What you will learn in Topic 4

Azzeddine
RIGAT

Topic 4: Build a Database Web App

- Part 1 - Setup phase
- Part 2 - MVC phase
- Part 3 - Refactoring and Services
- Part 4 - Add to Database
- Part 5 - Update Database
- Part 6 - Delete from Database

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Part 1 - Setup phase

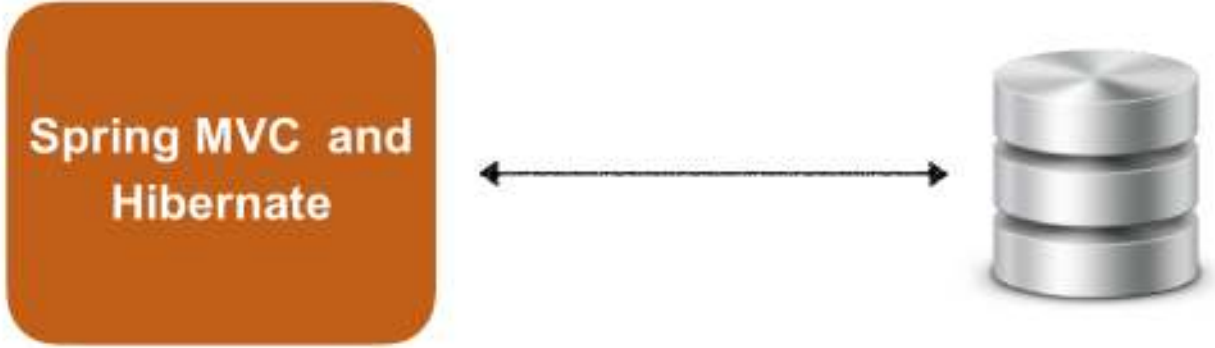
Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Create a Database Application

Class Project

Full working *Spring MVC and Hibernate* **application that connects to a database**



Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

To Do List

1. Create Eclipse Project
2. Download Hibernate Files
3. Download MySQL JDBC Driver
4. Add JAR files to Eclipse Project … Build Path

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Customer Relationship Management - CRM

- Set up Database Dev Environment
- List Customers
- Add a new Customer
- Update a Customer
- Delete a Customer

Step by Step

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Customer Relationship Management - CRM

A Demo

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Database scripts

Folder: sql-scripts

1. create-user.sql

2. customer-tracker.sql

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Database scripts

About:01-create-user.sql

1.Create a new MySQL user for our application

1. user id: **springstudent**
2. password: **springstudent**

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Database scripts

About: 02-customer-tracker.sql

1. Create a new database table: **customer**
2. Load table with sample data

Check **00-sql-scripts**

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Project - Test DB Connection

Test DB Connection

1. Set up our Eclipse project
2. Add JDBC Driver for MySQL
3. Sanity test … make sure we can connect :-)

Step by Step

Check **01-CRUD-CRM-setup-test**

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Dev Environment

1. Copy starter config files
   a. Web.xml and spring config
2. Copy over JSTL libs
3. Copy latest Spring JAR files
4. Copy latest Hibernate JAR files

Step by Step

Check **02-CRUD-CRM-setup-test**

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Project - Test DB Connection

Configuration for Spring + Hibernate

1. Define database dataSource / connection pool
2. Setup Hibernate session factory
3. Setup Hibernate transaction manager
4. Enable configuration of transactional annotations

Step by Step

Check **02-CRUD-CRM-setup-test**

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Setup Project - Test DB Connection

Placement of Configurations

Add the following configurations in your Spring MVC configuration file

For our example
spring-mvc-crud-demo-servlet.xml

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Step 1: Define database dataSource / connection pool

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

  <property name="driverClass" value="com.mysql.jdbc.Driver" />
  <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/web_customer_tracker?useSSL=false" />

  <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>
```

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

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.javaweb.springdemo.entity" />

        <property name="hibernateProperties">
            </props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
            </props>
        </property>

</bean>
```

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Step 3: Setup Hibernate transaction manager

```
<bean id="myTransactionManager"
            class="org.springframework.orm.hibernate5.HibernateTransactionManager">

      <property name="sessionFactory" ref="sessionFactory"/>

</bean>
```

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Step 4: Enable configuration of transactional annotations

```
<tx:annotation-driven transaction-manager="myTransactionManager" />
```

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Recap Configuration for Spring + Hibernate

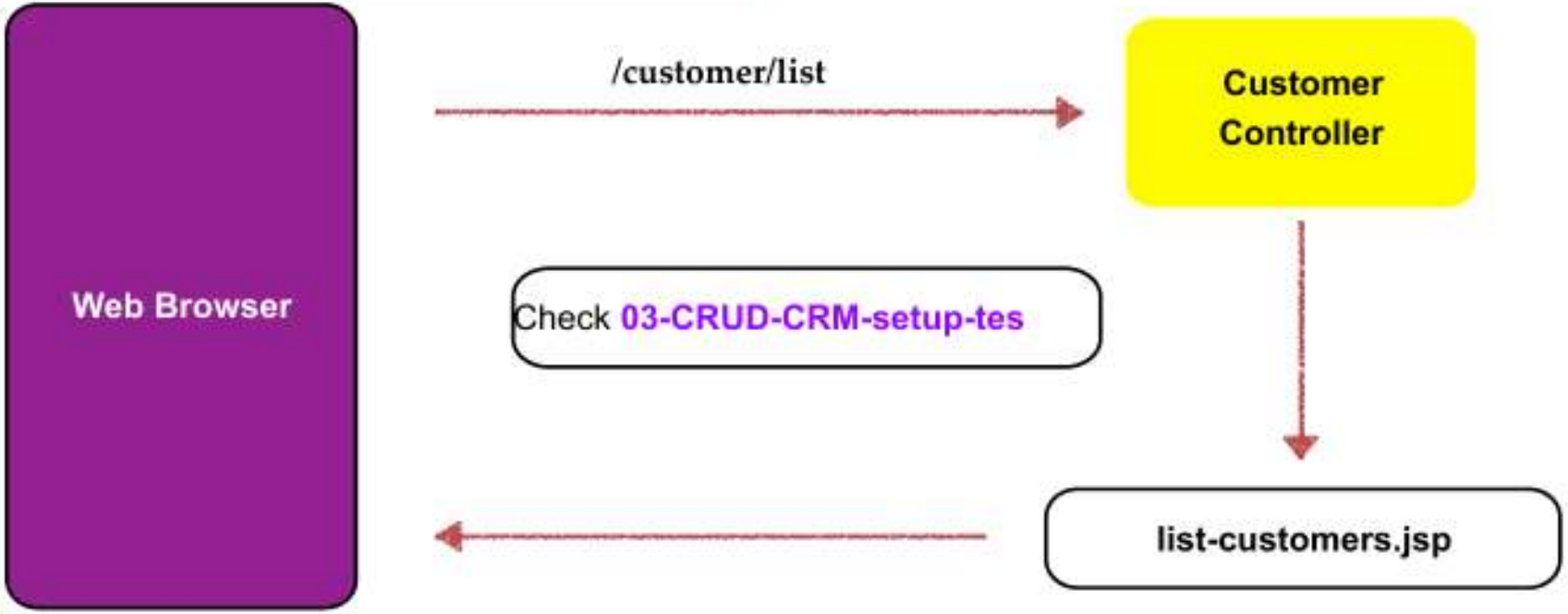
1. Define database dataSource / connection pool
2. Setup Hibernate session factory
3. Setup Hibernate transaction manager
4. Enable configuration of transactional annotations

