1 .a) Write a program to insert a record using Spring JDBC

Following is the content of the Data Access Object interface file **StudentDAO.java**.

```
package com.mrcet;
import java.util.List;
import javax.sql.DataSource;
public interface StudentDAO {
/**
* This is the method to be used to initialize
* database resources ie. connection.
*/
public void setDataSource(DataSource ds);
/**
* This is the method to be used to create
* a record in the Student table.
*/
public void create(String name, Integer age);

/**
* This is the method to be used to list down
* all the records from the Student table.
*/
public List<Student> listStudents();
}
```

Following is the content of the **Student.java** file.

```
package com.mrcet;
public class Student {
private Integer age;
private String name;
private Integer id;
public void setAge(Integer age) {
this.age = age;
}
public Integer getAge() {
return age;
}
public void setName(String name) {
this.name = name;
}
public String getName() {
return name;
}
public void setId(Integer id) {
this.id = id;
}
public Integer getId() {
return id;
}
}
```

Following is the content of the **StudentMapper.java** file.

```
package com.mrcet;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
public class StudentMapper implements RowMapper<Student> {
public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student student = new Student();
student.setId(rs.getInt("id"));
student.setName(rs.getString("name"));
student.setAge(rs.getInt("age"));
return student;
}
}
```

Following is the implementation class file **StudentJDBCTemplate.java** for the defined
DAO interface StudentDAO.

```
package com.mrcet;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
public class StudentJDBCTemplate implements StudentDAO {
private DataSource dataSource;
private JdbcTemplate jdbcTemplateObject;
public void setDataSource(DataSource dataSource) {
this.dataSource = dataSource;
this.jdbcTemplateObject = new JdbcTemplate(dataSource);
}
public void create(String name, Integer age) {
String insertQuery = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( insertQuery, name, age);
System.out.println("Created Record Name = " + name + " Age = " + age);
return;
}
public List<Student> listStudents() {
String SQL = "select * from Student";
List <Student> students = jdbcTemplateObject.query(SQL, new StudentMapper());
return students;
}
}
```

Following is the content of the **MainApp.java** file.

```
package com.mrcet;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.mrcet.StudentJDBCTemplate;

public class MainApp {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
StudentJDBCTemplate studentJDBCTemplate =
(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
```

```
System.out.println("------Records Creation--------" );
studentJDBCTemplate.create("Zara", 11);
studentJDBCTemplate.create("Nuha", 2);
studentJDBCTemplate.create("Ayan", 15);
System.out.println("------Listing Multiple Records--------" );
List<Student> students = studentJDBCTemplate.listStudents();
for (Student record : students) {
System.out.print("ID : " + record.getId() );
System.out.print(", Name : " + record.getName() );
System.out.println(", Age : " + record.getAge());
}
}
}
```

Following is the configuration file **Beans.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"
xsi:schemaLocation = "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">
<!-- Initialization for data source -->
<bean id = "dataSource"
class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name = "driverClassName" value = "com.mysql.cj.jdbc.Driver"/>
<property name = "url" value = "jdbc:mysql://localhost:3306/TEST"/>
<property name = "username" value = "root"/>
<property name = "password" value = "admin"/>
</bean>
<!-- Definition for studentJDBCTemplate bean -->
<bean id = "studentJDBCTemplate"
class = "com.mrcet.StudentJDBCTemplate">
<property name = "dataSource" ref = "dataSource" />
</bean>
</beans>
```

Output

```
------Records Creation--------
Created Record Name = Zara Age = 11
Created Record Name = Nuha Age = 2
Created Record Name = Ayan Age = 15
------Listing Multiple Records--------
ID : 1, Name : Zara, Age : 11
ID : 2, Name : Nuha, Age : 2
ID : 3, Name : Ayan, Ag
```

b) Write short note on DDL query with example

DDL

DDL stands for Data Definition Language.
DDL is used to define the structures like schema, database, tables, constraints etc.
Examples of DDL are create and alter statements.
Commands
CREATE, DROP, RENAME and
ALTER.

2.a) Write a program to Create a Simple Login form using React js

App.js----------

