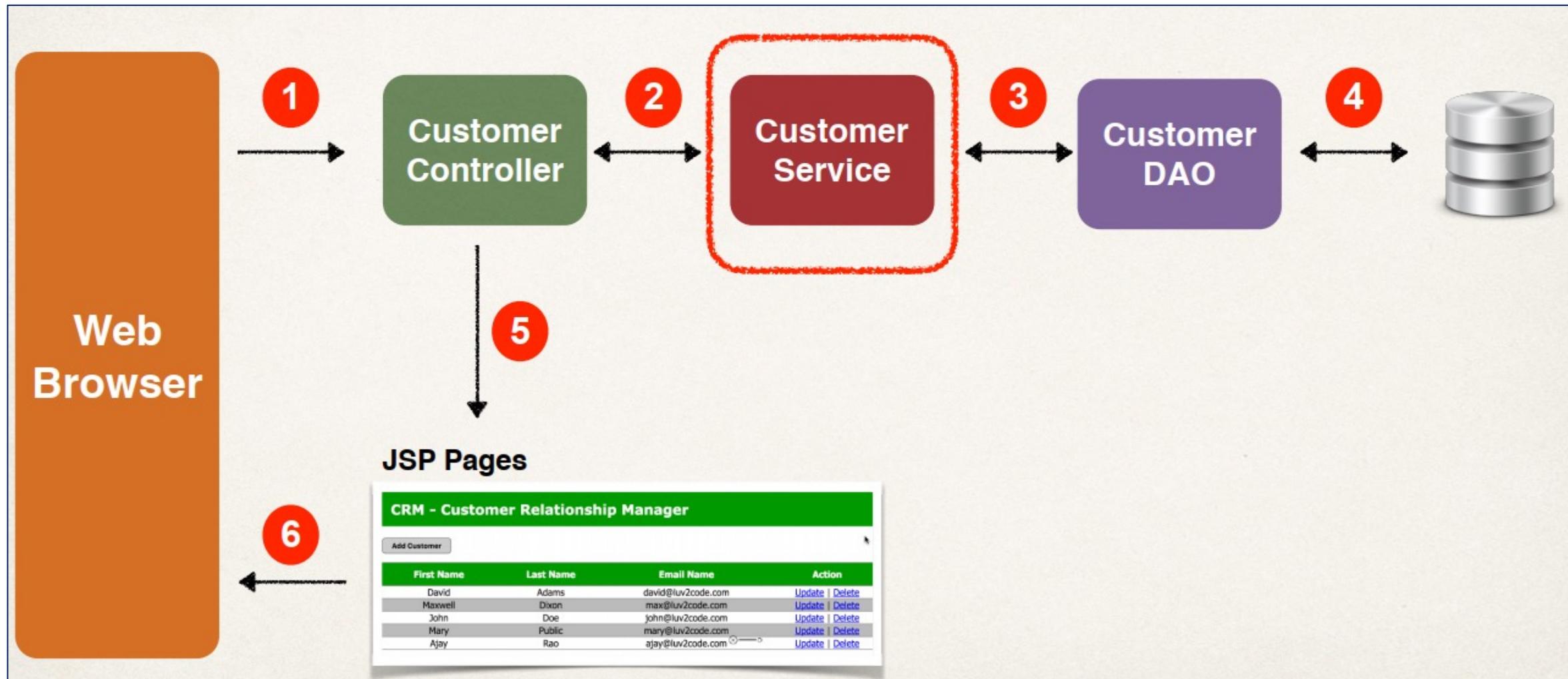
Step 4: Create view list-customer.jsp



File: list-customers.jsp

```
<c:forEach var="tempCustomer" items="${customers}">
    <!-- construct an "update" link with customer id -->
    <c:url var="updateLink" value="/customer/showFormForUpdate">
        <c:param name="customerId" value="${tempCustomer.id}" />
    </c:url>
    <!-- construct an "delete" link with customer id -->
    <c:url var="deleteLink" value="/customer/delete">
        <c:param name="customerId" value="${tempCustomer.id}" />
    </c:url>
    <tr>
        <td> ${tempCustomer.firstName} </td>
        <td> ${tempCustomer.lastName} </td>
        <td> ${tempCustomer.email} </td>
        <td>
            <!-- display the update link -->
            <a href="${updateLink}">Update</a>
            |
            <a href="${deleteLink}"
                onclick="if (!(confirm('Are you sure you want to delete this customer?'))) return false">Delete</a>
        </td>
    </tr>
</c:forEach>
```