Topic 4, Part 1 - Setup phase

Azzeddine
RIGAT

Test Basic Spring MVC Controller

Customer Controller



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Part 2 - MVC phase

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Sample App Architecture

Configuration for Spring + Hibernate

- List Customers
- Add a new Customer
- Update a Customer
- Delete a Customer

CRM - Customer Relationship Manager

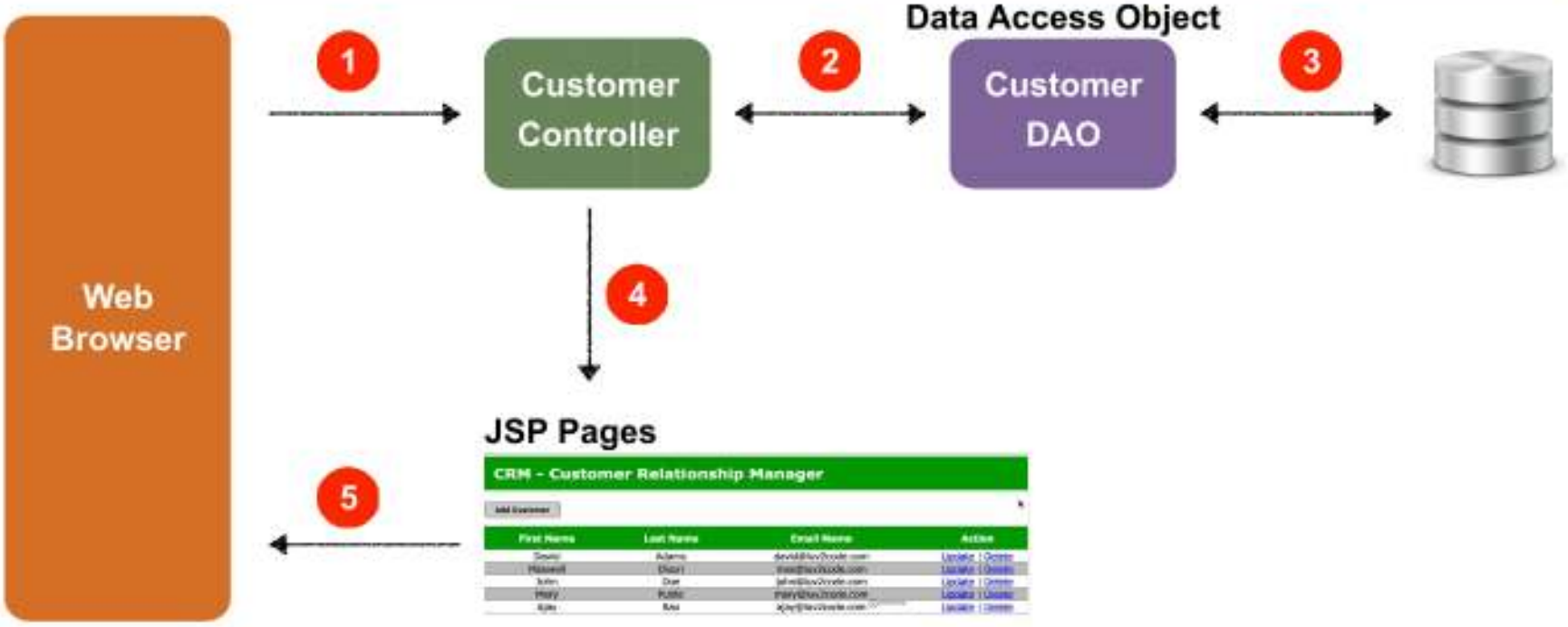
Add Customer

First Name	Last Name	Email Name
David	Adams	david@luv2code.com
Maxwell	Dixon	max@luv2code.com
John	Doe	john@luv2code.com
Mary	Public	mary@luv2code.com
Ajay	Rao	ajay@luv2code.com

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Big Picture



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Customer Data Access Object

- Responsible for interfacing with the database
- This is a common design pattern: **Data Access Object (DAO)**

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Customer Data Access Object

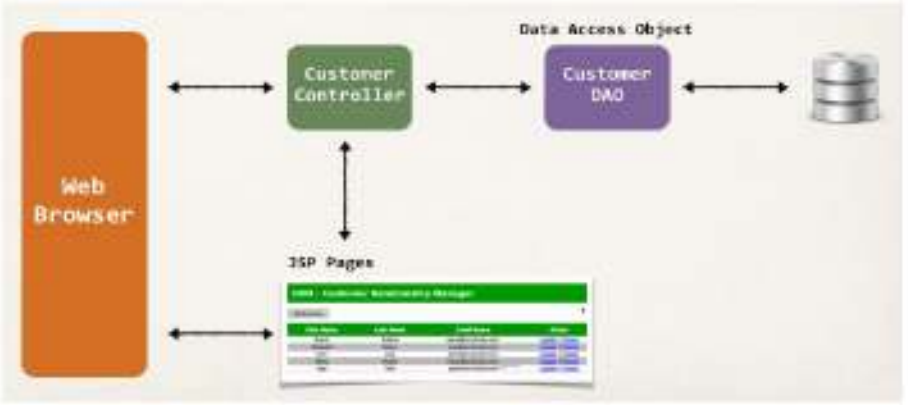
Methods
saveCustomer(…)
getCustomer(…)
getCustomers()
updateCustomer(…)
deleteCustomer(…)

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

List Customers - Dev Process

1. Create **Customer.java**
2. Create **CustomerDAO.java** and **CustomerDAOImpl.java**
3. Create **CustomerController.java**
4. Create JSP page: **list-customers.jsp**



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Hibernate Terminology - Refresh

Entity Class

Java class that is mapped to a database table

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Step 1: Map class to database table

```
@Entity
@Table(name="customer")
public class Customer {
}
```

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Step 2: Map fields to database columns

```
@Entity
@Table(name="customer")
public class Customer {

        @Id
        @Column(name="id")
        private int id;

        @Column(name="first_name")
        private String firstName;
```

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Entity Scanning

Remember our Spring MVC config file?

