

# **Laboratory Manual For Full Stack Development**

**(IT-622)**

**B.Tech (IT)  
SEM VI**



**Faculty of Technology  
Dharmsinh Desai University**

**Nadiad.  
[www.ddu.ac.in](http://www.ddu.ac.in)**

# TABLE OF CONTENTS

[illegible]

# TABLE OF CONTENTS

[illegible]

# TABLE OF CONTENTS

[illegible]

# TABLE OF CONTENTS

[illegible]

# Experiment: 1

**AIM:** Implement a JavaScript based app to demonstrate the use of manipulating DOM nodes and their attributes using DOM API within a web page created using HTML and CSS.

## index.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="style/style.css">
</head>
<body>
  <div class="centerdiv">
    <table>
      <tr class="rows">
        <td>
          <div>Enter Number 1 : </div>
        </td>
        <td>
          <input class="inp" id="num1" type="text">
        </td>
      </tr>
      <tr class="rows">
        <td>
          <div>Enter Number 2 : </div>
        </td>
        <td>
          <input class="inp" id="num2" type="text">
        </td>
      </tr>
      <tr class="rows">
        <td>
          <div>Ans : </div>
        </td>
        <td>
          <input disabled class="inp" id="ans" type="text">
        </td>
      </tr>
    </table>
    <div class="buttons">
      <button onclick="add()"><p class="text">Addition</p></button>
      <button onclick="sub()"><p class="text">Subtraction</p></button>
    </div>
  </div>
</body>
</html>
```

```

        <button onclick="mul()"><p class="text">Multiplication</p></button>
        <button onclick="div()"><p class="text">Division</p></button>
    </div>
</div>

<script src="script/script.js"></script>
</body>
</html>

```

### style.css :

```

@font-face {
    font-family: open;
    src: url(/fonts/OpenSans-Italic-VariableFont_wdth\,wght.ttf);
}

@font-face {
    font-family: Source;
    src: url(/fonts/SourceSansPro-Light.ttf);
}

body{
    background: #100c08;
    display: flex;
    font-family: open;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    height: 97vh;
}

.rows{
    display: flex;
    flex-direction: row;
    gap: 20px;
    align-items: center;
    justify-content: flex-end;
    padding: 5px 0;
}

.inp {
    border: none;
    outline: none;
    border-radius: 15px;
    padding: 0.5em 1em;
    background-color: #007ACC;
    color: white;
    box-shadow: inset 2px 5px 10px rgba(0,0,0,0.3);
    transition: 300ms ease-in-out;
}

```

```

}

.inp:focus {
  background-color: white;
  color: black;
  transform: scale(1.05);
  box-shadow: 13px 13px 100px #969696,
              -13px -13px 100px #ffffff;
}

.buttons{
  display: flex;
  gap: 10px;
  margin: 10px 0;
}

button {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 0px 15px;
  gap: 15px;
  background-color: #007ACC;
  outline: 3px #007ACC solid;
  outline-offset: -3px;
  border-radius: 5px;
  border: none;
  cursor: pointer;
  transition: 400ms;
}

button .text {
  color: white;
  font-weight: 700;
  font-size: 1em;
  transition: 400ms;
}

.centerdiv{
  background: lightgrey;
  padding: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
  border-radius: 15px;
}

```

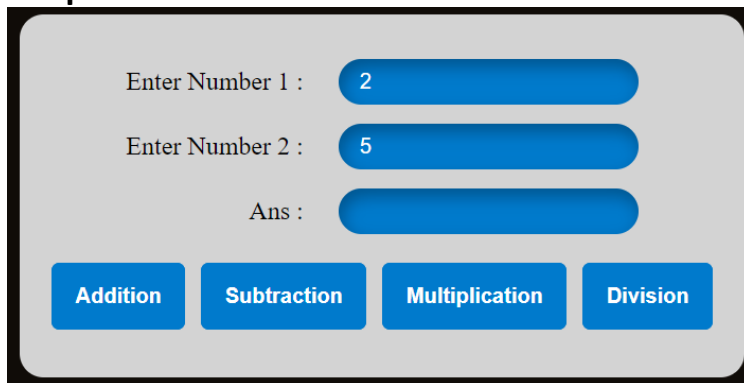


```
button:hover {  
    background-color: transparent;  
}  
  
button:hover .text {  
    color: #007ACC;  
}
```

### **script.js :**

```
const num1 = document.getElementById("num1");  
const num2 = document.getElementById("num2");  
const ans = document.getElementById("ans");  
  
function add(){  
    const a = num1.value;  
    const b = num2.value;  
  
    ans.value = parseInt(a)+parseInt(b);  
    alert("Addition is " + ans.value);  
}  
  
function sub(){  
    const a = num1.value;  
    const b = num2.value;  
  
    ans.value = parseInt(a)-parseInt(b);  
    alert("Subtraction is " + ans.value);  
}  
  
function mul(){  
    const a = num1.value;  
    const b = num2.value;  
  
    ans.value = parseInt(a)*parseInt(b);  
    alert("Multiplication is " + ans.value);  
}  
  
function div(){  
    const a = num1.value;  
    const b = num2.value;  
  
    ans.value = parseInt(a)/parseInt(b);  
    alert("Division is " + ans.value);  
}
```

## Output :



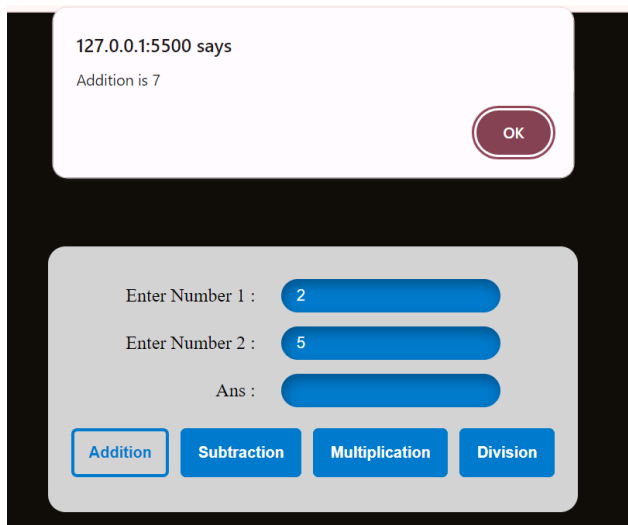
Enter Number 1 :

Enter Number 2 :

Ans :

**Addition**   **Subtraction**   **Multiplication**   **Division**

## onClick Addition:



127.0.0.1:5500 says  
Addition is 7

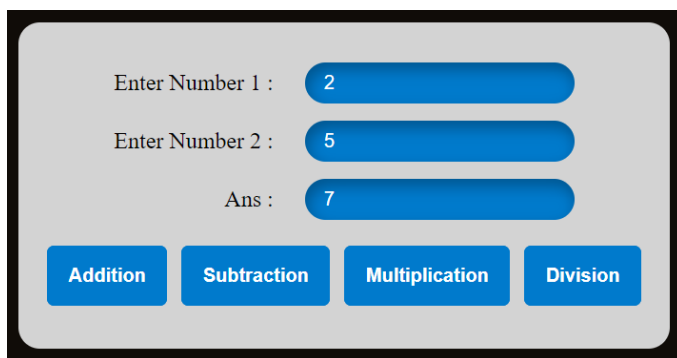
**OK**

Enter Number 1 :

Enter Number 2 :

Ans :

**Addition**   **Subtraction**   **Multiplication**   **Division**

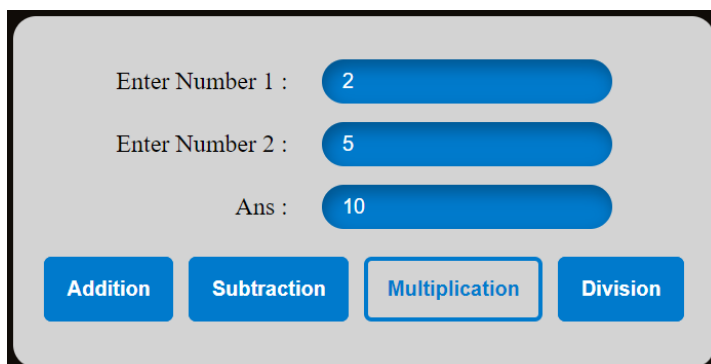


Enter Number 1 :

Enter Number 2 :

Ans :

**Addition**   **Subtraction**   **Multiplication**   **Division**



Enter Number 1 :

Enter Number 2 :

Ans :

**Addition**   **Subtraction**   **Multiplication**   **Division**

## Experiment: 2

**AIM:** Implement a JavaScript based app using modern features: arrow function, promise, async, and await and understand troubleshooting using Browser's Developer Tools..

### index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Display Countries and States</title>
  </head>
  <body>
    <h1>Countries and States</h1>
    <label for="countrySelect">Select a country:</label>
    <select id="countrySelect"></select>

    <div id="statesContainer"></div>

    <script src="script/script.js"></script>
  </body>
</html>
```