Revised App Process

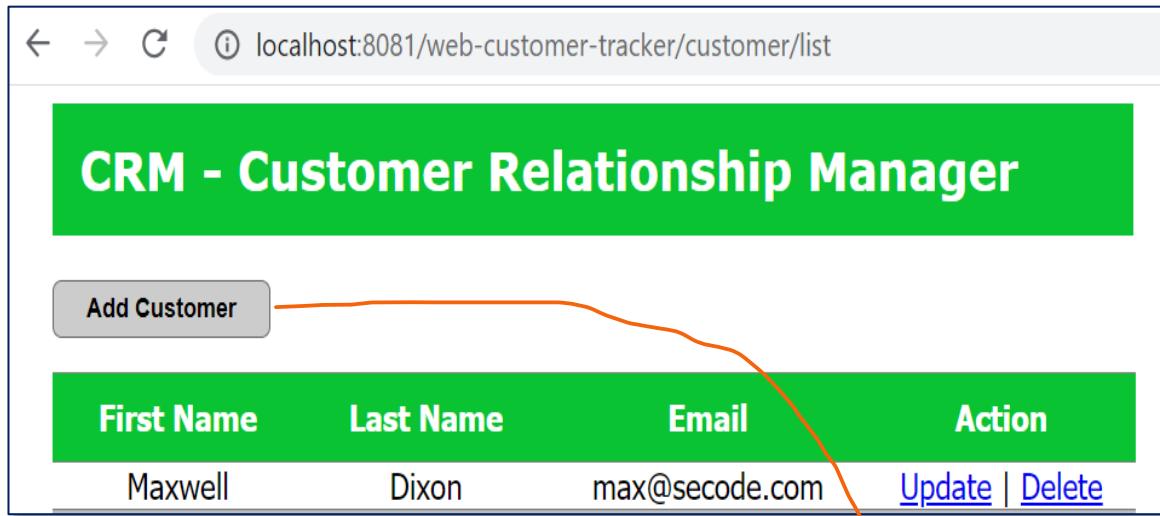




Add Customers

1. Create HTML form for new customer

2. Process Form Data: Controller -> Service -> DAO



A screenshot of a web browser displaying a table titled "CRM - Customer Relationship Manager". The table has columns for First Name, Last Name, Email, and Action. A row shows "Maxwell" in the First Name column, "Dixon" in the Last Name column, and "max@secode.com" in the Email column. In the Action column, there are "Update" and "Delete" links. Above the table, a grey button labeled "Add Customer" is visible. An orange arrow points from this button to the "Action" column of the table.

First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete

```
@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
    public String showFormForAdd(Model theModel) {  
        // create model attribute to bind form data  
        Customer theCustomer = new Customer();  
        theModel.addAttribute("customer", theCustomer);  
        return "customer-form";    }  
}
```

```
<input type="button" value="Add Customer"  
      onclick="window.location.href='showFormForAdd'; return false;"  
      <%-- window.location.href returns the href (URL) of the current page--%>  
      class="add-button" />
```

Step 1: Create HTML form for new customer (2)



```
<form:form action="saveCustomer" modelAttribute="customer" method="POST">
    <!-- need to associate this data with customer id -->
    <form:hidden path="id" />
    <table> <tbody>
        <tr>
            <td><label>First name:</label></td>
            <td><form:input path="firstName" /></td>
        </tr>
        <tr>
            <td><label>Last name:</label></td>
            <td><form:input path="lastName" /></td>
        </tr>
        <tr>
            <td><label>Email:</label></td>
            <td><form:input path="email" /></td>
        </tr>
        <tr>
            <td><label></label></td>
            <td><input type="submit" value="Save" class="save" /></td>
        </tr>
    </tbody> </table>
</form:form>
```

The screenshot shows a web browser window with the URL `localhost:8081/web-customer-tracker/customer/showFormForAdd`. The page title is "CRM - Customer Relationship Manager" and the section title is "Save Customer". There are three input fields: "First name" with value "David", "Last name" with value "Guetta", and "Email" with value "Guetta@secode.com". Below the inputs is a "Save" button, which is circled in orange. A purple arrow points from the "Save" button in the code to the "Save" button in the screenshot. A link "Back to List" is visible below the form.

```
@Controller
@RequestMapping("/customer")
public class CustomerController {
    @Autowired
    // need to inject our customer service
    private CustomerService customerService;
    @PostMapping("/saveCustomer")
    public String saveCustomer(@ModelAttribute("customer") Customer theCustomer) {
        // save the customer using our service
        customerService.saveCustomer(theCustomer);
        return "redirect:/customer/list";
    }
}
```