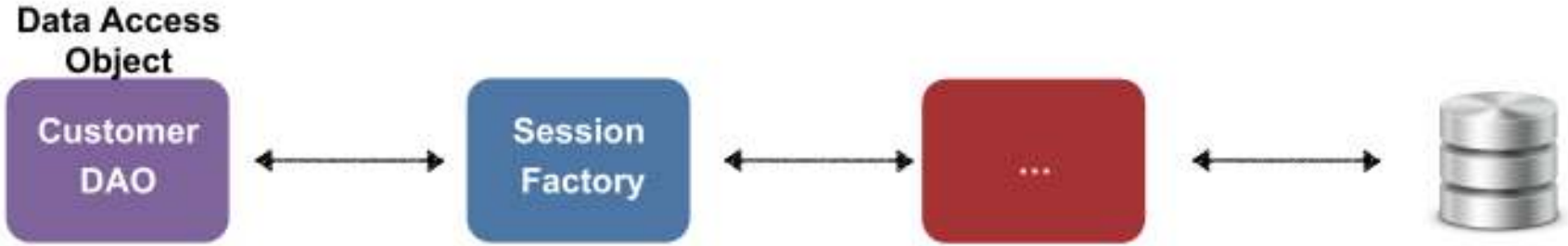
```
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

      <property name="dataSource" ref="myDataSource" />
      <property name="packagesToScan" value="javaweb.spring.entity" />
      ...
</bean>
```

Azzeddine RIGAT

Define Data Access Object

Customer Data Access Object

For Hibernate, our DAO needs a Hibernate SessionFactory



Topic 4, Part 2 - MVC phase

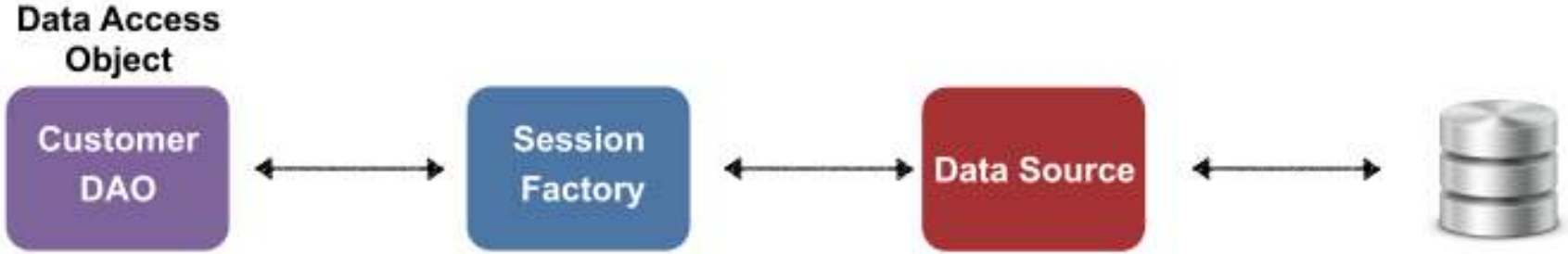
Azzeddine
RIGAT

Define Data Access Object

Hibernate Session Factory

Our Hibernate Session Factory needs a Data Source

- The data source defines database connection info



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Dependencies

These are all dependencies!

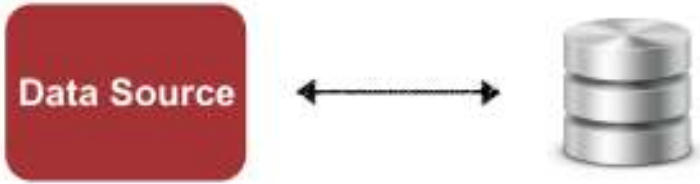
We will wire them together with Dependency Injection (DI)

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Data Source

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

  <property name="driverClass" value="com.mysql.jdbc.Driver" />
  <property name="jdbcUrl"
      value="jdbc:mysql://localhost:3306/web_customer_tracker?useSSL=false" />

          … user id, password etc …
</bean>
```



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Session Factory

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

    <property name="dataSource" ref="myDataSource" />

    ...

</bean>
```

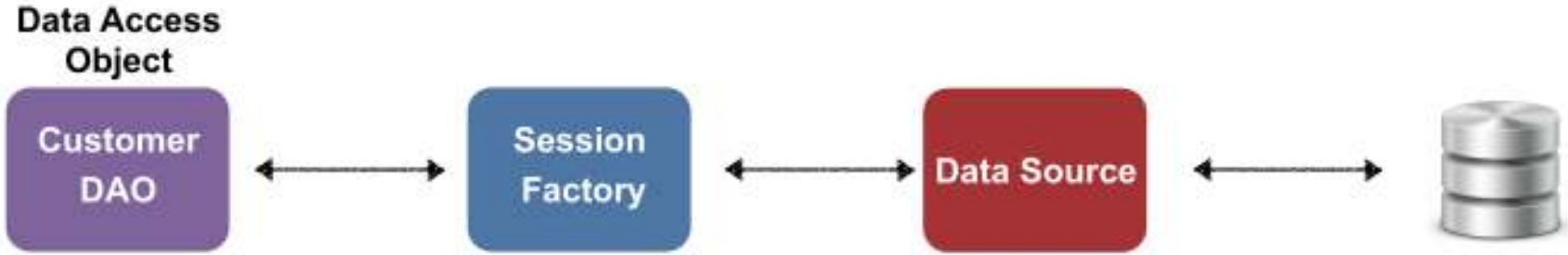
Topic 4, Part 2 - MVC phase

Azzeddine RIGAT

Customer DAO

1. Define DAO interface
2. Define DAO implementation
   a. Inject the session factory

Step by Step



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Step 1: Define DAO interface

```
public interface CustomerDAO {

    public List<Customer> getCustomers();

}
```

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Step 2: Define DAO implementation

```
public class CustomerDAOImpl implements CustomerDAO {

    @Autowired
    private SessionFactory sessionFactory;

    public List<Customer> getCustomers() {
        ...
    }

}
```

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Spring @Transactional

- Spring provides an **@Transactional** annotation
- **Automagically** begin and end a transaction for your Hibernate code
  - No need for you to explicitly do this in your code
- This Spring **magic** happens behind the scenes

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Flash back - Standalone Hibernate code

```
// start a transaction  session.beginTransaction();


// DO YOUR HIBERNATE STUFF HERE
// ...


// commit transaction  session.getTransaction().commit();
```

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

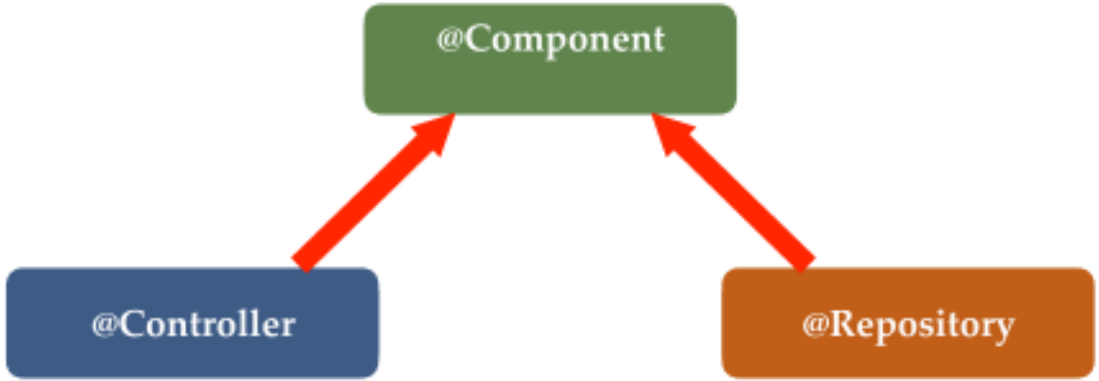
Spring @Transactional Magic

```
@Transactional
public List<Customer> getCustomers() {

    // get the current hibernate session
    Session currentSession = sessionFactory.getCurrentSession();

    // create a query
    Query<Customer> theQuery =
                        currentSession.createQuery("from Customer", Customer.class);

    // get the result list
    List<Customer> customers = theQuery.getResultList();

    return customers;
}
```

