

Practice Exercise

This document provides a list of exercises to be practiced by learners. Please raise a feedback in Talent Next, should you have any queries.

Skill	Spring Hands-on Mastery
Proficiency	S1
Document Type	Lab Practice Exercises
Author	L & D
Current Version	1.0
Current Version Date	1-Apr-2024
Status	Active

Document Control

Version	Change Date	Change Description	Changed By
1.0	01-Apr-2024	Baseline version	Vimala R

Contents

Practice Exercise	1
Document Control.....	2
Problem Statement 1: Dependency Injection via Setter Method.	4
Problem Statement 2: Dependency Injection via Constructor Method.	5
Problem Statement 3: Reading properties files in Spring with PropertyPlaceholderConfigurer bean	6
Problem Statement 4: Spring Bean Scopes in IoC Container	7
Problem Statement 5: Customize Spring Bean Life Cycle – Initialization and Destruction	8
Problem Statement 6: Implementing AOP in Shopping Cart Application.....	9
Problem Statement 7: Shopping Cart Application using Annotations	10
Problem Statement 8: Create MySQL Table.....	11
Problem Statement 9: The Spring JdbcTemplate for database access	12
Problem Statement 10: Spring ORM application using HibernateTemplate	15
Problem Statement 11: Adding Transaction Management capability to the Application	19
Problem Statement 12: Serving Web Content with Spring MVC	21
Problem Statement 13: Building REST services with Spring MVC	25
Problem Statement 14: Create MySQL Table	26
Problem Statement 15: Create Spring Boot Microservice with Spring Data JPA	27
Problem Statement 16: Create Spring Boot Microservice with Spring Data MongoDB	28
Problem Statement 17: Create Spring Boot Microservice with Spring Data Redis	30
Problem Statement 18: Configuring Spring Boot Actuator Endpoints – Product-ready services....	32

Note – All Problem Statements start in a fresh page.

Problem Statement 1: Dependency Injection via Setter Method.

Mphasis got a requirement to create an online shopping application to sell products online. We need to design the core structure of the application for shopping cart functionality.

Following are the steps you need to follow to build this application,

1. Create the Product POJO class, which has several properties, such as the product id, product name and price. As there are many types of products in your shop, you make the Product class abstract to extend it for different product subclasses.
2. Create two Product subclasses, Battery, and Disc. Each of them has its own properties as follows,

```
public class Battery extends Product {  
    private boolean rechargeable;  
    // setters and getters  
}  
  
public class Disc extends Product {  
    private int capacity;  
    //setters and getters  
}
```

3. Create XML Configuration to do the dependency injection via Setter Methods. (Note: Observe here you can't create the bean of Product class).
4. Write a Main method to bootstrap the Spring core container using XML file.
5. Print out all the beans created by the container.

Problem Statement 2: Dependency Injection via Constructor Method.

Modify the previous application to inject the beans by another way of Dependency Injection – Constructor.

1. In the Product class, add the parameterized constructor for all the properties.
2. Similarly modify the two Product subclasses, Battery, and Disc add the parameterized constructor for the properties.

```
public class Battery extends Product {  
    private boolean rechargeable;  
    // constructors  
}  
  
public class Disc extends Product {  
    private int capacity;  
    // constructors  
}
```

3. Update XML Configuration to do the dependency injection via Constructor Methods. (Note: Observe here you can't create the bean of Product class).
4. Modify the Main method to bootstrap the Spring core container using XML file.
5. Print out all the beans created by the container.

Problem Statement 3: Reading properties files in Spring with `PropertyPlaceholderConfigurer` bean

Modify the previous application to add the discount capability using Spring IoC features.

In a corporate environment, discount details is usually only can 'touch' by your system or database administrator, they just refuse to access your bean configuration file directly, and they will request a separate file for deployment configuration, for example, a simple property, with deployment detail only.

Following are the steps you must perform,

1. Let's create a property file for the discounts for special customer, summer, and end of year season.

discounts.properties:

```
specialcustomer.discount = 0.1  
summer.discount = 0.15  
endofyear.discount = 0.2
```

2. To fix it, you can use **PropertyPlaceholderConfigurer** class to externalize the deployment details into a properties file, and access from bean configuration file via a special format – `${variable}`.
3. Declare a **PropertyPlaceholderConfigurer** in bean configuration file and map to the 'discount.properties' properties file you created just now.
4. Access the values of the discounts.properties file to setup other bean properties.
5. Add a **discount** property to the Product, Battery, and Disc classes of the shopping application, you can inject bean properties using values from a properties file.
6. Modify the Main method to bootstrap the Spring core container using XML file.
7. You can test your shopping cart by adding some products and applying the discounts on it.

Problem Statement 4: Spring Bean Scopes in IoC Container

Modify the previous application to add the Shopping Cart capability using Spring IoC features.