Step 2: Process Form Data Controller -> Service -> DAO



Interface CustomerService.java

```
public interface CustomerService {  
    public List<Customer> getCustomers();  
    public void saveCustomer(Customer theCustomer);  
    public Customer getCustomer(int theId);  
    public void deleteCustomer(int theId);  
}
```

Class CustomerServiceImpl.java

```
@Service  
public class CustomerServiceImpl implements CustomerService {  
    // need to inject customer dao  
    @Autowired  
    private CustomerDAO customerDAO;  
    @Override  
    @Transactional  
    public List<Customer> getCustomers() {...3 lines }  
    @Override  
    @Transactional  
    public void saveCustomer(Customer theCustomer) {...4 lines }  
    @Override  
    @Transactional  
    public Customer getCustomer(int theId) {...4 lines }  
    @Override  
    @Transactional  
    public void deleteCustomer(int theId) {...4 lines }  
}
```

Class CustomerController.java

```
@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
    @Autowired  
    // need to inject our customer service  
    private CustomerService customerService;  
    @GetMapping("/showFormForAdd")  
    public String showFormForAdd(Model theModel) {  
        // create model attribute to bind form data  
        Customer theCustomer = new Customer();  
        theModel.addAttribute("customer", theCustomer);  
        return "customer-form";  
    }  
    @PostMapping("/saveCustomer")  
    public String saveCustomer(@ModelAttribute("customer") Customer theCustomer) {  
        // save the customer using our service  
        customerService.saveCustomer(theCustomer);  
        return "redirect:/customer/list";  
    }  
    Kỹ thuật Phần mềm - Phạm Quảng Trí
```



Update Customers

» Prepopulate the form

» Process Form Data: Controller -> Service -> DAO



First Name	Last Name	Email	Action
Maxwell	Dixon	max@secode.com	Update Delete
John	Doe	john@commail.se	Update Delete

```
@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
    @GetMapping("/showFormForUpdate")  
    public String showFormForUpdate(@RequestParam("customerId") int theId,  
                                   Model theModel) {  
        // get the customer from our service  
        Customer theCustomer = customerService.getCustomer(theId);  
        // set customer as a model attribute to pre-populate the form  
        theModel.addAttribute("customer", theCustomer);  
        // send over to our form  
        return "customer-form";  
    }  
}
```

```
<c:forEach var="tempCustomer" items="${customers}">  
    <!-- construct an "update" link with customer id -->  
    <c:url var="updateLink" value="/customer/showFormForUpdate">  
        <c:param name="customerId" value="${tempCustomer.id}" />  
    </c:url>
```

Update Customer – Prepopulate the form



```
@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
    @GetMapping("/showFormForUpdate")  
    public String showFormForUpdate(@RequestParam("customerId") int theId,  
                                   Model theModel) {  
        // get the customer from our service  
        Customer theCustomer = customerService.getCustomer(theId);  
        // set customer as a model attribute to pre-populate the form  
        theModel.addAttribute("customer", theCustomer);  
        // send over to our form  
        return "customer-form";    }  
}
```

The screenshot shows a web browser window with the URL `localhost:8081/web-customer-tracker/customer/showFormForAdd`. The page title is **CRM - Customer Relationship Manager**. The form is titled **Save Customer**. It contains three input fields: **First name:** David, **Last name:** Gueta, and **Email:** Gueta@secode.com. Below the form is a **Save** button and a [Back to List](#) link.

Update Customer – Process form Data



localhost:8081/web-customer-tracker/customer/showFormForAdd

CRM - Customer Relationship Manager

Save Customer

First name:

Last name:

Email:

Save

[Back to List](#)

```
@Controller  
@RequestMapping("/customer")  
public class CustomerController {  
    @PostMapping("/saveCustomer")  
    public String saveCustomer(@ModelAttribute("customer") Customer theCustomer) {  
        // save the customer using our service  
        customerService.saveCustomer(theCustomer);  
        return "redirect:/customer/list";  
    }  
}
```

```
@Service  
public class CustomerServiceImpl implements CustomerService {  
    // need to inject customer dao  
    @Autowired  
    private CustomerDAO customerDAO;  
    @Override  
    @Transactional  
    public List<Customer> getCustomers() {...3 lines...}  
    @Override  
    @Transactional  
    public void saveCustomer(Customer theCustomer) {...4 lines...}  
    @Override  
    @Transactional  
    public Customer getCustomer(int theId) {...4 lines...}  
    @Override  
    @Transactional  
    public void deleteCustomer(int theId) {...4 lines...}  
}
```

```
public interface CustomerService {  
    public List<Customer> getCustomers();  
    public void saveCustomer(Customer theCustomer);  
    public Customer getCustomer(int theId);  
    public void deleteCustomer(int theId);  
}
```



Questions