```
import './App.css';
import React, { useState } from "react";
import ReactDOM from "react-dom";
function App() {
// React States
const [errorMessages, setErrorMessages] = useState({});
const [isSubmitted, setIsSubmitted] = useState(false);
// User Login info
const database = [
{
username: "user1",
password: "pass1"
},
{
username: "user2",
password: "pass2"
}
];
const errors = {
uname: "invalid username",
```

```
pass: "invalid password"
};
const handleSubmit = (event) => {
//Prevent page reload
event.preventDefault();
var { uname, pass } = document.forms[0];
// Find user login info
const userData = database.find((user) => user.username === uname.value);
// Compare user info
if (userData) {
if (userData.password !== pass.value) {
// Invalid password
setErrorMessages({ name: "pass", message: errors.pass });
} else {
setIsSubmitted(true);
}
} else {
// Username not found
setErrorMessages({ name: "uname", message: errors.uname });
}
};
// Generate JSX code for error message
const renderErrorMessage = (name) =>
name === errorMessages.name && (
<div className="error">{errorMessages.message}</div>
);
// JSX code for login form
const renderForm = (
<div className="form">
<form onSubmit={handleSubmit}>
<div className="input-container">
```

<label>Username </label>

<input type="text" name="uname" required />

{renderErrorMessage("uname")}

<label>Password </label>

<input type="password" name="pass" required />

{renderErrorMessage("pass")}

<input type="submit" />

</form>

);

return (

Sign In

{isSubmitted ? User is successfully logged in : renderForm}

);

}

export default App;

----------Now Create a App.css file in same folder----------

App.css

.app {

font-family: sans-serif;

display: flex;

align-items: center;

```
justify-content: center;
flex-direction: column;
gap: 20px;
height: 100vh;
font-family: Cambria, Cochin, Georgia, Times, "Times New Roman", serif;
background-color: #f8f9fd;
}
input[type="text"],
input[type="password"] {
height: 25px;
border: 1px solid rgba(0, 0, 0, 0.2);
}
input[type="submit"] {
margin-top: 10px;
cursor: pointer;
font-size: 15px;
background: #01d28e;
border: 1px solid #01d28e;
color: #fff;
padding: 10px 20px;
}
input[type="submit"]:hover {
background: #6cf0c2;
}
.button-container {
display: flex;
justify-content: center;
}
.login-form {
background-color: white;
padding: 2rem;
```