Following are the steps you must perform,

1. Create the **ShoppingCart** class as follows:

```
public class ShoppingCart {  
    private List < Product > items = new ArrayList < Product >();  
    public void addItem(Product item) {  
        items.add(item);  
    }  
    public List < Product > getItems() {  
        return items;  
    }  
}
```

2. Modify the spring bean configuration file and add a Shopping Cart Bean.
3. Modify the Main method to bootstrap the Spring core container using XML file.
4. You can test your shopping cart by adding some products to it as follows,
 - a. Suppose that there are two customers navigating in your shop at the same time.
 - b. The first one gets a shopping cart by the `getBean()` method and adds two products to it.
 - c. The second customer also gets a shopping cart by the `getBean()` method and adds another product to it.
5. How to configure your shopping application in such a way that each customer to get a different shopping cart instance, when the `getBean()` method is called. Which bean scope is used here?

Problem Statement 5: Customize Spring Bean Life Cycle – Initialization and Destruction

Modify the previous application to add the checkout functionality for the shopping application using Spring IoC features.

Following are the steps you must perform,

1. Create a Cashier class to record a shopping cart's products and write the checkout time to a text file. Refer class definition below,

```
public class Cashier {
    private String fileName;
    private String path;
    private BufferedWriter writer;

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public void openFile() throws IOException {
        File targetDir = new File(path);
        if (!targetDir.exists()) {
            targetDir.mkdir();
        }

        File checkoutFile = new File(path, fileName + ".txt");
        if (!checkoutFile.exists()) {
            checkoutFile.createNewFile();
        }
        writer = new BufferedWriter(new OutputStreamWriter(new
        FileOutputStream(checkoutFile, true)));
    }
}
```

```
    public void checkout(ShoppingCart cart) throws IOException {
        writer.write(new Date() + "\t" + cart.getItems() + "\r\n");
        writer.flush();
    }

    public void closeFile() throws IOException {
        writer.close();
    }
}
```

2. In the above class,
 - a. openFile() method first verifies if the target directory and file exists.
 - b. When the checkout() method is called, the current date and cart items are appended to the text file.
 - c. closeFile() method closes the file to release its system resources.
3. Modify the spring bean configuration file and add a Cashier Bean. Add the openFile() method to init-method attribute and closeFile() method to destroy-method attribute in the XML file.
4. Modify the Main method to bootstrap the Spring core container using XML file.
5. You can test your shopping cart by invoking the checkout functionality.

Problem Statement 6: Implementing AOP in Shopping Cart Application

Modify the previous application to add the Logging cross-cutting concern for the shopping application using Spring AOP.

Following are the steps you must perform,

1. Let assume the previous application you created is having the **com.mphasis** package structure.
2. Create an Aspect for the Logging concern.
3. Add the different types of advices action to a specific pointcut expression.
4. Refer below table for more details,

Method Name	Advice Name	Pointcut Expression to be used
getItems	@Before	execution(* com.mphasis.ShoppingCart.getItems())
	@After	
	@AfterReturning	
	@Around	
addItem	@Before	execution(* com.mphasis.ShoppingCart.addItem(..))
	@After	
removeItem	@Before	execution(* com.mphasis.ShoppingCart.removeItem(..))
	@After	
updateItem	@Before	execution(* com.mphasis.ShoppingCart.updateItem(..))
	@After	
	@AfterThrowing	

5. Provide a complete implementation for the above case using Spring Framework's AOP classical and Aspect4j Approach.

Problem Statement 7: Shopping Cart Application using Annotations

After monitoring the application developer came to know that the bootstrap timing is increased due to parsing the huge and number of XML files. Now we got a requirement to redesign the Shopping Cart application using a new feature Annotation.

Following are the steps you must perform,

1. Observer carefully the previous application created using Problem Statement 1 to 8.
2. Identify the alternate annotation for each bean tags.
3. Few of the annotation mentioned below:
 - a. Stereotype - @Component
 - b. @Autowired
 - c. @Value
 - d. @Scope
 - e. @PostConstruct & @PreDestroy
 - f. @Configuration + @Bean
4. Create the Main method to bootstrap the Spring core container using AnnotationConfigApplicationContext.
5. Test all your functionalities again.

Problem Statement 8: Create MySQL Table

SRC Travel Agency is a domestic privatized company that runs the vehicles all over the country. They have several branches at different locations of the country, so that they can provide the transportation facilities between the places. They want desktop application to be developed, where the details of the bookings done, and the customer will be updated from time to time, and user can track the details of the available seats. As a first step we must design the database schema to store the passenger details.

Create a table Passenger_Details with the following attributes:

- Passenger_id
- Passenger_name
- Passenger_dob
- Passenger_phone
- Passenger_email

Where passenger_id will be the primary key. Create the table with all required constraints.