### script.js:

```
document.addEventListener("DOMContentLoaded", async function () {
  try {
    const response = await fetch(
      "https://countriesnow.space/api/v0.1/countries/states"
    );
    const data = await response.json();

    const countrySelect = document.getElementById("countrySelect");
    const statesContainer = document.getElementById("statesContainer");

    countrySelect.addEventListener("change", function () {
      statesContainer.innerHTML = "";
      const selectedCountry = data.data.find(
        (country) => country.name === countrySelect.value
      );
      const statesList = document.createElement("ul");
      if (selectedCountry && selectedCountry.states) {
        selectedCountry.states.forEach((state) => {
          const listItem = document.createElement("li");
          listItem.textContent = state.name;
          statesContainer.appendChild(listItem);
        });
      }
    });
  } catch (error) {
    console.error("Error fetching data:", error);
  }
});
```

```

    });
  }
});

data.data.forEach((country) => {
  const option = document.createElement("option");
  option.value = country.name;
  option.text = country.name;
  countrySelect.appendChild(option);
});
} catch (error) {
  console.error("Error fetching data:", error);
}
});

```

### Output:

## Countries and States

Select a country:

## Countries and States

Select a country:

- Australian Capital Territory
- New South Wales
- Northern Territory
- Queensland
- South Australia
- Tasmania
- Victoria
- Western Australia

## Experiment: 3

**AIM:** Implement a React JS based app using functional components to demonstrate use of the following (1) JSX, conditional rendering, lists, and props, (2) event handling of form fields with their state managed using useState.

### App.js:

```
import './App.css';
import { useState } from 'react';

const allIntersts = ["C", "C++", "Java", "Python", "Dart", "Javascript"];
const initialState = {
  username: "",
  password: "",
  gender: "",
  hindi: false,
  gujarati: false,
  english: false,
  intersts: [],
};

function App() {
  const [state, setState] = useState(initialState);

  const handleChange = (event) => {
    const value =
      event.target.type === "checkbox"
        ? event.target.checked
        : event.target.value;
    setState({ ...state, [event.target.name]: value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(state);
  };

  const handleSelectChange = (event) => {
    const selectedOptions = Array.from(
      event.target.selectedOptions,
      (option) => option.value
    );

    setState({ ...state, intersts: selectedOptions });
  };
}
```

```

return (
  <div>
    <form onSubmit={handleSubmit}>
      <div>
        <label>Enter Username : </label>
        <input
          type="text"
          name="username"
          value={state.username}
          onChange={handleChange}
        />
      </div>

      <div>
        <label>Enter Password : </label>
        <input
          type="text"
          name="password"
          value={state.password}
          onChange={handleChange}
        />
      </div>

      <div>
        <label>Select Gender : </label>
        <input
          type="radio"
          name="gender"
          value="Male"
          checked={state.gender === "Male"}
          onChange={handleChange}
        />
        <label>Male</label>
        <input
          type="radio"
          name="gender"
          value="Female"
          checked={state.gender === "Female"}
          onChange={handleChange}
        />
        <label>Female</label>
      </div>

      <div>
        <label>Known Languages : </label>
        <input
          type="checkbox"
          name="hindi"

```

```

        checked={state.hindi}
        onChange={handleChange}
      />
      <label>Hindi</label>

      <input
        type="checkbox"
        name="gujarati"
        checked={state.gujarati}
        onChange={handleChange}
      />
      <label>Gujarati</label>

      <input
        type="checkbox"
        name="english"
        checked={state.english}
        onChange={handleChange}
      />
      <label>English</label>
    </div>
    <div>
      <button type="submit">Submit</button>
    </div>

    <div>
      <label>Interests : </label>
      <select
        multiple
        name="interests"
        value={state.intersts}
        onChange={handleSelectChange}
      >
        {allIntersts.map((interest) => (
          <option value={interest} key={interest}>
            {interest}
          </option>
        ))}
      </select>
    </div>
  </form>
</div>
<div>
  <div>Username is : {state.username}</div>
  <div>Password is : {state.password}</div>
  <div>Gender is : {state.gender}</div>
  <div>Knows Hindi : {"" + state.hindi}</div>
  <div>Knows Gujarati : {"" + state.gujarati}</div>
  <div>Knows English : {"" + state.english}</div>

```

```

        <div>Interest : {"" + state.intersts}</div>
    </div>
</div>
);
}

export default App;

```

## Output:

Enter Username :   
 Enter Password :   
 Select Gender : ☐ Male ☒ Female  
 Known Languages : ☒ Hindi ☒ Gujarati ☒ English  
  

C

C++

Java

Python

 Interests :   
 Username is : Hetvi  
 Password is : hetvi123  
 Gender is : Female  
 Knows Hindi : true  
 Knows Gujarati : true  
 Knows English : true  
 Interest : C++,Java,Python



## Experiment: 4

**AIM:** Implement a React JS based app using functional components to demonstrate the use of the following: (1) useEffect with calling API using axios, (2) context API with reducer.

```
import React, { useEffect, useReducer } from 'react';
import axios from 'axios';

const initialState = {
  data: [],
  loading: true,
  error: null,
};

const reducer = (state, action) => {
  switch (action.type) {
    case 'FETCH_SUCCESS':
      return {
        ...state,
        data: action.payload,
        loading: false,
        error: null,
      };
    case 'FETCH_ERROR':
      return {
        ...state,
        data: [],
        loading: false,
        error: action.payload,
      };
    default:
      return state;
  }
};

const DataContext = React.createContext();

const DataProvider = ({ children }) => {
  const [state, dispatch] = useReducer(reducer, initialState);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await
        axios.get('https://jsonplaceholder.typicode.com/posts');

```

```

        dispatch({ type: 'FETCH_SUCCESS', payload: response.data });
      } catch (error) {
        dispatch({ type: 'FETCH_ERROR', payload: error.message });
      }
    };

    fetchData();

  }, []);

  return (
    <DataContext.Provider value={state}>
      {children}
    </DataContext.Provider>
  );
};

const DataConsumer = () => {
  const { data, loading, error } = React.useContext(DataContext);

  if (loading) {
    return <div>Loading...</div>;
  }

  if (error) {
    return <div>Error: {error}</div>;
  }

  return (
    <div>
      <h1>Data</h1>
      <ul>
        {data.map((item, index) => (
          <li key={item.id}>Title{index+1} : {item.title}</li>
        ))}
      </ul>
    </div>
  );
};

const App = () => {
  return (
    <div>
      <h1>React App</h1>
      <DataProvider>
        <DataConsumer />
      </DataProvider>
    </div>
  );
};