Azzeddine RIGAT

Specialized Annotation for DAOs

- Spring provides the **@Repository** annotation



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Specialized Annotation for DAOs

- Applied to DAO implementations
- Spring will automatically register the DAO implementation
  - thanks to component-scanning
- Spring also provides translation of any JDBC related exceptions

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Updates for the DAO implementation

```
@Repository
public class CustomerDAOImpl implements CustomerDAO {

    @Autowired
    private SessionFactory sessionFactory;

  @Transactional
    public List<Customer> getCustomers() {
        …
    }

}
```

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

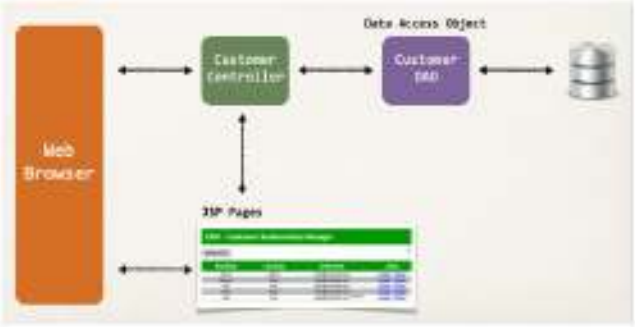
Inject DAO into Controller

List Customers

- Create **Customer.java**
- Create **CustomerDAO.java** and **CustomerDAOImpl.java**
- Create **CustomerController.java**

4. Create JSP page: **list-customers.jsp**

Check **04-CRUD-CRM-list-customers**



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

First Version - Plain

CRM - Customer Relationship Manager

First Name	Last Name	Email
David	Adams	david@luv2code.com
John	Doe	john@luv2code.com
Ajay	Rao	ajay@luv2code.com
Mary	Public	mary@luv2code.com
Maxwell	Dixon	max@luv2code.com

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Apply CSS and make it Pretty

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email Name
David	Adams	david@luv2code.com
Maxwell	Dixon	max@luv2code.com
John	Doe	john@luv2code.com
Mary	Public	mary@luv2code.com
Ajay	Rao	ajay@luv2code.com

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Development Process

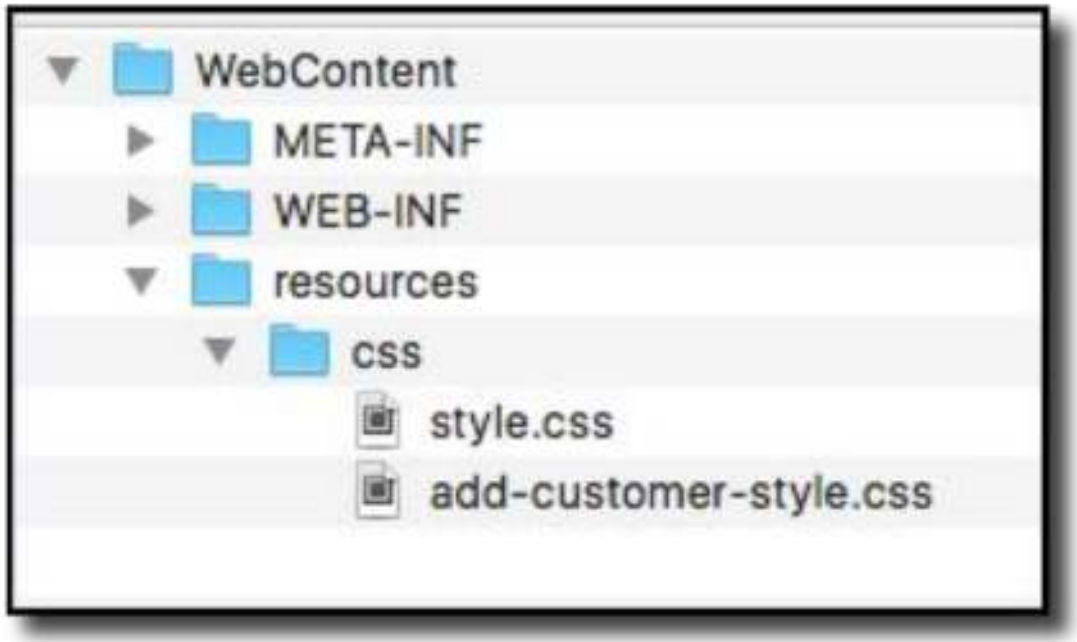
1. Place CSS in a ‘resources’ directory
2. Configure Spring to serve up ‘resources’ directory
3. Reference CSS in your JSP

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Step 1: Place CSS in 'resources' directory



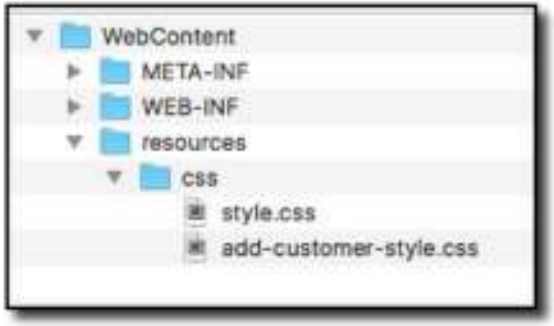
Azzeddine RIGAT

Create JSP View Page

Step 2: Configure Spring to serve up 'resources' directory

File: spring-mvc-crud-servlet.xml

```
<mvc:resources location="/resources/" mapping="/resources/**" />
```



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Step 3: Reference CSS in your JSP

File: list-customers.jsp

```
<head>
    <title>List Customers</title>
    <link type="text/css"  rel="stylesheet"
          href="${pageContext.request.contextPath}/resources/css/style.css">

</head>
```

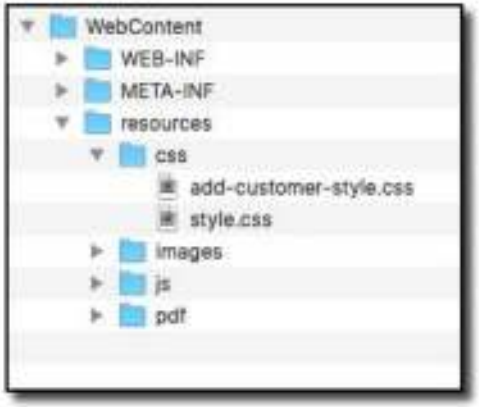
Check **05-CRUD-CRM-list-customers**

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Applies for JavaScript, images, pdfs etc…

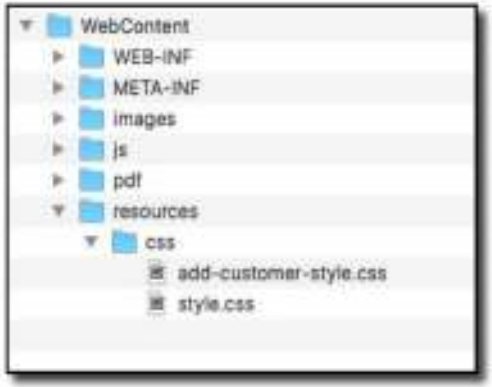


Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Alternate Directory Structure