Add few records inside the table:

Passenger_id	Passenger_name	Passenger_dob	Passenger_phone	Passenger_email
1234	Anu	2000-12-05	9876543266	anu@gmail.com
1235	Vikas	2001-01-05	8876543266	vikas@gmail.com
1236	Sharmi	2011-02-15	9666543266	sharmi@gmail.com
1237	Nikitha	1990-12-05	9076543266	nikitha@gmail.com
1238	Vinu	1999-06-05	9855432066	vinu@gmail.com

Problem Statement 9: The Spring JdbcTemplate for database access

Refer the passenger table created in Problem Statement 1 and develop a spring application to perform all CRUD operations like Create, Read, Update and Delete by using Spring JdbcTemplate.

Steps to create the application:

- Create a spring maven project and update pom.xml with below required dependencies.

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.48</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- To fetch the records from the database, use a RowMapper object.
- Write an annotation or XML based configuration to configure DataSource and JdbcTemplate objects.
- Create a DAO layer for all Database operations
- Create a Service layer which connects to DAO layer.
- Create Test class to call all the CRUD methods.

Spring Hands-on Mastery - S1 - Practice Exercise

Refer Below Screenshots for Your Menu-Driven application:

1. Retrieve Passenger Details

```
Console | Progress | Problems
App (2) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 5:32:08 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
Passenger_id  Passenger_name  dob      Phone      Email
1235          Vikas           2001-01-05  8876543266  vikas@gmail.com
1236          Sharmi         2011-02-15  9666543266  sharmi@gmail.com
1237          Nikitha        1990-12-05  9076543266  nikitha@gmail.com
1238          Vinu           1999-06-05  9855432066  vinu@gmail.com
Do u want to continue? press y or n
y
```

2. Insert Passenger Details

```
Console | Progress | Problems
App (2) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 5:25:44 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
1
Insert Passenger details
----- Inserting records -----
Passenger Details inserted successfully.....
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
Passenger_id  Passenger_name  dob      Phone      Email
1235          Vikas           2001-01-05  8876543266  vikas@gmail.com
1236          Sharmi         2011-02-15  9666543266  sharmi@gmail.com
1237          Nikitha        1990-12-05  9076543266  nikitha@gmail.com
1238          Vinu           1999-06-05  9855432066  vinu@gmail.com
1239          Anu            2000-01-01  1765433245  anu@gmail.com
Do u want to continue? press y or n
y
```

3. Update Passenger Details

```
Console | Progress | Problems
App (2) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 5:28:05 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
2
Update Passenger details
----- Update 1 Record -----
Passenger details updated successfully.....
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
Passenger_id  Passenger_name  dob      Phone      Email
1236          Sharmi         2011-02-15  9666543266  sharmi@gmail.com
1237          Nikitha        1990-12-05  9076543266  nikitha@gmail.com
1238          Vinu           1999-06-05  9855432066  vinu@gmail.com
1239          Anu            2000-01-01  1765433245  anuR@gmail.com
Do u want to continue? press y or n
y
```

4. Delete Passenger Details

```
App (2) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 5:28:05 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
4
Delete Passenger details
Enter the passenger id to delete
1235
Record deleted.....
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
Passenger_id  Passenger_name  dob      Phone      Email
1236          Sharmi          2011-02-15  9666543266  sharmi@gmail.com
1237          Nikitha         1990-12-05  9076543266  nikitha@gmail.com
1238          Vinu            1999-06-05  9855432066  vinu@gmail.com
1239          Anu             2000-01-01  1765433245  anu@gmail.com
Do u want to continue? press y or n
```

Problem Statement 10: Spring ORM application using HibernateTemplate

Refer the passenger table created in Problem Statement 1 and Develop a spring application to perform all CRUD operations like Create, Read, Update and Delete by using Spring HibernateTemplate.

- Create a spring maven project and update pom.xml with below required dependencies.