```

```
);  
};  
  
export default App;
```

## Output:

# React App

## Data

- Title1 : sunt aut facere repellat provident occaecati excepturi optio reprehenderit
- Title2 : qui est esse
- Title3 : ea molestias quasi exercitationem repellat qui ipsa sit aut
- Title4 : eum et est occaecati
- Title5 : nesciunt quas odio
- Title6 : dolore eum magni eos aperiam quia
- Title7 : magnam facilis autem
- Title8 : dolore dolore est ipsam
- Title9 : nesciunt iure omnis dolore tempora et accusantium
- Title10 : optio molestias id quia eum
- Title11 : et ea vero quia laudantium autem
- Title12 : in quibusdam tempore odit est dolore
- Title13 : dolorum ut in voluptas mollitia et saepe quo animi
- Title14 : voluptatem eligendi optio
- Title15 : eveniet quod temporibus
- Title16 : sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio
- Title17 : fugit voluptas sed molestias voluptatem provident
- Title18 : voluptate et itaque vero tempora molestiae
- Title19 : adipisci placeat illum aut reiciendis qui
- Title20 : doloribus ad provident suscipit at
- Title21 : asperiores ea ipsam voluptatibus modi minima quia sint
- Title22 : dolor sint quo a velit explicabo quia nam

# Experiment: 7

**AIM:** Implement a hibernate based application to demonstrate CRUD operations using DAO pattern.

## Hibernate.properties :

```
hibernate.connection.url=jdbc:mysql://localhost/test
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.connection.username=root
hibernate.connection.password=mysqlpassword
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.hbm2ddl.auto=create
```

## StudentDAO.java :

```
package org.example.dao;

import org.example.entities.Student;
import java.util.List;

public interface StudentDAO {
    void saveStudent(Student student);
    Student getStudentById(long id);
    List<Student> getAllStudents();
    void updateStudent(Student student);
    void deleteStudent(long id);
}
```

## StudentDAOImpl.java :

```
package org.example.dao;

import org.hibernate.query.Query;
import org.example.entities.Student;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import java.util.List;

public class StudentDAOImpl implements StudentDAO{
    private final SessionFactory sessionFactory;

    public StudentDAOImpl(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
}
```

```

    }

    @Override
    public void saveStudent(Student student) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();
            session.persist(student);
            transaction.commit();
        }
    }

    @Override
    public Student getStudentById(long id) {
        try (Session session = sessionFactory.openSession()) {
            return session.get(Student.class, id);
        }
    }

    @Override
    public List<Student> getAllStudents() {
        try (Session session = sessionFactory.openSession()) {
            Query<Student> query = session.createQuery("FROM Student",
Student.class);
            return query.list();
        }
    }

    @Override
    public void updateStudent(Student student) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();
            Student existingStudent = session.get(Student.class,
student.getId());
            existingStudent.setStudentName(student.getStudentName());
            transaction.commit();
        }
    }

    @Override
    public void deleteStudent(long id) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();
            Student student = session.get(Student.class, id);
            if (student != null) {
                session.remove(student);
            }
            transaction.commit();
        }
    }

```

```
}  
}
```

### Student.java:

```
package org.example.entities;  
import jakarta.persistence.*;  
  
@Entity  
public class Student {  
    @Id @GeneratedValue  
    private long id;  
  
    @Column(name = "student_name")  
    private String studentName;  
  
    public long getId() {  
        return id;  
    }  
  
    public void setId(long id) {  
        this.id = id;  
    }  
  
    public String getStudentName() {  
        return studentName;  
    }  
  
    public void setStudentName(String studentName) {  
        this.studentName = studentName;  
    }  
  
    @Override  
    public String toString() {  
        return "Student{" +  
            "id=" + id +  
            ", studentName='" + studentName + '\'' +  
            '}';  
    }  
}
```

## Main.java :

```
package org.example;

import org.example.dao.StudentDAO;
import org.example.dao.StudentDAOImpl;
import org.example.entities.Student;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        Configuration config = new Configuration();
        config.addAnnotatedClass(org.example.entities.Student.class);
        SessionFactory sessionFactory = config.buildSessionFactory();

        StudentDAO studentDAO = new StudentDAOImpl(sessionFactory);

        Student student1= new Student();
        student1.setStudentName("Tom Cruise");
        Student student2= new Student();
        student2.setStudentName("Will Smith");

        System.out.println("Adding student " + student1);
        System.out.println("Adding student " + student2);
        studentDAO.saveStudent(student1);
        studentDAO.saveStudent(student2);

        List<Student> students = studentDAO.getAllStudents();
        System.out.println("Students: "+students.toString());

        Student studentToUpdate = studentDAO.getStudentById(1);
        System.out.println("Updating student " + studentToUpdate);
        if(studentToUpdate!=null){
            studentToUpdate.setStudentName("Tomkumar Cruise");
            studentDAO.updateStudent(studentToUpdate);
        }
        Student updatedStudent = studentDAO.getStudentById(1);
        System.out.println("Updated student = "+updatedStudent);

        studentDAO.deleteStudent(2);
        System.out.println("Deleting student with id = "+ 2);
        List<Student> studentsAfterDelete = studentDAO.getAllStudents();
        System.out.println("Students: "+studentsAfterDelete.toString());

        sessionFactory.close();
    }
}
```

## Output :

```
Hibernate:
    drop table if exists Student
Hibernate:
    drop table if exists Student_SEQ
Hibernate:
    create table Student (
        id bigint not null,
        student_name varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:
    create table Student_SEQ (
        next_val bigint
    ) engine=InnoDB
Hibernate:
    insert into Student_SEQ values ( 1 )
Adding student Student{id=0, studentName='Tom Cruise'}
Adding student Student{id=0, studentName='Will Smith'}
Hibernate:
    select
        next_val as id_val
    from
        Student_SEQ for update
Hibernate:
    update
        Student_SEQ
    set
        next_val= ?
    where
        next_val=?
Hibernate:
    insert
    into
        Student
        (student_name, id)
    values
        (?, ?)
```



```
Hibernate:
  select
    next_val as id_val
  from
    Student_SEQ for update
```

```
Hibernate:
  update
    Student_SEQ
  set
    next_val= ?
  where
    next_val=?
```

```
Hibernate:
  insert
  into
    Student
    (student_name, id)
  values
    (?, ?)
```

```
Hibernate:
  select
    s1_0.id,
    s1_0.student_name
  from
    Student s1_0
```

Students: [Student{id=1, studentName='Tom Cruise'}, Student{id=2, studentName='Will Smith'}]

```
Hibernate:
  select
    s1_0.id,
    s1_0.student_name
  from
    Student s1_0
  where
    s1_0.id=?
```

Updating student Student{id=1, studentName='Tom Cruise'}

```
Hibernate:
  select
    s1_0.id,
    s1_0.student_name
  from
    Student s1_0
  where
    s1_0.id=?
```

```

Hibernate:
    update
        Student
    set
        student_name=?
    where
        id=?
Hibernate:
    select
        s1_0.id,
        s1_0.student_name
    from
        Student s1_0
    where
        s1_0.id=?
Updated student = Student{id=1, studentName='Tomkumar Cruise'}
Hibernate:
    select
        s1_0.id,
        s1_0.student_name
    from
        Student s1_0
    where
        s1_0.id=?
Hibernate:
    delete
    from
        Student
    where
        id=?
Deleting student with id = 2
Hibernate:
    select
        s1_0.id,
        s1_0.student_name
    from
        Student s1_0
Students: [Student{id=1, studentName='Tomkumar Cruise'}]