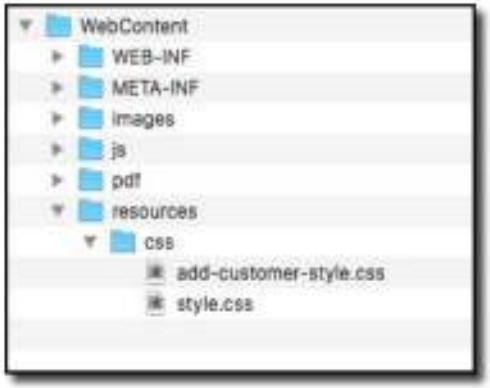
Azzeddine RIGAT

Create JSP View Page

Alternate Directory Structure

File: spring-mvc-crud-demo-servlet.xml

```
<mvc:resources location="/resources/" mapping="/resources/**" />
<mvc:resources location="/images/" mapping="/images/**" />
<mvc:resources location="/js/" mapping="/js/**" />
<mvc:resources location="/pdf/" mapping="/pdf/**" />
```



Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Adding a Welcome File

This is our “welcome page” ???

Topic 4, Part 2 - MVC phase

Azzeddine
RIGAT

Create JSP View Page

Adding a Welcome File

http://localhost:8080/web-customer-tracker

- Server will look for a welcome file
- If it doesn't find one, then you'll get 404 :-(
- Welcome files are configured in web.xml

Check **06-CRUD-CRM-list-customers**

Topic 4, Part 3 - Refactoring and Services

Azzeddine
RIGAT

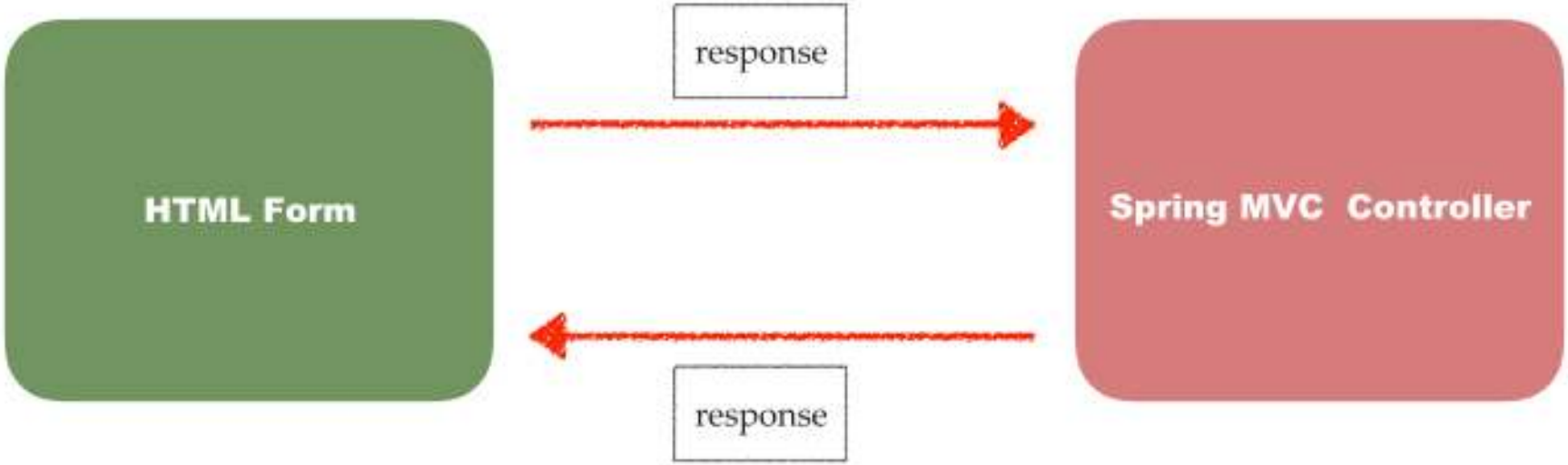
Part 3 - Refactoring and Services

Topic 4, Part 3 - Refactoring and Services

Azzeddine
RIGAT

@GetMapping and @PostMapping

HTTP Request / Response



Topic 4, Part 3 - Refactoring and Services

Azzeddine
RIGAT

@GetMapping and @PostMapping

Most Commonly Used HTTP Methods

Method	Description
GET	Requests data from given resource
POST	Submits data to given resource
others...	

@GetMapping and @PostMapping

Sending Data with GET method

```
<form action="processForm" method="GET" ...>
    ...
</form>
```

- Form data is added to end of URL as name/value pairs
  - **theUrl?field1=value1&field2=value2…**

Azzeddine
RIGAT

@GetMapping and @PostMapping

Handling Form Submission

```
@RequestMapping("/processForm")
public String processForm(...) {
        ...
}
```

- This mapping handles ALL HTTP methods
- **GET, POST, etc …**

@GetMapping and @PostMapping

Constrain the Request Mapping - GET

```
@RequestMapping(path="/processForm", method=RequestMethod.GET)
public String processForm(...) {
        ...
}
```

- This mapping **ONLY** handles **GET** method
- Any other HTTP REQUEST method will get rejected

@GetMapping and @PostMapping

New Annotation Short-Cut

```
@GetMapping("/processForm")
public String processForm(...) {
        ...
}
```

- This mapping **ONLY** handles **GET** method
- Any other HTTP REQUEST method will get rejected
- New annotation: **@GetMapping**

@GetMapping and @PostMapping

Sending Data with POST method

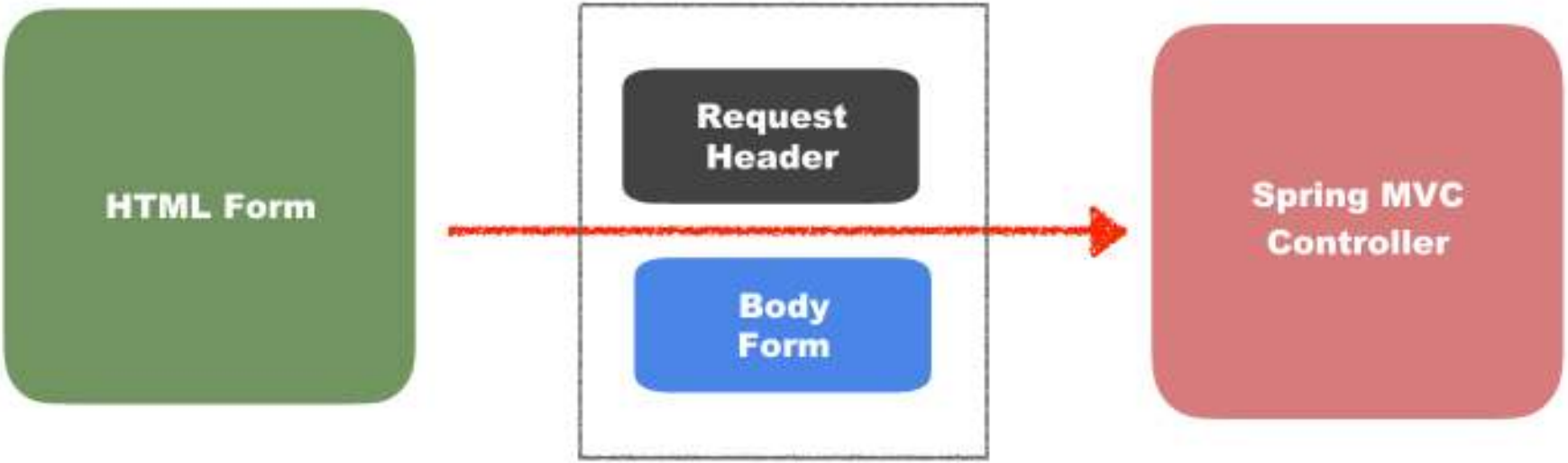
```
<form action="processForm" method="POST" ...>
    ...
</form>
```

