File upload and field level validation and global exception handling:  
=============================

Add these dependencies in pom.xml  
<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

</dependencies>

Create Enity class and DTO class both are same   
=====================================

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.mongodb.core.mapping.Document;

**import** javax.validation.constraints.Email;

**import** javax.validation.constraints.Max;

**import** javax.validation.constraints.Min;

**import** javax.validation.constraints.NotBlank;

**import** javax.validation.constraints.NotNull;

**import** javax.validation.constraints.Pattern;

**import** javax.validation.constraints.Size;

**import** lombok.Data;

@Data

@Document(collection = "students")

**public** **class** Student {

@Id

**private** String id;

**private** String name;

**private** String email;

**private** String phone;

**private** Integer age;

**private** **byte**[] resume;

**private** **byte**[] photo;

}

Create repository class:  
===============================

**import** org.springframework.data.mongodb.repository.MongoRepository;

**public** **interface** StudentRepository **extends** MongoRepository<Student, String> {

}

**Create service class:  
===============**

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** org.springframework.web.multipart.MultipartFile;

**import** java.io.IOException;

@Service

**public** **class** StudentService {

@Autowired

**private** StudentRepository studentRepository;

**public** **void** saveStudent(StudentDTO studentDTO) **throws** IOException {

Student student = **new** Student();

student.setName(studentDTO.getName());

student.setEmail(studentDTO.getEmail());

MultipartFile resumeFile = studentDTO.getResume();

MultipartFile photoFile = studentDTO.getPhoto();

**if** (resumeFile != **null** && !resumeFile.isEmpty()) {

student.setResume(resumeFile.getBytes());

}

**if** (photoFile != **null** && !photoFile.isEmpty()) {

student.setPhoto(photoFile.getBytes());

}

studentRepository.save(student);

}

}

**Create controller class:  
====================**

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.validation.annotation.Validated;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestPart;

**import** org.springframework.web.bind.annotation.RestController;

**import** org.springframework.web.servlet.mvc.support.RedirectAttributes;

@RestController

@RequestMapping("/student")

**public** **class** StudentController {

@Autowired

**private** StudentService studentService;

@PostMapping("/upload")

**public** ResponseEntity<String> uploadStudent(@Validated @ModelAttribute StudentDTO studentDTO, RedirectAttributes redirectAttributes) {

**try** {

studentService.saveStudent(studentDTO);

**return** ResponseEntity.*ok*("Files uploaded successfully!");

} **catch** (Exception e) {

**return** ResponseEntity.*status*(500).body("Failed to upload files: " + e.getMessage());

}

}

}

**Field level validations:  
===================**  
  
we need this dependency in pom.xml   
  
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

Field-level validation ensures that each field in your DTO (Data Transfer Object) or entity adheres to specific rules and constraints. This is essential to guarantee data integrity and prevent invalid data from being processed or stored. In Spring Boot, field-level validation is typically performed using annotations from the javax.validation package.

**Common Validation Annotations**

Here are some commonly used validation annotations:

1. **@NotNull**: Ensures the field is not null.
2. **@NotBlank**: Ensures the field is not null and the trimmed length is greater than zero (usually used with strings).
3. **@NotEmpty**: Ensures the field is not null and the collection size, array length, or string length is greater than zero.
4. **@Size**: Specifies the size constraints for a collection, array, or string.
5. **@Min** and **@Max**: Specifies the minimum and maximum value for a numeric field.
6. **@Email**: Validates that the field is a well-formed email address.
7. **@Pattern**: Validates that the field matches a regular expression pattern.
8. **@Positive** and **@Negative**: Ensures the field's value is positive or negative, respectively.
9. **@Future** and **@Past**: Ensures the date is in the future or past, respectively.

Example:  
  
DTO:  
========  
**import** javax.validation.constraints.Email;

**import** javax.validation.constraints.Max;

**import** javax.validation.constraints.Min;

**import** javax.validation.constraints.NotBlank;

**import** javax.validation.constraints.NotNull;

**import** javax.validation.constraints.Pattern;

**import** javax.validation.constraints.Size;

**import** org.springframework.web.multipart.MultipartFile;

**import** lombok.Data;

@Data

**public** **class** StudentDTO {

@NotBlank(message = "Name is mandatory")

@Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")

**private** String name;

@Email(message = "Email should be valid")

@NotBlank(message = "Email is mandatory")

**private** String email;

@Pattern(regexp = "^\\+?[0-9]{10,15}$", message = "Phone number is invalid")

**private** String phone;

@NotNull(message = "Age is mandatory")

@Min(value = 0, message = "Age must be at least 0")

@Max(value = 100, message = "Age must be less than or equal to 100")

**private** Integer age;

@NotNull(message = "Resume is mandatory")

**private** MultipartFile resume;

@NotNull(message = "Photo is mandatory")

**private** MultipartFile photo;

}

Entity:  
=========

**package** com.k7it.model;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.mongodb.core.mapping.Document;

**import** jakarta.validation.constraints.Email;

**import** jakarta.validation.constraints.Max;

**import** jakarta.validation.constraints.Min;

**import** jakarta.validation.constraints.NotBlank;

**import** jakarta.validation.constraints.NotNull;

**import** jakarta.validation.constraints.Pattern;

**import** jakarta.validation.constraints.Size;

**import** lombok.Data;

@Data

@Document(collection = "students")

**public** **class** Student {

@Id

**private** String id;

**private** String name;

**private** String email;

**private** String phone;

**private** Integer age;

**private** **byte**[] resume;

**private** **byte**[] photo;

}

Controller class:

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.validation.annotation.Validated;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestPart;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/student")