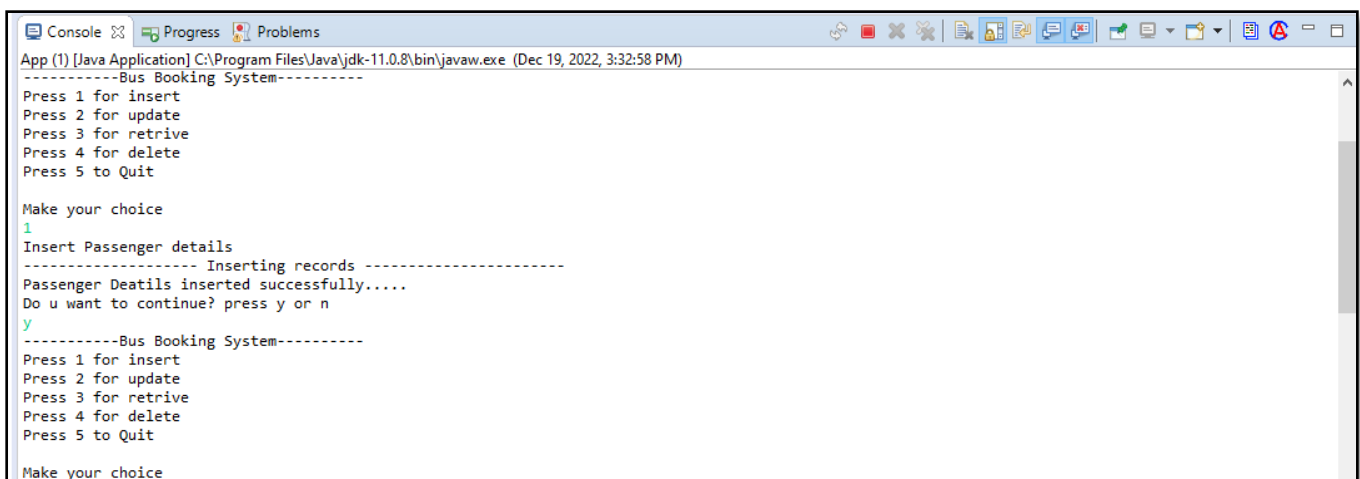
```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
  </dependency>
</dependencies>
```

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.1.0.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-java8</artifactId>
  <version>5.1.0.Final</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.48</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
</dependencies>
```

- Add all the required JPA annotations inside Passenger class to create persistent object.
- Write an annotation or XML based configuration to configure DataSource, SessionFactory and HibernateTemplate objects.
- Create a DAO layer for DB operations use save, update, delete, get and loadAll methods.
- Create a Service layer which connects to DAO layer.
- Create Test class to call all the CRUD methods.

Refer Below Screenshots for Your Menu-Driven application:

1. Passenger Details inserted



```
App (1) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 3:32:58 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit
Make your choice
1
Insert Passenger details
----- Inserting records -----
Passenger Deatils inserted successfully.....
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit
Make your choice
```


2. Retrieve Passenger Details

```
App (1) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 4:15:15 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
Passenger_id  Passenger_name  dob      Phone      Email
1             Anu            2000-01-01  1765433245  anu@gmail.com
2             Jith          2002-03-05  1765432775  jith@gmail.com
3             Tharun        2008-06-02  1765322466  Tharun@gmail.com
Do u want to continue? press y or n
```

```
App (1) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 3:32:58 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
1  Anu    2000-01-01  1765433245  anu@gmail.com
2  Jith   2002-03-05  1765432775  jith@gmail.com
3  Tharun 2008-06-02  1765322466  Tharun@gmail.com
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
```

3. Update Passenger details

```
App (1) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 3:32:58 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
2
Update Passenger details
----- Update 1 Record -----
Passenger details updated successfully.....
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
```

4. Delete Passenger Details

```
App (1) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (Dec 19, 2022, 3:58:40 PM)
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
4
Delete Passenger details
Enter the passenger id to delete
1
Record deleted.....
Do u want to continue? press y or n
y
-----Bus Booking System-----
Press 1 for insert
Press 2 for update
Press 3 for retrieve
Press 4 for delete
Press 5 to Quit

Make your choice
3
Retrieve Passenger details
----- Fetch all records -----
2   Jith   2002-03-05   1765432775   jith@gmail.com
3   Tharun 2008-06-02   1765322466   Tharun@gmail.com
Do u want to continue? press y or n
```

Problem Statement 11: Adding Transaction Management capability to the Application

Bus Booking application maintains passenger details in Passenger_details table. While adding new passenger information the data should be updated. If there is an exception while processing any passenger data, roll back all the updated data for that passenger but other passenger details should persist. Use Declarative approach for Transaction Management.

Modify the application with transaction,

- Create a spring maven project and update pom.xml with required dependencies.

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
</dependencies>
```

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.1.0.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-java8</artifactId>
  <version>5.1.0.Final</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.48</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
</dependencies>
```

- Create Passenger_Details Document.
- Create a DAO interface that queries Passenger class.
- Create a DAO implementation class to execute the queries.
- Create a Passenger Service that will use the PassengerDAO implementation and provide transaction management to the method (Use @Transaction)
- Configure the DataSource and TransactionManager properties in XML file.
- Modify the Spring Initializer class to generate the output.

Problem Statement 12: Serving Web Content with Spring MVC

Create a spring MVC web application to implement Passenger Registration use case.

Follow the below steps to create the application,

- Create a maven project and update pom.xml with below required dependencies.

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>5.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
  </dependency>
  <dependency>
    <groupId>javax.el</groupId>
    <artifactId>javax.el-api</artifactId>
    <version>3.0.0</version>
  </dependency>
</dependencies>
```

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
</dependency>
<dependency>
  <groupId>javax.el</groupId>
  <artifactId>javax.el-api</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-mongodb</artifactId>
  <version>2.2.6.RELEASE</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
</dependencies>
```

- Configure the DispatcherServlet for MVC application in web.xml as shown below.

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>

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

  <servlet-mapping>
    <servlet-name>spring-mvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

- Configure the view resolver in spring configuration file **spring-mvc-servlet.xml** as shown below.