- Form data is passed in the body of HTTP request message

Azzeddine
RIGAT

@GetMapping and @PostMapping

Sending Data with POST method



@GetMapping and @PostMapping

Constrain the Request Mapping - POST

```
@RequestMapping(path="/processForm", method=RequestMethod.POST)
public String processForm(...) {
        ...
    }
```

- This mapping **ONLY** handles **POST** method
- Any other HTTP REQUEST method will get rejected

@GetMapping and @PostMapping

New Annotation Short-Cut

```
@PostMapping("/processForm")
public String processForm(...) {
        ...
}
```

- This mapping **ONLY** handles **POST** method
- Any other HTTP REQUEST method will get rejected
- New annotation: **@PostMapping**

Azzeddine RIGAT

@GetMapping and @PostMapping

Well .... which one???

GET

- Good for debugging
- Bookmark or email URL
- Limitations on data length

POST

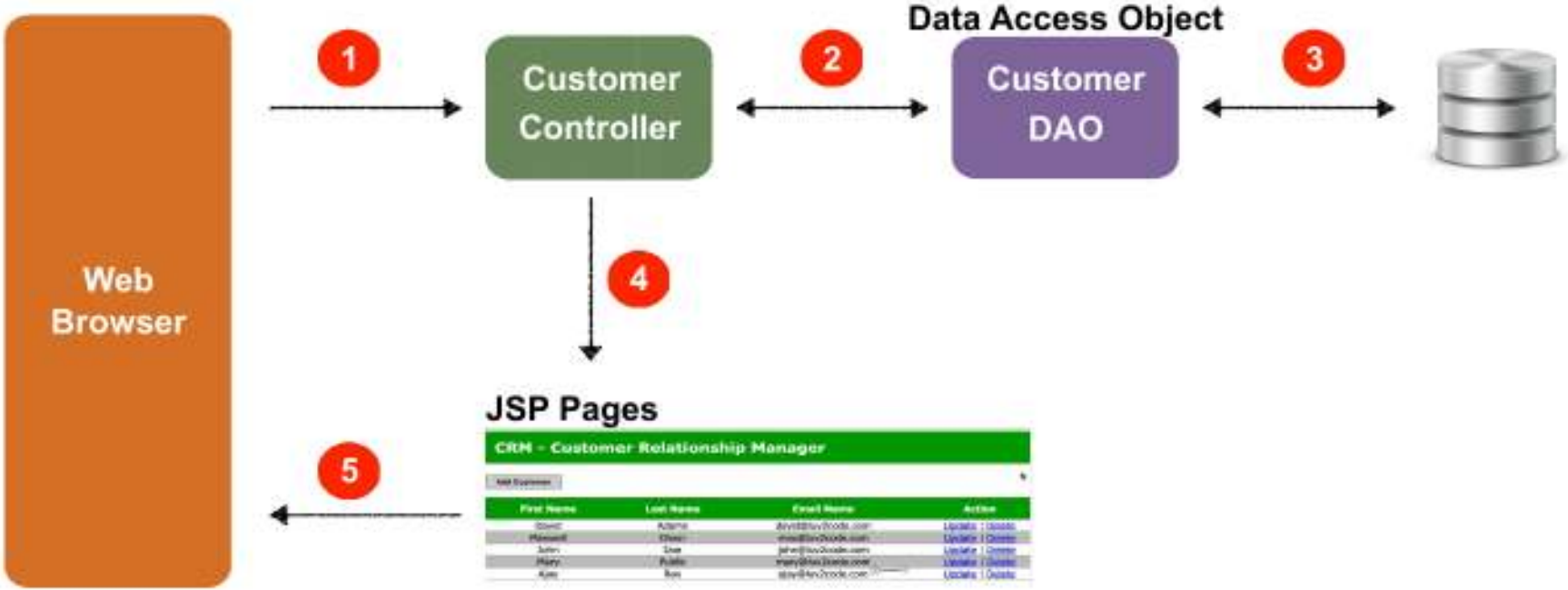
- Can't bookmark or email URL
- No limitations on data length
- Can also send binary data