```
box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
}
.list-container {
display: flex;
}
.error {
color: red;
font-size: 12px;
}
.title {
font-size: 25px;
margin-bottom: 20px;
}
.input-container {
display: flex;
flex-direction: column;
gap: 8px;
margin: 10px;
}
```

Output

b) Explain Real World Applications of Cloud Computing.

Real World Applications of Cloud Computing

In simple Cloud Computing refers to the on-demand availability of IT resources
over internet. It delivers different types of services to the customer over the internet. There are three basic types of services models are available in cloud computing i.e., Infrastructure As A Service (IAAS), Platform As A Service (PAAS),
Software As A Service (SAAS).

1. **Online Data Storage:** Cloud computing allows storing data like files, images, audios, and videos, etc on the cloud storage. The organization need not set physical storage systems to store a huge volume of business data

which costs so high nowadays. As they are growing technologically, data generation is also growing with respect to time, and storing that becoming problem. In that situation, Cloud storage is providing this service to store and access data any time as per requirement.

2. **Backup and Recovery :** Cloud vendors provide security from their side by storing safe to the data as well as providing a backup facility to the data. They offer various recovery application for retrieving the lost data. In the traditional way backup of data is a very complex problem and also it is very difficult sometimes impossible to recover the lost data. But cloud computing has made backup and recovery applications very easy where there is no fear of running out of backup media or loss of data.

3. **Bigdata Analysis :** We know the volume of big data is so high where storing
that in traditional data management system for an organization is impossible. But cloud computing has resolved that problem by allowing the organizations to store their large volume of data in cloud storage without worrying about physical storage. Next comes analyzing the raw data and finding out insights or useful information from it is a big challenge as it requires high-quality tools for data analytics. Cloud computing provides the biggest facility to organizations in terms of storing and analyzing big data.

4. **Testing and development :** Setting up the platform for development and finally performing different types of testing to check the readiness of the product before delivery requires different types of IT resources and infrastructure. But Cloud computing provides the easiest approach for development as well as testing even if deployment by using their IT resources with minimal expenses. Organizations find it more helpful as they got scalable and flexible cloud services for product development, testing, and deployment.

5. **Anti-Virus Applications :** Previously, organizations were installing antivirus
software within their system even if we will see we personally also keep antivirus software in our system for safety from outside cyber threats. But nowadays cloud computing provides cloud antivirus software which means the software is stored in the cloud and monitors your system/organization's system remotely. This antivirus software identifies the security risks and fixes them. Sometimes also they give a feature to download the software.

6. **E-commerce Application :** Cloud-based e-commerce allows responding quickly to the opportunities which are emerging. Users respond quickly to the market opportunities as well as the traditional e-commerce responds to the challenges quickly. Cloud-based e-commerce gives a new approach to doing business with the minimum amount as well as minimum time possible. Customer data, product data, and other operational systems are managed in cloud environments.

7. **Cloud computing in education :** Cloud computing in the education sector brings an unbelievable change in learning by providing e-learning, online distance learning platforms, and student information portals to the students. It is a new trend in education that provides an attractive environment for

learning, teaching, experimenting, etc to students, faculty members, and researchers. Everyone associated with the field can connect to the cloud of their organization and access data and information from there.
8. **E-Governance Application :** Cloud computing can provide its services to multiple activities conducted by the government. It can support the government to move from the traditional ways of management and service providers to an advanced way of everything by expanding the availability of the environment, making the environment more scalable and customized. It can help the government to reduce the unnecessary cost in managing, installing, and upgrading applications and doing all these with help of could computing and utilizing that money public service.

3.a) Write a program to create a simple calculator Application using React JS

```
class App extends Component {
constructor() {
super()
this.state = { operations: [] }
}
.....
}
render() {
 return (
 <div className="App">
 <Display data={this.state.operations} />
 <Buttons>
 <Button onClick={this.handleClick} label="C"
value="clear" />
 <Button onClick={this.handleClick} label="7" value="7" />
 <Button onClick={this.handleClick} label="4" value="4" />
 <Button onClick={this.handleClick} label="1" value="1" />
 <Button onClick={this.handleClick} label="0" value="0" />
<Button onClick={this.handleClick} label="/" value="/" />
 <Button onClick={this.handleClick} label="8" value="8" />
```

```
 <Button onClick={this.handleClick} label="5" value="5" />
 <Button onClick={this.handleClick} label="2" value="2" />
 <Button onClick={this.handleClick} label="." value="." />
<Button onClick={this.handleClick} label="x" value="*" />
 <Button onClick={this.handleClick} label="9" value="9" />
 <Button onClick={this.handleClick} label="6" value="6" />
 <Button onClick={this.handleClick} label="3" value="3" />
 <Button label="" value="null" /> <Button
onClick={this.handleClick} label="-" value="-" />
 <Button onClick={this.handleClick} label="+" size="2"
value="+" />
 <Button onClick={this.handleClick} label="=" size="2"
value="equal" />
 </Buttons>
 </div>
 )
class Buttons extends Component {
render() {
return <div className="Buttons"> {this.props.children} </div>
}
} class Button extends Component {
 render() {
```

```
 return (
 <div
 onClick={this.props.onClick}
 className="Button"
```

```
 data-size={this.props.size}
 data-value={this.props.value}
 >
 {this.props.label}
 </div>
 )
 }
}
class Display extends Component {
 render() {
 const string = this.props.data.join('')
 return <div className="Display"> {string} </div>
 }
}
handleClick = e => {
 const value = e.target.getAttribute('data-value')
 switch (value) {
 case 'clear':
 this.setState({
 operations: [],
 })
 break
 case 'equal':
 this.calculateOperations()
 break
 default:
 const newOperations = update(this.state.operations, {
 $push: [value],
 })
```