```
<?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:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
                           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">

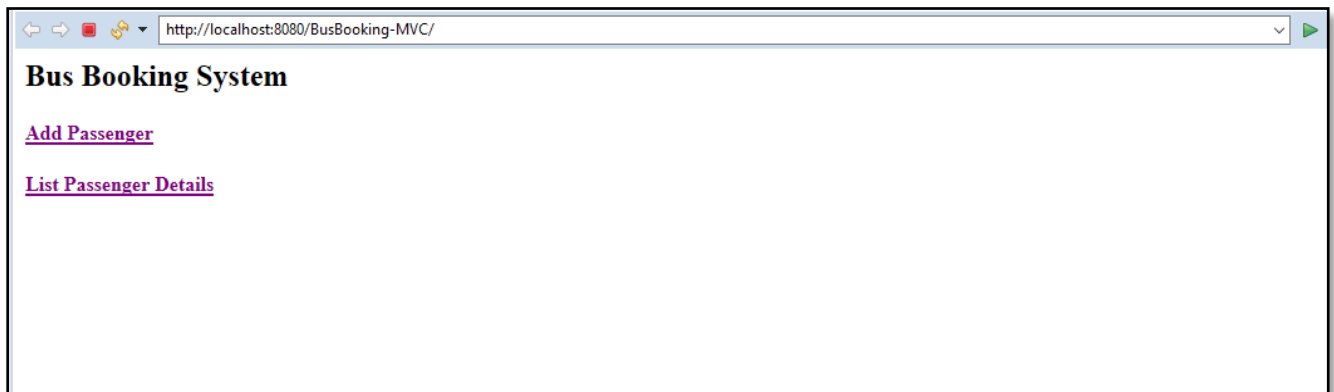
    <context:component-scan
        base-package="com" />

    <mvc:annotation-driven />

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

- Create the Model class as Passenger_Details (Passenger_id, Passenger_name, Passenger_dob, Passenger_phone, Passenger_email)
- Create Controller class as PassengerController.
- Create PassengerForm.jsp which has Passenger registration form.

Home Page



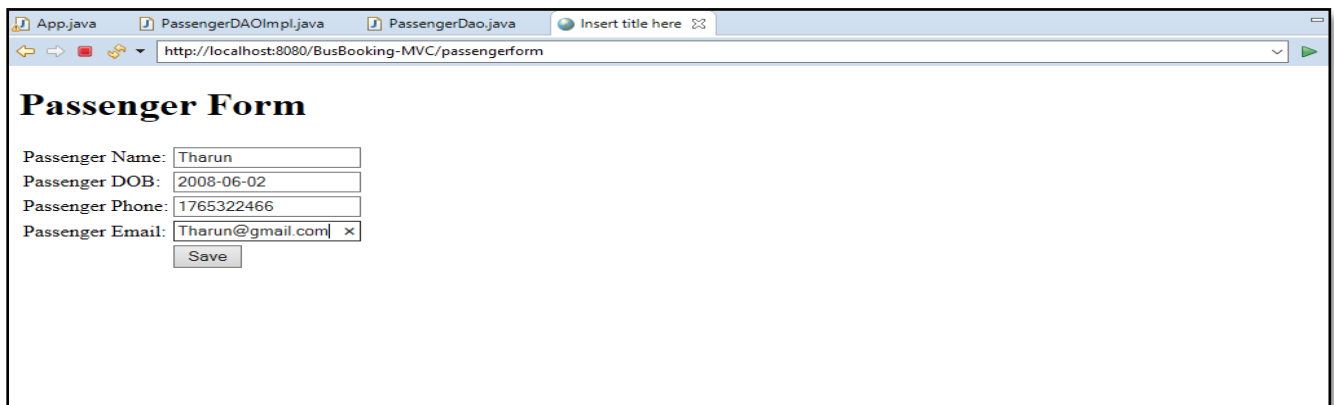
http://localhost:8080/BusBooking-MVC/

Bus Booking System

[Add Passenger](#)

[List Passenger Details](#)