Check **07-CRUD-CRM-refactor-add-get-post-mapping**

Azzeddine
RIGAT

Define Services with @Service

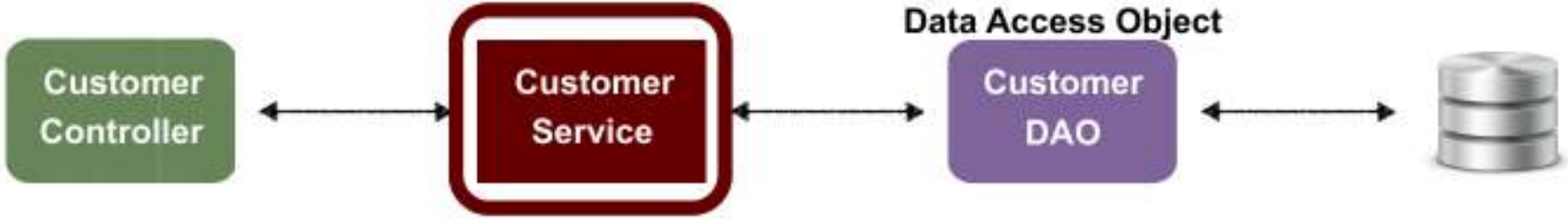
Big Picture



Azzeddine
RIGAT

Define Services with @Service

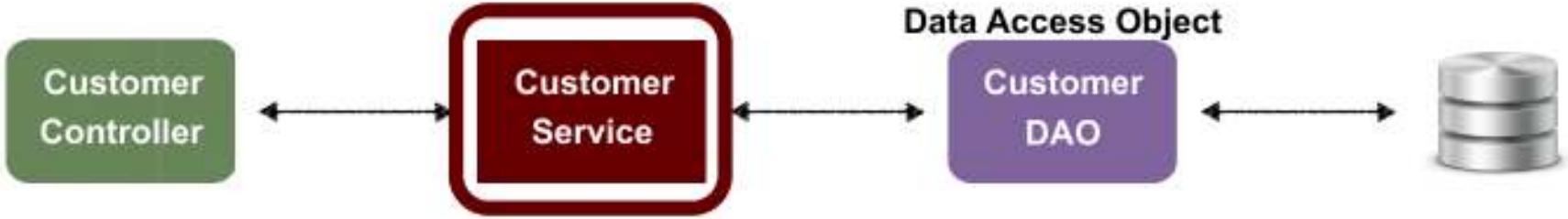
Refactor: Add a Service Layer



Define Services with @Service

Purpose of Service Layer

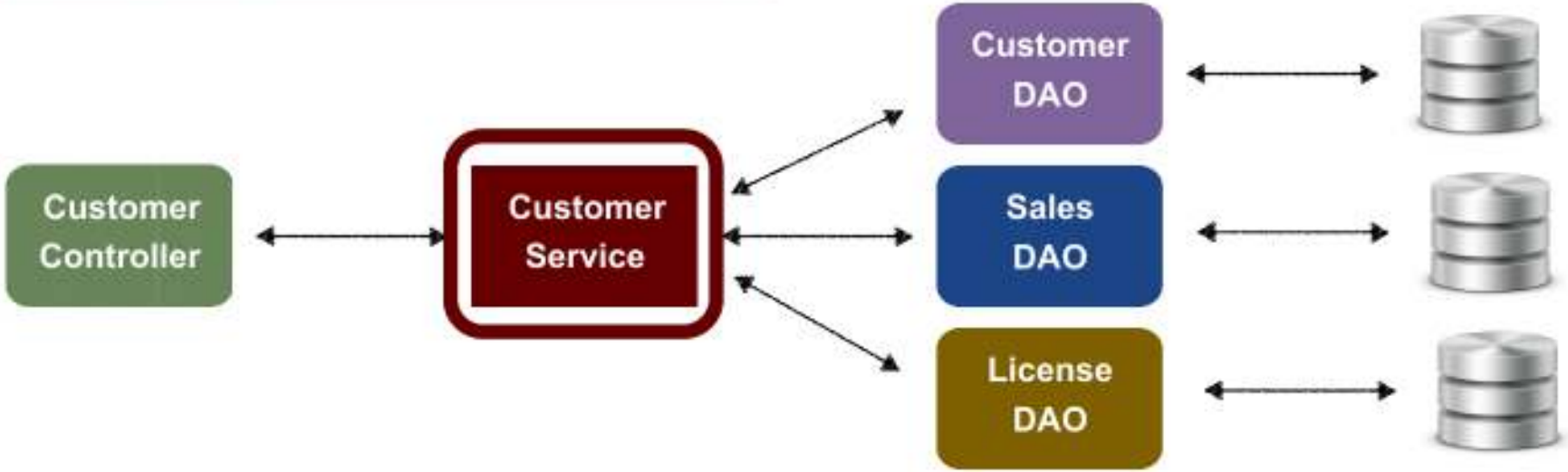
- **Service Facade** design pattern
- Intermediate layer for custom business logic
- Integrate data from multiple sources (DAO/repositories)



Azzeddine
RIGAT

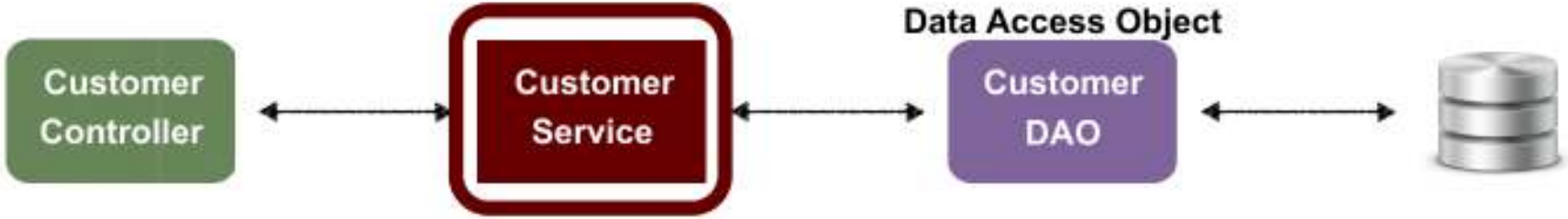
Define Services with @Service

Integrate Multiple Data Sources



Define Services with @Service

Most Times - Delegate Calls

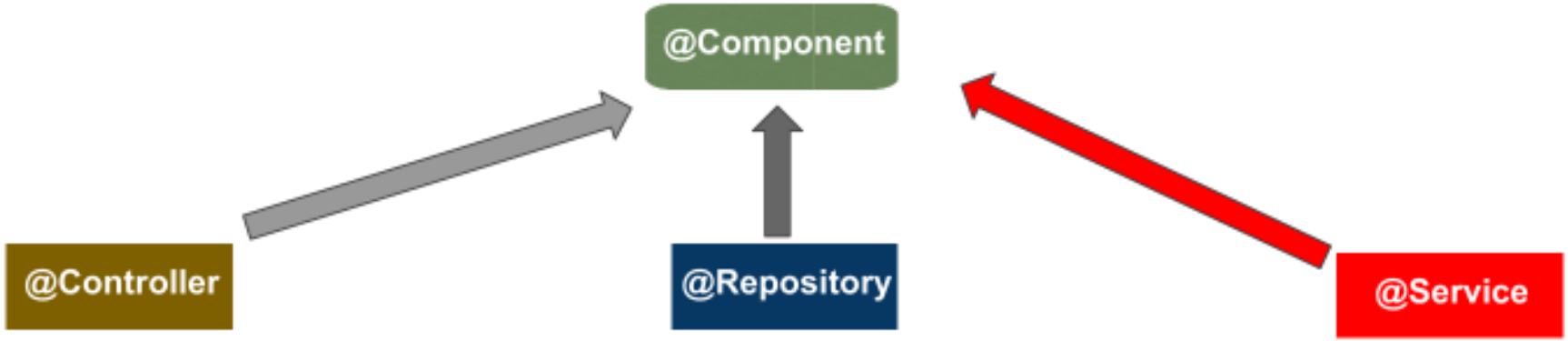


Azzeddine
RIGAT

Define Services with @Service

Specialized Annotation for Services

- Spring provides the @Service annotation



Define Services with @Service

Specialized Annotation for Services

- **@Service** applied to Service implementations

- Spring will automatically register the Service implementation
  - thanks to component-scanning

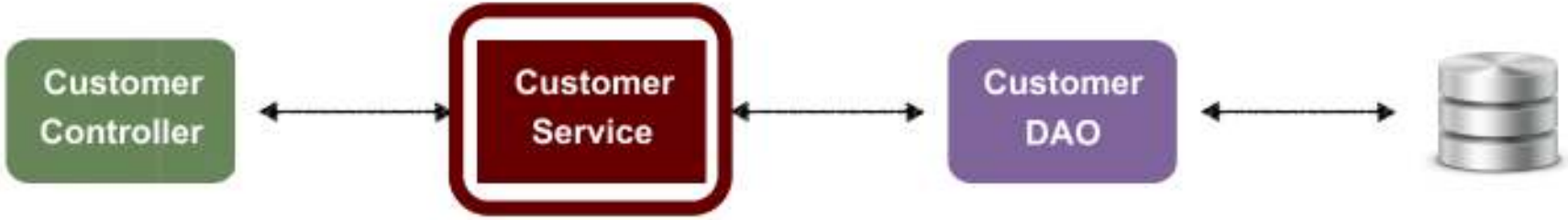
Azzeddine
RIGAT

Define Services with @Service

Custom Service

1. Define Service interface
2. Define Service implementation
   a. Inject the CustomerDAO

Step by Step



Define Services with @Service

Step 1: Define Service interface

```
public interface CustomerService {

    public List<Customer> getCustomers();

}
```