```
 this.setState({
 operations: newOperations,
 })
 break
 }
calculateOperations = () => {
 let result = this.state.operations.join('')
 if (result) {
 result = math.eval(result)
 result = math.format(result, { precision: 14 })
```



```
 result = String(result)
 this.setState({
 operations: [result],
 })
 }
 }
```

b) Explain in detail Single Page Applications React Forms with example.

A Single Page Application (SPA) is a web application that is designed to be displayed as a single, static page.

This approach makes the application more user-friendly and easier to navigate, as users can see the entire application at once.

With a traditional web application, users are redirected to a new page every time they make a change to their data, which can be frustrating if they want to edit information in multiple places.

In contrast, SPA-based applications are completely focused on one thing at a time, allowing users to work through each step in a consistent manner.

Creating Single Page Application With React

You have to follow the steps below for building a react single page application;

1. Use your desired location to create the react app with the command below;

```
npx create-react-app app-name
```

App-name directory to be built in the following default files:

```
app-name
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    ├── serviceWorker.js
    └── setupTests.js
```

2. Execute the following command to install react-router-dom:

```
npm install react-router-dom
```

3. Wrapping The App Component

React Router has 2 types of routers: BrowserRouter and HashRouter. We're using BrowserRouter in this example because it makes the URL's look like example.com/about rather than example.com/#/about. React Router uses a hash (hash) at the end of the URL to determine what page to load.

You must include the code below in your src.index.js:

```
import React from "react"
import { render } from "react-dom"
import { BrowserRouter } from "react-router-dom"
```

```
import App from "./App"
render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.querySelector("#root")
)
```

4. The following code should be used to create a file named src/pages/HomePage.js:

```
 import React from "react";
export default function HomePage() {
return (
  <>
   <h1>Hey from HomePage</h1>
    <p>This is your awesome HomePage subtitle</p>
  </>
);
}
```

5. The following code should be used to create a file named src/pages/UserPage.js.

```
import React from "react";
import { useParams } from "react-router-dom";
export default function UserPage() {
  let { id } = useParams();
  return (
    <>
      <h1>Hello there user {id}</h1>
      <p>This is your awesome User Profile page</p>
    </>
  );
}
```

6. Route and switch group all routes together to make sure that they take precedence over each other.

The following routes should be there in your App.js file:

```
import React from "react"
import { Route, Switch } from "react-router-dom"
// We will create these two pages in a moment
import HomePage from "./pages/HomePage"
import UserPage from "./pages/UserPage"
export default function App() {
  return (
    <Switch>
      <Route exact path="/" component={HomePage} />
      <Route path="/:id" component={UserPage} />
    </Switch>
  )
}
```

7. You can link to a page inside the SPA.

```
import React from "react"
import { Link } from "react-router-dom"
export default function HomePage() {
  return (
    <div className="container">
      <h1>Home </h1>
      <p>
        <Link to="/your desired link">Your desired link.</Link>
      </p>
    </div>
  )
}
```

4.a) Describe in detail Steps to create spring application

Steps to create spring application

Let's see the 5 steps to create the first spring application.

1) Create Java class

This is the simple java bean class containing the name property only.

```
package com.mrcet;
public class Student {
private String name;
public String getName() {
return name;
public void setName(String name) {
this.name = name;
}
public void displayInfo(){
System.out.println("Hello: "+name); } }
```

This is simple bean class, containing only one property name with its getters and setters
method. This class contains one extra method named displayInfo() that prints the student name by the hello message.

2) Create the xml file

In case of myeclipse IDE, you don't need to create the xml file as myeclipse does this for
yourselves. Open the applicationContext.xml file, and write the following code:

```
<?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:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
<bean id="studentbean" class="com.mrcet.Student">
<property name="name" value="Vimal Jaiswal"></property>
</bean>
</beans>
```

The **bean** element is used to define the bean for the given class.

The **property** subelement of bean specifies the property of the Student class named name. The value specified in the property element will be set in the Student class object
by the IOC container.

3) Create the test class

Create the java class e.g. Test. Here we are getting the object of Student class from the
IOC container using the getBean() method of BeanFactory. Let's see the code of test class.