Passenger Form



App.java PassengerDAOImpl.java PassengerDao.java Insert title here

http://localhost:8080/BusBooking-MVC/passengerform

Passenger Form

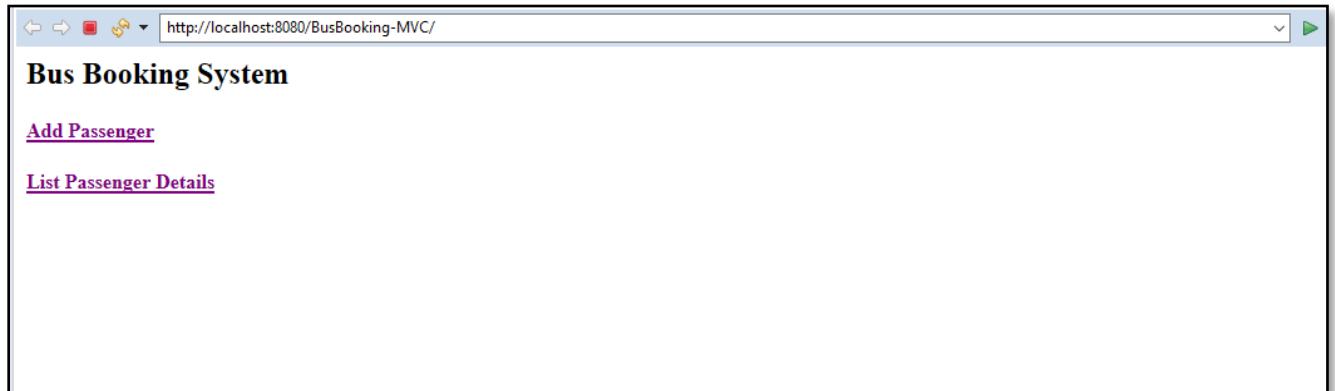
Passenger Name:

Passenger DOB:

Passenger Phone:

Passenger Email: X

Form when submitted is redirected to home page



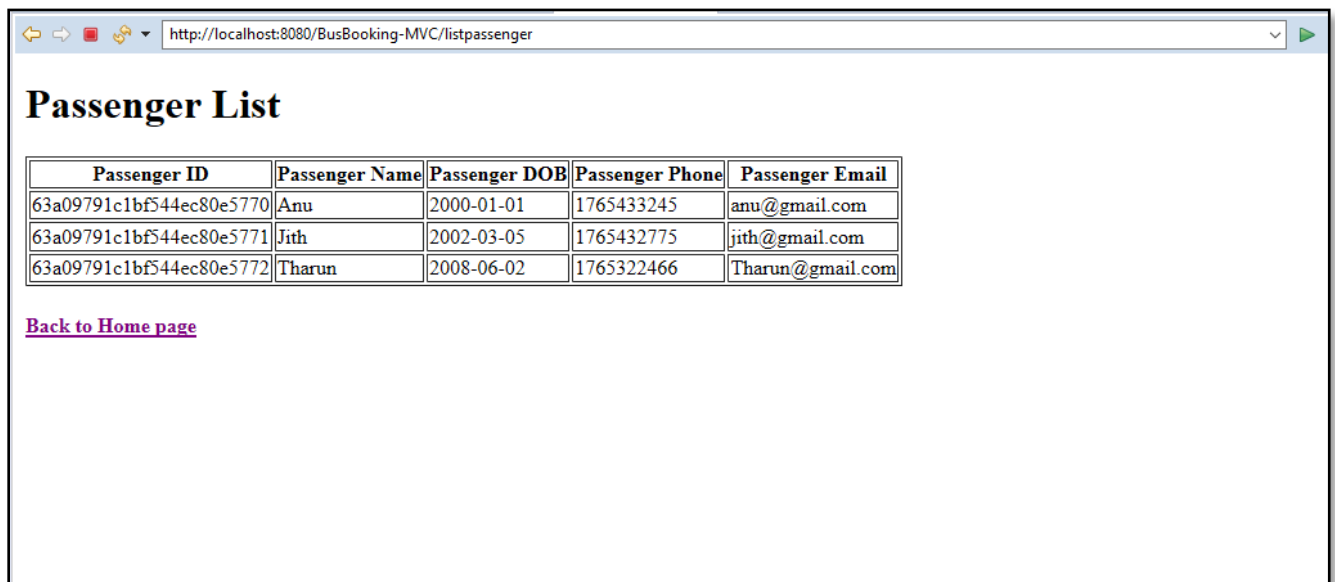
A screenshot of a web browser showing the 'Bus Booking System' home page. The browser's address bar displays 'http://localhost:8080/BusBooking-MVC/'. The page has a white background with a black border. At the top, the title 'Bus Booking System' is displayed in a bold, black font. Below the title, there are two links: 'Add Passenger' and 'List Passenger Details', both underlined and in a purple color.

Bus Booking System

[Add Passenger](#)

[List Passenger Details](#)

Passenger Details



A screenshot of a web browser showing the 'Passenger List' page. The browser's address bar displays 'http://localhost:8080/BusBooking-MVC/listpassenger'. The page has a white background with a black border. At the top, the title 'Passenger List' is displayed in a bold, black font. Below the title, there is a table with five columns: 'Passenger ID', 'Passenger Name', 'Passenger DOB', 'Passenger Phone', and 'Passenger Email'. The table contains three rows of data. Below the table, there is a link 'Back to Home page' which is underlined and in a purple color.