```

student x student\_seq

Limit to 1000 rows

1 • `SELECT * FROM test.student;`

Result Grid

	id	student_name
▶	1	Tomkumar Cruise
*	NULL	NULL

student student\_seq x

Limit to 1000 rows

1 • `SELECT * FROM test.student_seq;`

Result Grid

	next_val
▶	101

# Experiment: 8

**AIM:** Implement a hibernate based application to demonstrate relationships among hibernate entities.

## 1. One-to-one:

### entities/Student.java:

```
package org.example.entities;

import jakarta.persistence.*;

@Entity
public class Student {
    @Id
    private long studentId;

    @Column(name = "student_name")
    private String studentName;

    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "Student_fk")
    private StudentDetails studentDetails;

    public long getStudentId() {
        return studentId;
    }

    public void setStudentId(long studentId) {
        this.studentId = studentId;
    }

    public StudentDetails getStudentDetails() {
        return studentDetails;
    }

    public void setStudentDetails(StudentDetails studentDetails) {
        this.studentDetails = studentDetails;
    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
```

```

        this.studentName = studentName;
    }

    @Override
    public String toString(){
        return studentId + " ----- " + studentName;
    }
}

```

## entities/StudentDetails.java :

```

package org.example.entities;

import jakarta.persistence.*;

@Entity
public class StudentDetails {
    @Id
    private Long studentDetailsId;
    @Column(name = "zip_code")
    private int zipCode;
    @OneToOne(cascade = CascadeType.ALL)
    private Student student;

    public Student getStudent() {
        return student;
    }

    public void setStudent(Student student) {
        this.student = student;
    }

    public Long getStudentDetailsId() {
        return studentDetailsId;
    }

    public void setStudentDetailsId(Long studentDetailsId) {
        this.studentDetailsId = studentDetailsId;
    }

    public int getZipCode() {
        return zipCode;
    }

    public void setZipCode(int zipCode) {
        this.zipCode = zipCode;
    }
}

```

## Main.java :

```
package org.example;

import org.example.entities.StudentDetails;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.example.entities.Student;

public class Main {
    public static void main(String[] args) {
        Configuration config = new Configuration();
        config.addAnnotatedClass(org.example.entities.Student.class);
        config.addAnnotatedClass(org.example.entities.StudentDetails.class);
        SessionFactory sessionFactory = config.buildSessionFactory();
        Session session = sessionFactory.openSession();

        try{
            session.beginTransaction();

            StudentDetails sd1 = new StudentDetails();
            sd1.setStudentDetailsId(101L);
            sd1.setZipCode(393002);

            StudentDetails sd2 = new StudentDetails();
            sd2.setStudentDetailsId(103L);
            sd2.setZipCode(393001);

            Student s1 = new Student();
            s1.setStudentId(1);
            s1.setStudentName("Hetvi");
            s1.setStudentDetails(sd2);

            Student s2 = new Student();
            s2.setStudentId(2);
            s2.setStudentName("Aneri");
            s2.setStudentDetails(sd1);

            sd1.setStudent(s2);
            sd2.setStudent(s1);

            session.persist(sd1);
            session.persist(sd2);
            session.persist(s1);
        }
    }
}
```

```
        session.persist(s2);
        session.getTransaction().commit();
    }finally {
        session.close();
    }
}
```

## Output:

```
Hibernate:
    alter table Student
        drop
        foreign key FKt09sfi9uty4vfv5w8c723w1ck
Hibernate:
    alter table StudentDetails
        drop
        foreign key FKm52cr45dq8q4jahn2wL5m5qgr
Hibernate:
    drop table if exists Student
Hibernate:
    drop table if exists StudentDetails
Hibernate:
    create table Student (
        Student_fK bigint,
        studentId bigint not null,
        student_name varchar(255),
        primary key (studentId)
    ) engine=InnoDB
```

```

Hibernate:
    create table StudentDetails (
        zip_code integer,
        studentDetailsId bigint not null,
        student_studentId bigint,
        primary key (studentDetailsId)
    ) engine=InnoDB
Hibernate:
    alter table Student
        add constraint UK_l8vw8g9foarwfc3r3s1kxw4pi unique (Student_fK)
Hibernate:
    alter table StudentDetails
        add constraint UK_s43twsu620qbpyc93tp571ypu unique (student_studentId)
Hibernate:
    alter table Student
        add constraint FKt09sfi9uty4vfv5w8c723w1ck
        foreign key (Student_fK)
        references StudentDetails (studentDetailsId)
Hibernate:
    alter table StudentDetails
        add constraint FKm52cr45dq8q4jahn2wl5m5qgr
        foreign key (student_studentId)
        references Student (studentId)
Hibernate:
    insert
    into
        Student
        (Student_fK, student_name, studentId)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        StudentDetails
        (student_studentId, zip_code, studentDetailsId)
    values
        (?, ?, ?)

```



```
Hibernate:
  insert
  into
    Student
    (Student_fK, student_name, studentId)
  values
    (?, ?, ?)
Hibernate:
  insert
  into
    StudentDetails
    (student_studentId, zip_code, studentDetailsId)
  values
    (?, ?, ?)
Hibernate:
  update
    Student
  set
    Student_fK=?,
    student_name=?
  where
    studentId=?
Hibernate:
  update
    Student
  set
    Student_fK=?,
    student_name=?
  where
    studentId=?
```

student x studentdetails

1 • `SELECT * FROM test.student;`

Result Grid Filter Rows: Edit:

	Student_fk	studentId	student_name
▶	103	1	Hetvi
	101	2	Aneri
*	NULL	NULL	NULL

student x studentdetails

1 • `SELECT * FROM test.studentdetails;`

Result Grid Filter Rows: Edit:

	zip_code	studentDetailsId	student_studentId
▶	393002	101	2
	393001	103	1
*	NULL	NULL	NULL

## 2. One-to-many & many-to-one :

### entities/Student.java:

```
package org.example.entities;

import jakarta.persistence.*;

@Entity
```

```

public class Student {
    @Id
    @Column(name = "student_id")
    private int studentId;

    @Column(name = "student_name")
    private String studentName;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "college_fk")
    private Student_College studentCollege;

    public Student_College getStudentCollege() {
        return studentCollege;
    }

    public void setStudentCollege(Student_College studentCollege) {
        this.studentCollege = studentCollege;
    }

    public int getStudentId() {
        return studentId;
    }

    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
}

```

### **entities/Student\_College :**

```

package org.example.entities;

import jakarta.persistence.*;

import java.util.List;

@Entity
public class Student_College {

```

```

@Id
@Column(name = "college_id")
private int collegeId;

@Column(name = "college_name")
private String collegeName;

@OneToMany(mappedBy = "studentCollege", cascade = CascadeType.ALL, fetch =
FetchType.EAGER)
private List<Student> students;

public List<Student> getStudents() {
    return students;
}

public void setStudents(List<Student> students) {
    this.students = students;
}

public int getCollegeId() {
    return collegeId;
}

public void setCollegeId(int collegeId) {
    this.collegeId = collegeId;
}

public String getCollegeName() {
    return collegeName;
}

public void setCollegeName(String collegeName) {
    this.collegeName = collegeName;
}
}