Define Services with @Service

Step 2: Define Service implementation

```
@Service
public class CustomerServiceImpl implements CustomerService {

        @Autowired
        private CustomerDAO customerDAO;

        @Transactional
        public List<Customer> getCustomers() {

        ....
        }
}
```

Define Services with @Service

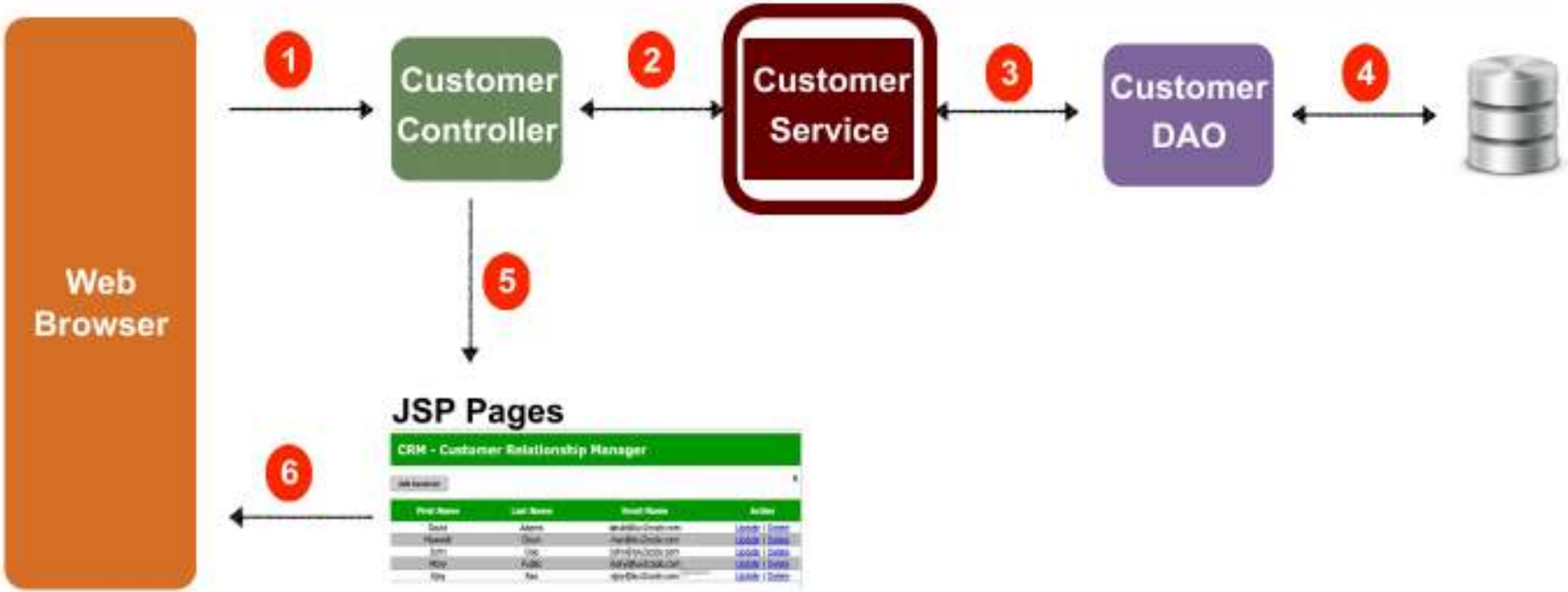
Updates for the DAO implementation

```
@Repository
public class CustomerDAOImpl implements CustomerDAO {

        @Autowired
        private SessionFactory sessionFactory;

        public List<Customer> getCustomers() {



        ....
        }
}
```

Check **08-CRUD-CRM-refactor-add-service-layer**

Azzeddine
RIGAT

Revise the big picture



Topic 4, Part 4 - Add to Database

Azzeddine
RIGAT

Part 4 - Add Customer

Topic 4, Part 4 - Add to Database

Azzeddine
RIGAT

Add Customer

1. **Update list-customer.jsp**
   a. New “Add Customer” button

Step by Step

2. **Create HTML form for new customer**

3. **Process Form Data**
   a. Controller -> Service -> DAO

Assignment : Deadline 2019-Dec-5

Topic 4, Part 5 - Update Database

Azzeddine
RIGAT

Part 5 - Update Database

Topic 4, Part 5 - Update Database

Azzeddine RIGAT

Update Customer

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email	Action
David	Adams	david@luv2code.com	Update
Trupti	Bose	trupti@luv2code.com	Update
Maxwell	Dixon	max@luv2code.com	Update
John	Doe	john@luv2code.com	Update
Ajay	Rao	ajay@luv2code.com	Update
Mary	Zeno	happymary@gmail.com	Update

Each row has an **Update** link

- current customer id embedded in link

When **clicked**

- will load the customer from database pre-populate the form

Topic 4, Part 5 - Update Database

Azzeddine RIGAT

Update Customer

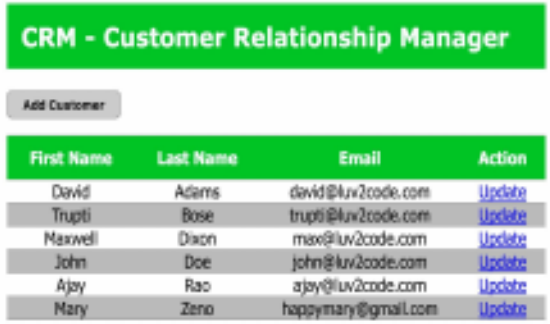
1. **Update list-customers.jsp**
   a. New “Update” link

2. **Create customer-form.jsp**
   a. Pre-populate the form

3. **Process form data**
   a. Controller > Service > DAO

Step by Step

Assignment : Deadline 2019-Dec-5



Topic 4, Part 6 - Delete from Database

Azzeddine
RIGAT

Part 6 - Delete Customer

Topic 4, Part 6 - Delete from Database

Azzeddine RIGAT

Delete Customer

CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email	Action
David	Adams	david@luv2code.com	Update | Delete
Trupti	Bose	trupti@luv2code.com	Update | Delete
Maxwell	Dixon	max@luv2code.com	Update | Delete
John	Doe	john@luv2code.com	Update | Delete
Ajay	Rao	ajay@luv2code.com	Update | Delete
Mary	Zeno	happymary@gmail.com	Update | Delete

Each row has a **Delete** link

- current customer id embedded in link

When **clicked**

- prompt user will delete the customer from database

Topic 4, Part 6 - Delete from Database

Azzeddine
RIGAT

Delete Customer

1. **Add "Delete" link on JSP**

2. **Add code for "Delete"**
   a. Controller > Service > DAO

Step by Step

Assignment : Deadline 2019-Dec-5

Topic 4, Part 6 - Delete from Database

Azzeddine
RIGAT

This the End of CRUD tutorial, now you a strong grasp of developing web application using Spring 5.0 and Hibernate

Next, class will be about how to deal with dependencies using Maven