Passenger List

Passenger ID	Passenger Name	Passenger DOB	Passenger Phone	Passenger Email
63a09791c1bf544ec80e5770	Anu	2000-01-01	1765433245	anu@gmail.com
63a09791c1bf544ec80e5771	Jith	2002-03-05	1765432775	jith@gmail.com
63a09791c1bf544ec80e5772	Tharun	2008-06-02	1765322466	Tharun@gmail.com

[Back to Home page](#)

Problem Statement 13: Building REST services with Spring MVC

EasyBooking Company wants to create an application to perform CRUD operations by using REST API.

Follow the below steps to create the application,

- Create a maven project and update pom.xml with same dependencies used in Problem Statement 6
- Develop a below Passenger REST API to perform the crud operations using appropriate annotations.

URI	Methods	Description	Format
/passengers	POST	Add passenger details	JSON
/passengers	GET	Get all passengers	JSON
/passengers/{passengerId}	PUT	Update passenger details by passengerId	JSON
/passengers/{passengerId}	GET	Get passenger details by passengerId	JSON
/passengers/{passengerId}	DELETE	Delete passenger by passengerId	JSON

- Test all the RESTful API's by using Postman REST client application.

Problem Statement 14: Create MySQL Table

Create a table Book_Details with the following attributes:

- id
- book_title
- book_publisher
- book_isbn
- book_number_of_pages
- book_year

Where id will be the primary key.

Add few records inside the table:

Title	ISBN	Page	Year
.NET Core in Action	978-1-61729-427-3	288	2018
.NET Development Using the Compiler API	978-1-484221-10-5	176	2016
.Net Framework 4.5 Expert Programming Cookbook	978-1-84968-742-3	276	2013
.NET Framework Essentials, 2nd Edition	978-0-596-00302-9	320	2002
.NET IL Assembler	978-1-4302-6761-4	492	2014
.NET Standard 2.0 Cookbook	978-1-78883-466-7	394	2018
.NET Test Automation Recipes	978-1-59059-663-0	408	2006
10 LED Projects for Geeks	978-1-59327-825-0	240	2018
101 Design Ingredients to Solve Big Tech Problems	978-1-93778-532-1	298	2013
101 Excel 2013 Tips, Tricks and Timesavers	978-1-118-64218-4	312	2013
101 UX Principles	978-1-78883-736-1	414	2018
101 Windows Phone 7 Apps	978-0-672-33552-5	1152	2011
20 Easy Raspberry Pi Projects	978-1-59327-843-4	288	2018
20 Recipes for Programming MVC 3	978-1-4493-0986-2	122	2011
20 Recipes for Programming PhoneGap	978-1-4493-1954-0	78	2012
21 Recipes for Mining Twitter	978-1-4493-0316-7	72	2011
21st Century C	978-1-4493-2714-9	296	2012
21st Century C, 2nd Edition	978-1-49190-389-6	408	2014
21st Century Robot	978-1-44933-821-3	278	2014
25 Recipes for Getting Started with R	978-1-4493-0323-5	50	2011

Problem Statement 15: Create Spring Boot Microservice with Spring Data JPA

Mphasis got a requirement to create an eBookStore portal, wherein multiple users access the books details. We need to design the API for the application so we can achieve the interoperability feature in the application.

Technology stack:

- Spring Boot
- Spring REST
- Spring Data JPA

Building a RESTful web application where CRUD operations to be carried out on entities.

Here will have multiple layers into the application:

1. Create an Entity: Book
2. Create a BookStoreRepository interface and will make use of Spring Data JPA (JpaRepository)
 - a. Will inherit few behavior's:
 - Add the Book details
 - Update the Book details
 - Will have findAll books method
 - Will have findByIdByBookId method
 - Delete book by id method
 - b. Need to add few behavior's using JPA Query Methods:
 - Will have findByBookTitle method
 - Will have findByBookPublisherLike method
 - Will have findByYear method
3. Create a BookStoreService class and will expose all these services
4. Finally, create a BookStoreRestController will have the following uri's:

URI	METHODS	Description	Format
/books	POST	Add the Book details	JSON
/books	PUT	Update the Book details	JSON
/books	GET	Give all books details	JSON
/books/{book_id}	GET	Give a single book description searched based on id	JSON
/books/{book_id}	DELETE	Delete book by id method	
/books/{book_title}	GET	Give all books details searched based on title	JSON
/books/{book_publisher}	GET	Give all books details searched based on publisher	JSON
/books?year={book_year}	GET	Give all books details searched based on year	JSON

5. Running on a web server tier (using tomcat).

Problem Statement 16: Create Spring Boot Microservice with Spring Data MongoDB

Due to huge traffic now there's an impact on the performance. To handle the spike, it's decided to replace the backend from MySQL to MongoDB. So, need to re-design the eBookStore API with Spring Data MongoDB project.