```

### **Main.java :**

```

package org.example;

import org.example.entities.Student;
import org.example.entities.Student_College;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.util.ArrayList;
import java.util.List;

public class Main {

```

```

public static void main(String[] args) {
    Configuration config = new Configuration();
    config.addAnnotatedClass(org.example.entities.Student.class);
    config.addAnnotatedClass(org.example.entities.Student_College.class);

    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.openSession();

    try{
        session.beginTransaction();

        Student_College college1 = new Student_College();
        college1.setCollegeId(101);
        college1.setCollegeName("DDIT");

        Student_College college2 = new Student_College();
        college2.setCollegeId(102);
        college2.setCollegeName("NIRMA");

        Student student1 = new Student();
        student1.setStudentId(1);
        student1.setStudentName("Hetvi");

        Student student2 = new Student();
        student2.setStudentId(2);
        student2.setStudentName("Aneri");

        Student student3 = new Student();
        student3.setStudentId(3);
        student3.setStudentName("Manisha");

        Student student4 = new Student();
        student4.setStudentId(4);
        student4.setStudentName("Krina");

        student1.setStudentCollege(college1);
        student2.setStudentCollege(college2);
        student3.setStudentCollege(college1);
        student4.setStudentCollege(college2);

        List<Student> ddit_students = new ArrayList<>();
        ddit_students.add(student1);
        ddit_students.add(student3);

        List<Student> nirma_students = new ArrayList<>();
        nirma_students.add(student2);
        nirma_students.add(student4);
    }
}

```

```

        college1.setStudents(ddit_students);
        college2.setStudents(nirma_students);

        session.persist(college1);
        session.persist(college2);

        session.getTransaction().commit();
    }finally {
        session.close();
        sessionFactory.close();
    }
}
}

```

## Output:

```

Hibernate:
    alter table Student
        drop
            foreign key FKeh0l8scst7dj0frqqprmv5l4y
Hibernate:
    drop table if exists Student
Hibernate:
    drop table if exists Student_College
Hibernate:
    create table Student (
        college_fk integer,
        student_id integer not null,
        student_name varchar(255),
        primary key (student_id)
    ) engine=InnoDB
Hibernate:
    create table Student_College (
        college_id integer not null,
        college_name varchar(255),
        primary key (college_id)
    ) engine=InnoDB
Hibernate:
    alter table Student
        add constraint FKeh0l8scst7dj0frqqprmv5l4y
        foreign key (college_fk)
        references Student_College (college_id)

```

```
Hibernate:
    insert
    into
        Student_College
        (college_name, college_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Student
        (college_fk, student_name, student_id)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Student
        (college_fk, student_name, student_id)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Student_College
        (college_name, college_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Student
        (college_fk, student_name, student_id)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Student
        (college_fk, student_name, student_id)
    values
        (?, ?, ?)
```

student x student\_college

1 • `SELECT * FROM test.student;`

Result Grid Filter Rows:  Edit

	college_fk	student_id	student_name
▶	101	1	Hetvi
	102	2	Aneri
	101	3	Manisha
	102	4	Krina
*	NULL	NULL	NULL

student student\_college x

1 • `SELECT * FROM test.student_colle`

Result Grid Filter Rows:  Edit

	college_id	college_name
▶	101	DDIT
	102	NIRMA
*	NULL	NULL



### 3. Many-to-many:

#### entities/Certification.java :

```
package org.example.entities;

import jakarta.persistence.*;
import java.util.Set;

@Entity
public class Certification {
    @Id
    @Column(name = "certification_id")
    private int certificationId;
    @Column(name = "certification_name")
    private String certificationName;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "Join_Student_Certification",
        joinColumns = {@JoinColumn(name = "certification_id_fk")},
        inverseJoinColumns = {@JoinColumn(name = "student_id_fk")}
    )
    Set<Student> students;
    public Set<Student> getStudents() {
        return students;
    }

    public void setStudents(Set<Student> students) {
        this.students = students;
    }

    public int getCertificationId() {
        return certificationId;
    }

    public void setCertificationId(int certificationId) {
        this.certificationId = certificationId;
    }

    public String getCertificationName() {
        return certificationName;
    }

    public void setCertificationName(String certificationName) {
        this.certificationName = certificationName;
    }
}
```

## entities/Student.java :

```
package org.example.entities;

import jakarta.persistence.*;
import java.util.Set;

@Entity
public class Student {
    @Id
    @Column(name = "student_id")
    private int studentId;

    @Column(name = "student_name")
    private String studentName;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "Join_Student_Certification",
        joinColumns = {@JoinColumn(name = "student_id_fk")},
        inverseJoinColumns = {@JoinColumn(name = "certification_id_fk")})
    private Set<Certification> studentCertificates;

    public Set<Certification> getStudentCertificates() {
        return studentCertificates;
    }

    public void setStudentCertificates(Set<Certification> studentCertificates)
    {
        this.studentCertificates = studentCertificates;
    }

    public int getStudentId() {
        return studentId;
    }

    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
}
```

## Main.java:

```
package org.example;

import org.example.entities.Certification;
import org.example.entities.Student;
import org.example.entities.Student_College;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Main {
    public static void main(String[] args) {
        Configuration config = new Configuration();
        config.addAnnotatedClass(org.example.entities.Student.class);
        config.addAnnotatedClass(org.example.entities.Certification.class);

        SessionFactory sessionFactory = config.buildSessionFactory();
        Session session = sessionFactory.openSession();

        try{
            session.beginTransaction();

            Student student1 = new Student();
            student1.setStudentId(1);
            student1.setStudentName("Hetvi");

            Student student2 = new Student();
            student2.setStudentId(2);
            student2.setStudentName("Aneri");

            Student student3 = new Student();
            student3.setStudentId(3);
            student3.setStudentName("Manisha");

            Student student4 = new Student();
            student4.setStudentId(4);
            student4.setStudentName("Krina");

            Certification aws = new Certification();
            aws.setCertificationId(101);
            aws.setCertificationName("AWS");

            Certification oracle = new Certification();
            oracle.setCertificationId(102);
```

```

        oracle.setCertificationName("Oracle");

        Certification machineLearning = new Certification();
        machineLearning.setCertificationId(103);
        machineLearning.setCertificationName("Machine Learning");

        Set<Certification> hetvicerti = new HashSet<>();
        hetvicerti.add(machineLearning);
        hetvicerti.add(aws);

        Set<Certification> aneriCerti = new HashSet<>();
        aneriCerti.add(oracle);

        Set<Certification> manishaCerti = new HashSet<>();
        manishaCerti.add(oracle);
        manishaCerti.add(aws);

        Set<Certification> krinaCerti = new HashSet<>();
        krinaCerti.add(aws);
        krinaCerti.add(machineLearning);
        krinaCerti.add(oracle);

        student1.setStudentCertificates(hetvicerti);
        student2.setStudentCertificates(neriCerti);
        student3.setStudentCertificates(manishaCerti);
        student4.setStudentCertificates(krinaCerti);

        session.persist(student1);
        session.persist(student2);
        session.persist(student3);
        session.persist(student4);

        session.getTransaction().commit();
    }finally {
        session.close();
        sessionFactory.close();
    }
}
}
}

```

## Output:

```

Hibernate:
    alter table Join_Student_Certification
    drop
    foreign key FKqljtfevgrm3uixxejehyns41c