```
package com.mrcet;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class Test {
public static void main(String[] args) {
Resource resource=new ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
Student student=(Student)factory.getBean("studentbean");
student.displayInfo();
```

The **Resource** object represents the information of applicationContext.xml file. The Resource is the interface and the **ClassPathResource** is the implementation class of the
Reource interface. The **BeanFactory** is responsible to return the bean.

The **XmlBeanFactory** is the implementation class of the BeanFactory. There are many
methods in the BeanFactory interface. One method is **getBean()**, which returns the object of the associated class.

4) Load the jar files required for spring framework

There are mainly three jar files required to run this application.

- **org.springframework.core-3.0.1.RELEASE-A**
- **com.springsource.org.apache.commons.logging-1.1.1**
- **org.springframework.beans-3.0.1.RELEASE-A**

For the future use, You can download the required jar files for spring core application.

download the core jar files for spring

download the all jar files for spring including core, web, aop, mvc, j2ee, remoting, oxm,
jdbc, orm etc.

To run this example, you need to load only spring core jar files.

5) Run the test class

Now run the Test class. You will get the output Hello: Vimal Jaiswal.

In command promt

D:\ cd fsp

D:\ cd fsp> javac -d . Student.java

D:\ cd fsp> javac -d . Test.java

D:\ cd fsp> java com.mrcet.Test

Output:

Hello: Vimal Jaiswal

b) Write a program to update a record using Spring JDBC

```
package com.mrcet;
import java.util.List;
import javax.sql.DataSource;
public interface StudentDAO {
/**
* This is the method to be used to initialize
* database resources ie. connection.
*/
public void setDataSource(DataSource ds);
/**
* This is the method to be used to update
* a record into the Student table.
*/
public void update(Integer id, Integer age);
/**
* This is the method to be used to list down
* a record from the Student table corresponding
* to a passed student id.
*/
public Student getStudent(Integer id);
}
```

Following is the content of the **Student.java** file.

```
package com.mrcet;
public class Student {
private Integer age;
private String name;
private Integer id;
public void setAge(Integer age) {
this.age = age;
}
public Integer getAge() {
return age;
}
public void setName(String name) {
this.name = name;
```

```
}
public String getName() {
return name;
}
public void setId(Integer id) {
this.id = id;
}
public Integer getId() {
return id;
}
}
```

Following is the content of the **StudentMapper.java** file.

```
package com.mrcet;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
public class StudentMapper implements RowMapper<Student> {
public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
Student student = new Student();
student.setId(rs.getInt("id"));
student.setName(rs.getString("name"));
student.setAge(rs.getInt("age"));
return student;
}
}
```

Following is the implementation class file **StudentJDBCTemplate.java** for the defined
DAO interface StudentDAO.

```
package com.mrcet;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
public class StudentJDBCTemplate implements StudentDAO {
private DataSource dataSource;
private JdbcTemplate jdbcTemplateObject;
public void setDataSource(DataSource dataSource) {
this.dataSource = dataSource;
this.jdbcTemplateObject = new JdbcTemplate(dataSource);
}
public void update(Integer id, Integer age){
String SQL = "update Student set age = ? where id = ?";
jdbcTemplateObject.update(SQL, age, id);
System.out.println("Updated Record with ID = " + id );
return;
}
public Student getStudent(Integer id) {
String SQL = "select * from Student where id = ?";
Student student = jdbcTemplateObject.queryForObject(
SQL, new Object[]{id}, new StudentMapper()
);
```

```
return student;
}
}
```

Following is the content of the **MainApp.java** file.