Technology stack:

- Spring Boot
- Spring REST
- Spring Data MongoDB

Start the MongoDB Server:

1. Start the Standalone MongoDB Server:
mongod.exe --dbpath=c:\mongodata
2. MongoDB server will be running on localhost:27107

Building a RESTful web application where CRUD operations to be carried out on entities.

Here will have multiple layers into the application:

1. Update the host & port details in application.properties.
2. Create an Entity: Book
3. Create a BookStoreRepository interface and will make use of Spring Data MongoDB (MongoRepository)
 - a. Will Inherit few behavior's:
 - Add the Book details
 - Update the Book details
 - Will have findAll books method
 - Will have findByIdByBookId method
 - Delete book by id method
 - b. Need to add few behavior's using Query Methods:
 - Will have findByBookTitle method
 - Will have findByBookPublisherLike method
 - Will have findByYear method
4. Create a BookStoreService class and will expose all these services
5. Finally, create a BookStoreRestController will have the following uri's:

URI	METHODS	Description	Format
/books	POST	Add the Book details	JSON
/books	PUT	Update the Book details	JSON
/books	GET	Give all books details	JSON
/books/{book_id}	GET	Give a single book description searched based on id	JSON
/books/{book_id}	DELETE	Delete book by id method	
/books/{book_title}	GET	Give all books details searched based on title	JSON
/books/{book_publisher}	GET	Give all books details searched based on publisher	JSON
/books?year={book_year}	GET	Give all books details searched based on year	JSON

6. Running on a web server tier (using tomcat).

Problem Statement 17: Create Spring Boot Microservice with Spring Data Redis

For faster performance and low latency, the developer team decided to use Redis - Fast, open source in-memory data structure store, used as a database and cache. So, let's re-design the eBookStore API with Spring Data Redis project.

Technology stack:

- Spring Boot
- Spring REST
- Spring Data Redis

Start the MongoDB Server:

1. Start the Standalone Redis Server:

redis-server.exe

2. Redis server will be running on localhost:6379

Building a RESTful web application where CRUD operations to be carried out on entities.

Here will have multiple layers into the application:

1. Update the host & port details in application.properties.
2. Create an Entity: Book
3. Create a BookStoreRepository interface and will make use of Spring Data Redis (CrudRepository)
 - a. Will Inherit few behavior's:
 - Add the Book details
 - Update the Book details
 - Will have findAll books method
 - Will have findAllByBookId method
 - Delete book by id method
 - b. Need to add few behavior's using Query Methods:
 - Will have findByBookTitle method
 - Will have findByBookPublisherLike method
 - Will have findByYear method
4. Create a BookStoreService class and will expose all these services
5. Finally, create a BookStoreRestController will have the following uri's:

URI	METHODS	Description	Format
/books	POST	Add the Book details	JSON
/books	PUT	Update the Book details	JSON
/books	GET	Give all books details	JSON
/books/{book_id}	GET	Give a single book description searched based on id	JSON
/books/{book_id}	DELETE	Delete book by id method	
/books/{book_title}	GET	Give all books details searched based on title	JSON
/books/{book_publisher}	GET	Give all books details searched based on publisher	JSON
/books?year={book_year}	GET	Give all books details searched based on year	JSON

6. Running on a web server tier (using tomcat).

Problem Statement 18: Configuring Spring Boot Actuator Endpoints – Product-ready services.

Modify the previous application to add the capability of Actuator in your Spring Boot application, you can monitor, gather metrics, understand traffic, and manage your application by invoking different HTTP endpoints exposed by Spring Boot Actuator.

Technology stack:

- Spring Actuator

Initialization steps are mentioned below:

1. To enable Spring Boot Actuator, we just need to add the *spring-boot-actuator* dependency to our package manager.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

2. Add the property into application.properties:

management.endpoints.web.exposure.include=*

3. Try accessing some available endpoints:

ENDPOINTS	USAGE
/actuator/metrics	To view the application metrics such as memory used, memory free, threads, classes, system uptime etc.
/actuator/env	To view the list of Environment variables used in the application.
/actuator/beans	To view the Spring beans and its types, scopes, and dependency.
/actuator/health	To view the application health
/actuator/info	To view the information about the Spring Boot application.
/actuator/trace	To view the list of Traces of your Rest endpoints.
/actuator/configprops	Allows us to fetch all @ConfigurationProperties beans.
/actuator/heapdump	Builds and returns a heap dump from the JVM used by our application.
/actuator/threaddumps	Dumps the thread information of the underlying JVM.