```

```

Hibernate:
    alter table Join_Student_Certification
        drop
        foreign key FKrc0ls0h4h0l79cp0qej3rg6k4
Hibernate:
    drop table if exists Certification
Hibernate:
    drop table if exists Join_Student_Certification
Hibernate:
    drop table if exists Student
Hibernate:
    create table Certification (
        certification_id integer not null,
        certification_name varchar(255),
        primary key (certification_id)
    ) engine=InnoDB
Hibernate:
    create table Join_Student_Certification (
        certification_id_fk integer not null,
        student_id_fk integer not null,
        primary key (certification_id_fk, student_id_fk)
    ) engine=InnoDB
Hibernate:
    create table Student (
        student_id integer not null,
        student_name varchar(255),
        primary key (student_id)
    ) engine=InnoDB
Hibernate:
    alter table Join_Student_Certification
        add constraint FKqljtfevgrm3uixxejehyns41c
        foreign key (student_id_fk)
        references Student (student_id)
Hibernate:
    alter table Join_Student_Certification
        add constraint FKrc0ls0h4h0l79cp0qej3rg6k4
        foreign key (certification_id_fk)
        references Certification (certification_id)

```

```
Hibernate:
    insert
    into
        Student
        (student_name, student_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Certification
        (certification_name, certification_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Certification
        (certification_name, certification_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Student
        (student_name, student_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Certification
        (certification_name, certification_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Student
        (student_name, student_id)
    values
        (?, ?)
```

```
Hibernate:
    insert
    into
        Student
        (student_name, student_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
```

```

Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)
Hibernate:
    insert
    into
        Join_Student_Certification
        (student_id_fk, certification_id_fk)
    values
        (?, ?)

```

The screenshot shows a database management tool interface. At the top, there are three tabs: 'certification', 'join\_student\_certification', and 'student'. The 'certification' tab is active. Below the tabs is a toolbar with various icons. The main area displays a SQL query: `SELECT * FROM test.certification;`. Below the query is a 'Result Grid' section. It includes a 'Filter Rows' input field and an 'Edit' button. The result grid contains a table with two columns: 'certification\_id' and 'certification\_name'. The table has four rows: three data rows and one row with 'NULL' values.

certification_id	certification_name
101	AWS
102	Orade
103	Machine Learning
NULL	NULL



certification join\_student\_certification student

1 • `SELECT * FROM test.student;`

Result Grid Filter Rows: Edit:

	student_id	student_name
▶	1	Hetvi
	2	Aneri
	3	Manisha
	4	Krina
•	NULL	NULL

certification join\_student\_certification student

1 • `SELECT * FROM test.join_student_certification;`

Limit to 1000 rows

Result Grid Filter Rows: Edit:

	certification_id_fk	student_id_fk
▶	101	1
	103	1
	102	2
	101	3
	102	3
	101	4
	102	4
	103	4
•	NULL	NULL

# Experiment: 9

**AIM:** Implement a SpringMVC based application to perform form handling with data binding and its use in JSP.

## config/MySpringMvcAppInitializer.java:

```
package com.example.config;

import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class MySpringMvcAppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {

        //Load Spring Web application configuration
        AnnotationConfigWebApplicationContext context = new
AnnotationConfigWebApplicationContext();
        //register beans config class
        context.register(WebApplicationContextConfig.class);

        //Create and register the DispatcherServlet
        DispatcherServlet servlet = new DispatcherServlet(context);
        ServletRegistration.Dynamic registration =
servletContext.addServlet("dispatcher", servlet);
        registration.setLoadOnStartup(1);
        registration.addMapping("/");
    }
}
```

## config/WebApplicationContextConfig.java:

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
```

```

//substitutue of root beans element
@Configuration
//substitutue of <mvc:annotation-driver>
@EnableWebMvc
//substitute of <context:component-scan>
@ComponentScan("com.example")
public class WebApplicationContextConfig implements WebMvcConfigurer {

    //substitute of viewResolver bean in .xml file
    @Bean
    public ViewResolver getViewResolver(){
        InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/jsp/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}

```

## Patient.java:

```

package com.example;

import jakarta.validation.Valid;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Past;
import jakarta.validation.constraints.Pattern;
import org.springframework.format.annotation.DateTimeFormat;

import java.util.Date;
import java.util.List;

public class Patient {

    private String patientName;
    private String patientContact;
    private String patientGender;
    private Date patientDoB;
    private List<String> patientAllergies;
    private PatientAddress patientAddress;

    public String getPatientName() {
        return patientName;
    }

    public void setPatientName(String patientName) {
        this.patientName = patientName;
    }
}

```

```

public String getPatientContact() {
    return patientContact;
}

public void setPatientContact(String patientContact) {
    this.patientContact = patientContact;
}

public String getPatientGender() {
    return patientGender;
}

public void setPatientGender(String patientGender) {
    this.patientGender = patientGender;
}

public Date getPatientDoB() {
    return patientDoB;
}

public void setPatientDoB(Date patientDoB) {
    this.patientDoB = patientDoB;
}

public List<String> getPatientAllergies() {
    return patientAllergies;
}

public void setPatientAllergies(List<String> patientAllergies) {
    this.patientAllergies = patientAllergies;
}

public PatientAddress getPatientAddress() {
    return patientAddress;
}

public void setPatientAddress(PatientAddress patientAddress) {
    this.patientAddress = patientAddress;
}
}

```

## PatientAddress.java:

```
package com.example;

import jakarta.validation.constraints.*;

public class PatientAddress {
    private String street;
    private String city;
    private String state;
    private String country;
    private int pincode;

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public int getPincode() {
        return pincode;
    }
}
```

```

        public void setPincode(int pincode) {
            this.pincode = pincode;
        }
    }
}

```

## PatientController.java:

```

package com.example;

import jakarta.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

import java.util.Arrays;
import java.util.Map;

@Controller
@SessionAttributes({"patient"})
public class PatientController {
    @RequestMapping("/appointment")
    public ModelAndView appointment(Model model){
        Patient patient = new Patient();
        patient.setPatientGender("Male");
        model.addAttribute("countryList",
Arrays.asList("India", "USA", "Canada", "UK"));
        model.addAttribute("allergyList",Arrays.asList("Peanuts", "Dust", "Smoke
"));

        return new ModelAndView("appointment","patient",patient);
    }

    @PostMapping("/addappointment")
    public ModelAndView addAppointment(@Valid @ModelAttribute Patient patient,
BindingResult bindingResult, Model model)
    {
        if(bindingResult.hasErrors()){
            model.addAttribute("countryList",
Arrays.asList("India", "USA", "Canada", "UK"));
            model.addAttribute("allergyList",Arrays.asList("Peanuts", "Dust", "S
moke"));

            return new ModelAndView("appointment","patient",patient);
        }
        ModelAndView modelAndView = new ModelAndView("addappointment");
        modelAndView.addObject("message",

```

```

        "We have registered your details"
    );

    return modelAndView;
}

@ModelAttribute
public void addingCommonObjects(Model model){
    model.addAttribute("mainHeader","Welcome to the best Clinic");
}
}

```

### jsp/appointment.jsp:

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <title>Patient Appointment Registration</title>
</head>
<body>
<h1>${mainHeader}</h1>
<h2>patient Appointment Registration</h2>
<form:form method="post" action="addappointment" modelAttribute="patient">
    <table>
        <tr>
            <td>Name:</td>
            <td>
                <form:input type="text" name="patientName"
path="patientName"/>
            </td>
        </tr>
        <tr>
            <td>Contact No:</td>
            <td>
                <form:input type="text" name="patientContact"
path="patientContact"/>
            </td>
        </tr>
        <tr>
            <td>Gender:</td>
            <td>
                <form:radiobutton path="patientGender" value="Male"
label="Male"/>
                <form:radiobutton path="patientGender" value="Female"
label="Female"/>
            </td>
        </tr>
    </table>