```
package com.mrcet;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.mrcet.StudentJDBCTemplate;
public class MainApp {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
StudentJDBCTemplate studentJDBCTemplate =
(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
System.out.println("----Updating Record with ID = 2 -----" );
studentJDBCTemplate.update(2, 20);
System.out.println("----Listing Record with ID = 2 -----" );
Student student = studentJDBCTemplate.getStudent(2);
System.out.print("ID : " + student.getId() );
System.out.print(", Name : " + student.getName() );
System.out.println(", Age : " + student.getAge());
}
}
```

Following is the configuration file **Beans.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"
xsi:schemaLocation = "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">
<!-- Initialization for data source -->
<bean id = "dataSource"
class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name = "driverClassName" value = "com.mysql.cj.jdbc.Driver"/>
<property name = "url" value = "jdbc:mysql://localhost:3306/TEST"/>
<property name = "username" value = "root"/>
<property name = "password" value = "admin"/>
</bean>
<!-- Definition for studentJDBCTemplate bean -->
<bean id = "studentJDBCTemplate"
class = "com.mrcet.StudentJDBCTemplate">
<property name = "dataSource" ref = "dataSource" />
</bean>
</beans>
```

Output

```
----Updating Record with ID = 2 -----
Updated Record with ID = 2
----Listing Record with ID = 2 -----
ID : 2, Name : Nuha, Age : 20
```

5-a) Write a program to delete a record using Spring JDBC

```
package com.mrcet;
import java.util.List;
import javax.sql.DataSource;
public interface StudentDAO {
/**
* This is the method to be used to initialize
* database resources ie. connection.
*/
public void setDataSource(DataSource ds);
/**
* This is the method to be used to list down
* all the records from the Student table.
*/
public List<Student> listStudents();
/**
* This is the method to be used to delete
* a record from the Student table corresponding
* to a passed student id.
*/
public void delete(Integer id);
}
```

Following is the content of the **Student.java** file.

```
package com.mrcet;
public class Student {
private Integer age;
private String name;
private Integer id;
public void setAge(Integer age) {
this.age = age;
}
public Integer getAge() {
return age;
}
public void setName(String name) {
this.name = name;
}
public String getName() {
return name;
}
public void setId(Integer id) {
this.id = id;
}
public Integer getId() {
return id;
}
}
```

Following is the content of the **StudentMapper.java** file.

```
package com.mrcet;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
public class StudentMapper implements RowMapper<Student> {
public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
Student student = new Student();
student.setId(rs.getInt("id"));
student.setName(rs.getString("name"));
student.setAge(rs.getInt("age"));
return student;
}
}
```

Following is the implementation class file **StudentJDBCTemplate.java** for the defined
DAO interface StudentDAO.

```
package com.mrcet;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
public class StudentJDBCTemplate implements StudentDAO {
private DataSource dataSource;
private JdbcTemplate jdbcTemplateObject;
public void setDataSource(DataSource dataSource) {
this.dataSource = dataSource;
this.jdbcTemplateObject = new JdbcTemplate(dataSource);
}
public List<Student> listStudents() {
String SQL = "select * from Student";
List <Student> students = jdbcTemplateObject.query(SQL, new StudentMapper());
return students;
}
public void delete(Integer id){
String SQL = "delete from Student where id = ?";
jdbcTemplateObject.update(SQL, id);
System.out.println("Deleted Record with ID = " + id );
return;
}
}
```

Following is the content of the **MainApp.java** file.

```
package com.mrcet;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.mrcet.StudentJDBCTemplate;
public class MainApp {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
StudentJDBCTemplate studentJDBCTemplate =
(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
System.out.println("----Delete Record with ID = 2 -----" );
studentJDBCTemplate.delete(2);
```

```
System.out.println("------Listing Multiple Records--------" );
List<Student> students = studentJDBCTemplate.listStudents();
for (Student record : students) {
System.out.print("ID : " + record.getId() );
System.out.print(", Name : " + record.getName() );
System.out.println(", Age : " + record.getAge());
}
}
}
```