```

```

        <td>Date of Birth:</td>
        <td>
            <form:input type="date" path="patientDoB"/>
        </td>
    </tr>
    <tr>
        <td>Allergies:</td>
        <td>
            <form:checkboxes path="patientAllergies"
items="${allergyList}"/>
        </td>
    </tr>
    <tr>
        <td colspan="2">Address Details:</td>
    </tr>
    <tr>
        <td>Street:</td>
        <td>
            <form:input path="patientAddress.street" type="text"/>
        </td>
    </tr>
    <tr>
        <td>City:</td>
        <td>
            <form:errors path="patientAddress.city"
style="color:red"/><br/>
            <form:input path="patientAddress.city" type="text"/>
        </td>
    </tr>
    <tr>
        <td>State:</td>
        <td>
            <form:errors path="patientAddress.state"
style="color:red"/><br/>
            <form:input path="patientAddress.state" type="text"/>
        </td>
    </tr>
    <tr>
        <td>Country:</td>
        <td>
            <form:errors path="patientAddress.country"
style="color:red"/><br/>
            <form:select path="patientAddress.country">
                <form:options items="${countryList}"/>
            </form:select>
        </td>
    </tr>
    <tr>

```



```

        <td>Pincode:</td>
        <td>
            <form:errors path="patientAddress.pincode"
style="color:red"/><br/>
            <form:input path="patientAddress.pincode" type="number"/>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <input type="submit" value="Add my Appointment"/>
        </td>
    </tr>
</table>
</form:form>
</body>
</html>

```

### jsp/addAppointment.jsp:

```

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<html>
<head>
    <title>Patient Appointment Registration Status</title>
</head>
<body>
    <h1>${mainHeader}</h1>
    <h2>Patient Appointment Registration Status</h2>
    <p>${message}</p>

    <table>
        <tr>
            <td>Patient Name:</td>
            <td>${patient.patientName}</td>
        </tr>
        <tr>
            <td>Patient Contact:</td>
            <td>${patient.patientContact}</td>
        </tr>
        <tr>
            <td>Patient Gender:</td>
            <td>${patient.patientGender}</td>
        </tr>
        <tr>
            <td>Patient Date of Birth:</td>
            <td>${patient.patientDoB}</td>
        </tr>
    </table>

```

```

        <td>Patient Allergies:</td>
        <td>
            <c:forEach var="allergy" items="${patient.patientAllergies}">
                ${allergy}<br/>
            </c:forEach>
        </td>
    </tr>
    <tr>
        <td colspan="2">Patient's Address Details:</td>
    </tr>
    <tr>
        <td>Street:</td>
        <td>${patient.patientAddress.street}</td>
    </tr>
    <tr>
        <td>City:</td>
        <td>${patient.patientAddress.city}</td>
    </tr>
    <tr>
        <td>Street:</td>
        <td>${patient.patientAddress.state}</td>
    </tr>
    <tr>
        <td>Country:</td>
        <td>${patient.patientAddress.country}</td>
    </tr>
    <tr>
        <td>Pincode:</td>
        <td>${patient.patientAddress.pincode}</td>
    </tr>
</table>
</body>
</html>

```

## Output:

← → ↻ ⓘ localhost:8080/form/appointment

### Welcome to the best Clinic

#### patient Appointment Registration

Name:

Contact No:

Gender: ☐ Male ☒ Female

Date of Birth:

Allergies: ☐ Peanuts ☒ Dust ☒ Smoke

Address Details:

Street:

City:

State:

Country:  ▼

Pincode:

← → ↻ ⓘ localhost:8080/form/addappointment

### Welcome to the best Clinic

#### Patient Appointment Registration Status

We have registered your details

Patient Name: Hetvi

Patient Contact: 8758033238

Patient Gender: Female

Patient Date of Birth: Fri Jan 12 00:00:00 IST 2024

Patient Allergies: Dust  
Smoke

Patient's Address Details:

Street: B/56 Ganesh Park-2

City: Ankleshwar

State: Gujarat

Country: India

Pincode: 393002

# Experiment: 10

**AIM:** Implement a SpringMVC based application to perform form validation and its use in Thymeleaf template file.

## config/MySpringMvcAppInitializer.java:

```
package com.example.config;

import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class MySpringMvcAppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {

        //Load Spring Web application configuration
        AnnotationConfigWebApplicationContext context = new
AnnotationConfigWebApplicationContext();
        //register beans config class
        context.register(WebApplicationContextConfig.class);

        //Create and register the DispatcherServlet
        DispatcherServlet servlet = new DispatcherServlet(context);
        ServletRegistration.Dynamic registration =
servletContext.addServlet("dispatcher", servlet);
        registration.setLoadOnStartup(1);
        registration.addMapping("/");
    }
}
```

## config/WebApplicationContextConfig.java:

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
```

```

//substitutue of root beans element
@Configuration
//substitutue of <mvc:annotation-driver>
@EnableWebMvc
//substitute of <context:component-scan>
@ComponentScan("com.example")
public class WebApplicationContextConfig implements WebMvcConfigurer {

    //substitute of viewResolver bean in .xml file
    @Bean
    public ViewResolver getViewResolver(){
        InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/jsp/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}

```

## Patient.java:

```

package com.example;

import jakarta.validation.Valid;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Past;
import jakarta.validation.constraints.Pattern;
import org.springframework.format.annotation.DateTimeFormat;

import java.util.Date;
import java.util.List;

public class Patient {
    @Pattern(regexp = "^[a-zA-Z.]+$",
        message = "Patient name should only contain alphabets and dots.")
    private String patientName;
    @Pattern(regexp = "[6-9]\\d{9}$",
        message = "Please enter a valid 10-digit mobile number starting with
6.")
    private String patientContact;
    @Pattern(regexp = "^(Male|Female)$",
        message = "Please select a valid gender (Male or Female).")
    private String patientGender;
    @NotNull(message = "Birthdate cannot be empty")
    @Past(message = "Please enter a valid past")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date patientDoB;
    private List<String> patientAllergies;
}

```

```

@Valid
private PatientAddress patientAddress;

public String getPatientName() {
    return patientName;
}

public void setPatientName(String patientName) {
    this.patientName = patientName;
}

public String getPatientContact() {
    return patientContact;
}

public void setPatientContact(String patientContact) {
    this.patientContact = patientContact;
}

public String getPatientGender() {
    return patientGender;
}

public void setPatientGender(String patientGender) {
    this.patientGender = patientGender;
}

public Date getPatientDoB() {
    return patientDoB;
}

public void setPatientDoB(Date patientDoB) {
    this.patientDoB = patientDoB;
}

public List<String> getPatientAllergies() {
    return patientAllergies;
}

public void setPatientAllergies(List<String> patientAllergies) {
    this.patientAllergies = patientAllergies;
}

public PatientAddress getPatientAddress() {
    return patientAddress;
}

public void setPatientAddress(PatientAddress patientAddress) {

```

```
        this.patientAddress = patientAddress;
    }
}
```

### PatientAddress.java:

```
package com.example;

import jakarta.validation.constraints.*;

public class PatientAddress {
    @Size(min = 3, max = 40,
        message = "Street must contain minimum 3 characters and maximum 40 characters")
    private String street;
    @NotNull(message = "City cannot be empty")
    @NotEmpty(message = "City cannot be empty")
    private String city;
    @NotBlank(message = "State cannot be blank")
    private String state;
    @Pattern(regexp = "^(India|USA|Canada|UK)$",
        message = "Please select a valid Country from the list.")
    private String country;
    @Min(value = 1000,message = "Pincode must be greater then or equal to 1000.")
    @Max(value = 999999,message = "Pincode must be less than or equal to 999999.")
    private int pincode;

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }
}
```

```

    public void setState(String state) {
        this.state = state;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public int getPincode() {
        return pincode;
    }

    public void setPincode(int pincode) {
        this.pincode = pincode;
    }
}

```

### PatientController.java:

```

package com.example;

import jakarta.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

import java.util.Arrays;
import java.util.Map;

@Controller
@SessionAttributes({"patient"})
public class PatientController {
    @RequestMapping("/appointment")
    public ModelAndView appointment(Model model){
        Patient patient = new Patient();
        patient.setPatientGender("Male");
        model.addAttribute("countryList",
Arrays.asList("India", "USA", "Canada", "UK"));
        model.addAttribute("allergyList",Arrays.asList("Peanuts", "Dust", "Smoke
"));

        return new ModelAndView("appointment","patient",patient);
    }
}

```



```

    @PostMapping("/addappointment")
    public ModelAndView addAppointment(@Valid @ModelAttribute Patient patient,
    BindingResult bindingResult, Model model)
    {
        if(bindingResult.hasErrors()){
            model.addAttribute("countryList",
Arrays.asList("India", "USA", "Canada", "UK"));
            model.addAttribute("allergyList", Arrays.asList("Peanuts", "Dust", "S
moke"));

            return new ModelAndView("appointment", "patient", patient);
        }
        ModelAndView modelAndView = new ModelAndView("addappointment");
        modelAndView.addObject("message",
            "We have registered your details"
        );
        return modelAndView;
    }

    @ModelAttribute
    public void addingCommonObjects(Model model){
        model.addAttribute("mainHeader", "Welcome to the best Clinic");
    }
}

```

### jsp/appointment.jsp:

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <title>Patient Appointment Registration</title>
</head>
<body>
<h1>${mainHeader}</h1>
<h2>patient Appointment Registration</h2>
<form:form method="post" action="addappointment" modelAttribute="patient">
    <table>
        <tr>
            <td>Name:</td>
            <td>
                <form:errors path="patientName" style="color:red"/><br/>
                <form:input type="text" name="patientName"
path="patientName"/>
            </td>
        </tr>
        <tr>
            <td>Contact No:</td>

```

```

        <td>
            <form:errors path="patientContact" style="color:red"/><br/>
            <form:input type="text" name="patientContact"
path="patientContact"/>
        </td>
    </tr>
    <tr>
        <td>Gender:</td>
        <td>
            <form:errors path="patientGender" style="color:red"/><br/>
            <form:radiobutton path="patientGender" value="Male"
label="Male"/>
            <form:radiobutton path="patientGender" value="Female"
label="Female"/>
        </td>
    </tr>
    <tr>
        <td>Date of Birth:</td>
        <td>
            <form:errors path="patientDoB" style="color:red"/><br/>
            <form:input type="date" path="patientDoB"/>
        </td>
    </tr>
    <tr>
        <td>Allergies:</td>
        <td>
            <form:checkboxes path="patientAllergies"
items="${allergyList}"/>
        </td>
    </tr>
    <tr>
        <td colspan="2">Address Details:</td>
    </tr>
    <tr>
        <td>Street:</td>
        <td>
            <form:errors path="patientAddress.street"
style="color:red"/><br/>
            <form:input path="patientAddress.street" type="text"/>
        </td>
    </tr>
    <tr>
        <td>City:</td>
        <td>
            <form:errors path="patientAddress.city"
style="color:red"/><br/>
            <form:input path="patientAddress.city" type="text"/>
        </td>
    </tr>

```

```

        </tr>
        <tr>
            <td>State:</td>
            <td>
                <form:errors path="patientAddress.state"
style="color:red"/><br/>
                <form:input path="patientAddress.state" type="text"/>
            </td>
        </tr>
        <tr>
            <td>Country:</td>
            <td>
                <form:errors path="patientAddress.country"
style="color:red"/><br/>
                <form:select path="patientAddress.country">
                    <form:options items="${countryList}"/>
                </form:select>
            </td>
        </tr>
        <tr>
            <td>Pincode:</td>
            <td>
                <form:errors path="patientAddress.pincode"
style="color:red"/><br/>
                <form:input path="patientAddress.pincode" type="number"/>
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="Add my Appointment"/>
            </td>
        </tr>
    </table>
</form:form>
</body>
</html>

```

## jsp/addAppointment.jsp:

```

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<html>
<head>
    <title>Patient Appointment Registration Status</title>
</head>
<body>

```

```

<h1>${mainHeader}</h1>
<h2>Patient Appointment Registration Status</h2>
<p>${message}</p>

<table>
  <tr>
    <td>Patient Name:</td>
    <td>${patient.patientName}</td>
  </tr>
  <tr>
    <td>Patient Contact:</td>
    <td>${patient.patientContact}</td>
  </tr>
  <tr>
    <td>Patient Gender:</td>
    <td>${patient.patientGender}</td>
  </tr>
  <tr>
    <td>Patient Date of Birth:</td>
    <td>${patient.patientDoB}</td>
  </tr>
  <tr>
    <td>Patient Allergies:</td>
    <td>
      <c:forEach var="allergy" items="${patient.patientAllergies}">
        ${allergy}<br/>
      </c:forEach>
    </td>
  </tr>
  <tr>
    <td colspan="2">Patient's Address Details:</td>
  </tr>
  <tr>
    <td>Street:</td>
    <td>${patient.patientAddress.street}</td>
  </tr>
  <tr>
    <td>City:</td>
    <td>${patient.patientAddress.city}</td>
  </tr>
  <tr>
    <td>Street:</td>
    <td>${patient.patientAddress.state}</td>
  </tr>
  <tr>
    <td>Country:</td>
    <td>${patient.patientAddress.country}</td>
  </tr>

```

```

        <tr>
            <td>Pincode:</td>
            <td>${patient.patientAddress.pincode}</td>
        </tr>
    </table>

    <a href="dashboard">click to dashboard</a>
</body>
</html>

```

## Output:

The screenshot shows a web browser at the URL `localhost:8080/form/addappointment`. The page has a title "Welcome to the best Clinic" and a subtitle "patient Appointment Registration". The form contains the following fields and messages:

- Name:** A text input field with a red error message: "Patient name should only contain alphabets and dots."
- Contact No:** A text input field with a red error message: "Please enter a valid 10-digit mobile number starting with 6."
- Gender:** Radio buttons for "Male" (selected) and "Female".
- Date of Birth:** A date picker with a red error message: "Birthdate cannot be empty". The input shows "dd-mm-yyyy".
- Allergies:** Checkboxes for "Peanuts", "Dust" (checked), and "Smoke".
- Address Details:**
  - Street:** A text input field with a red error message: "Street must contain minimum 3 characters and maximum 40 characters".
  - City:** A text input field with a red error message: "City cannot be empty".
  - State:** A text input field with a red error message: "State cannot be blank".
  - Country:** A dropdown menu showing "India".
  - Pincode:** A text input field with a red error message: "Pincode must be greater then or equal to 1000.". The input shows "0".

At the bottom of the form is a button labeled "Add my Appointment".

# Welcome to the best Clinic

## Patient Appointment Registration Status

We have registered your details

Patient Name: Hetvi  
Patient Contact: 8758033238  
Patient Gender: Female  
Patient Date of Birth: Fri Jan 12 00:00:00 IST 2024  
Patient Allergies: Dust  
Smoke  
Patient's Address Details:  
Street: B/56 Ganesh Park-2  
City: Ankleshwar  
Street: Gujarat  
Country: India  
Pincode: 393002