Following is the configuration file **Beans.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"
xsi:schemaLocation = "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">
<!-- Initialization for data source -->
<bean id = "dataSource"
class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name = "driverClassName" value = "com.mysql.cj.jdbc.Driver"/>
<property name = "url" value = "jdbc:mysql://localhost:3306/TEST"/>
<property name = "username" value = "root"/>
<property name = "password" value = "admin"/>
</bean>
<!-- Definition for studentJDBCTemplate bean -->
<bean id = "studentJDBCTemplate"
class = "com.mrcet.StudentJDBCTemplate">
<property name = "dataSource" ref = "dataSource" />
</bean>
</beans>
```

Output

```
----Updating Record with ID = 2 -----
Updated Record with ID = 2
----Listing Record with ID = 2 -----
ID : 2, Name : Nuha, Age : 20
```

6.a) Write a program to read records using Spring JDBC

b) Describe Different types Of cloud computing deployment models

Different Types of Cloud Computing Deployment Models

Most cloud hubs have tens of thousands of servers and storage devices to enable fast
loading. It is often possible to choose a geographic area to put the data "closer" to users.

Thus, deployment models for cloud computing are categorized based on their location.
To know which model would best fit the requirements of your organization, let us first
learn about the various types.

Public Cloud

The name says it all. It is accessible to the public. Public deployment models in the cloud
are perfect for organizations with growing and fluctuating demands. It also makes a great
choice for companies with low-security concerns. Thus, you pay a cloud service provider
for networking services, compute virtualization & storage available on the public internet.
It is also a great delivery model for the teams with development and testing. Its configuration and deployment are quick and easy, making it an ideal choice for test environments.

Private Cloud

Now that you understand what the public cloud could offer you, of course, you are keen
to know what a private cloud can do. Companies that look for cost efficiency and greater
control over data & resources will find the private cloud a more suitable choice.
It means that it will be integrated with your data center and managed by your IT team.
Alternatively, you can also choose to host it externally. The private cloud offers bigger
opportunities that help meet specific organizations' requirements when it comes to customization. It's also a wise choice for mission-critical processes that may have frequently changing requirements.

Community Cloud

The community cloud operates in a way that is similar to the public cloud. There's just one
difference - it allows access to only a specific set of users who share common objectives
and use cases. This type of deployment model of cloud computing is managed and hosted

internally or by a third-party vendor. However, you can also choose a combination of all three

Hybrid Cloud

As the name suggests, a hybrid cloud is a combination of two or more cloud architectures.

While each model in the hybrid cloud functions differently, it is all part of the same architecture. Further, as part of this deployment of the cloud computing model, the internal or external providers can offer resources.

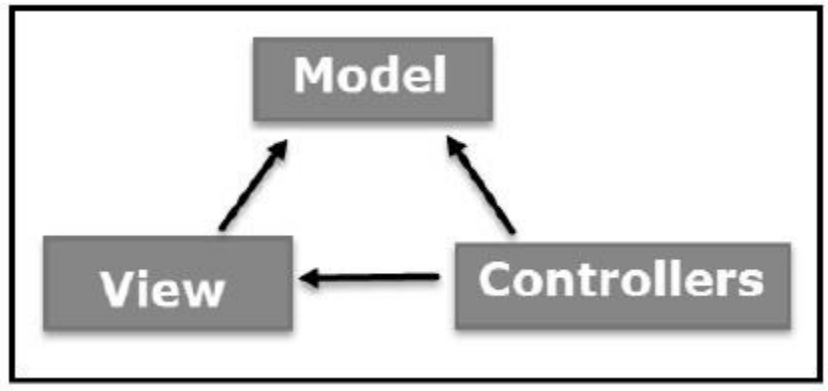
Let's understand the hybrid model better. A company with critical data will prefer storing on a private cloud, while less sensitive data can be stored on a public cloud. The hybrid cloud is also frequently used for 'cloud bursting'. It means, supposes an organization runs an application on-premises, but due to heavy load, it can burst into the public cloud.

8.a) Explain the Spring Web MVC Framework Example

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the **model**, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

MVC Components

Following are the components of MVC –



Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.