**public** **class** StudentController {

@Autowired

**private** StudentService studentService;

@PostMapping("/upload")

**public** ResponseEntity<String> uploadStudent(@Validated @ModelAttribute StudentDTO studentDTO, RedirectAttributes redirectAttributes) {

**try** {

studentService.saveStudent(studentDTO);

**return** ResponseEntity.*ok*("Files uploaded successfully!");

} **catch** (Exception e) {

**return** ResponseEntity.*status*(500).body("Failed to upload files: " + e.getMessage());

}

}

Service:  
=========

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** org.springframework.web.multipart.MultipartFile;

**import** java.io.IOException;

@Service

**public** **class** StudentService {

@Autowired

**private** StudentRepository studentRepository;

**public** **void** saveStudent(StudentDTO studentDTO) **throws** IOException {

Student student = **new** Student();

student.setName(studentDTO.getName());

student.setEmail(studentDTO.getEmail());

student.setPhone(studentDTO.getPhone());

student.setAge(studentDTO.getAge());

MultipartFile resumeFile = studentDTO.getResume();

MultipartFile photoFile = studentDTO.getPhoto();

**if** (resumeFile != **null** && !resumeFile.isEmpty()) {

student.setResume(resumeFile.getBytes());

}

**if** (photoFile != **null** && !photoFile.isEmpty()) {

student.setPhoto(photoFile.getBytes());

}

studentRepository.save(student);

}

}

Global Exception handler :  
==================

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.MethodArgumentNotValidException;

**import** org.springframework.web.bind.annotation.ExceptionHandler;

**import** org.springframework.web.bind.annotation.RestControllerAdvice;

**import** java.util.HashMap;

**import** java.util.Map;

@RestControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(MethodArgumentNotValidException.**class**)

**public** ResponseEntity<Map<String, String>> handleValidationExceptions(MethodArgumentNotValidException ex) {

Map<String, String> errors = **new** HashMap<>();

ex.getBindingResult().getFieldErrors().forEach(error ->

errors.put(error.getField(), error.getDefaultMessage())

);

**return** **new** ResponseEntity<>(errors, HttpStatus.***BAD\_REQUEST***);

}

@ExceptionHandler(NullPointerException.**class**)

**public** ResponseEntity< String> handleNPExceptions(NullPoinerException ex) {

**return** **new** ResponseEntity<>(ex.getMessage(), HttpStatus.***BAD\_REQUEST***);

}

}

Note:

**@ModelAttribute**

* **Purpose**: Binds a method parameter or method return value to a named model attribute and exposes it to a web view.
* **Usage**: Commonly used for form data binding, where you need to bind HTTP request parameters (from application/x-www-form-urlencoded or multipart/form-data) to a Java object.
* **Functionality**: Automatically binds all form fields to the corresponding fields in the Java object.

Mini Project: for Student crud operations   
==========================================

Step1: Create HomeController.java which is using to manage Home.html and getting all CRUD actions html pages view files , that’s the reason we should use this HomeController as Spring Web MVC controller instead of RestfullController   
  
HomeController.java  
=====================

**package** com.k7it.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.GetMapping;

@Controller

**public** **class** HomeController {

@GetMapping("/")

**public** String home() {

**return** "home";

}

@GetMapping("/create-student")

**public** String createStudentPage() {

**return** "create\_student";

}

@GetMapping("/students")

**public** String listStudentsPage() {

**return** "list\_students";

}

@GetMapping("/edit-student")

**public** String editStudentPage() {

**return** "edit\_student";

}

@GetMapping("/view-student")

**public** String viewStudentPage() {

**return** "view\_student";

}

}

why HomeController use @Controller instead of @RestController

The @Controller and @RestController annotations in Spring serve different purposes, and their use depends on what you want to achieve with your controller.

### @Controller

* **Purpose**: The @Controller annotation is used to indicate that a class serves as a controller in the Spring MVC framework. It is typically used to handle web page navigation.
* **Usage**: Primarily used to return view templates (like HTML, JSP, etc.). It is often used in conjunction with view technologies like Thymeleaf, FreeMarker, or JSP.
* **Functionality**: Methods in a @Controller can return a String representing the name of a view template, and Spring will render this view.

### @RestController

* **Purpose**: The @RestController annotation is a specialized version of @Controller. It is used to create RESTful web services.
* **Usage**: Primarily used to return JSON or XML data directly to the client. It combines @Controller and @ResponseBody annotations.
* **Functionality**: Methods in a @RestController return data (usually JSON or XML) directly, rather than rendering a view.

### Example of Usage

**Using** @Controller **for Navigation:**

java

@Controller

public class HomeController {

@GetMapping("/")

public String home() {

return "home"; // Returns the "home.html" template

}

@GetMapping("/create-student")

public String createStudentPage() {

return "create\_student"; // Returns the "create\_student.html" template

}

}

**Using** @RestController **for RESTful Services:**

java

@RestController

@RequestMapping("/api")

public class StudentRestController {

@GetMapping("/students")

public List<Student> getAllStudents() {

return studentService.getStudents(); // Returns JSON data

}

@GetMapping("/students/{id}")

public Student getStudentById(@PathVariable String id) {

return studentService.getStudentById(id); // Returns JSON data

}

}

### Why Use @Controller for HomeController:

* **Navigation and View Rendering**: The HomeController in your project is responsible for navigating to different pages like the home page, create student page, list students page, etc. These pages are rendered as HTML views, not as JSON or XML data. Therefore, @Controller is the appropriate choice.
* **View Technology**: When you return a String from a method annotated with @Controller, Spring uses the view resolver to render the corresponding HTML page.

### Summary:

* **Use** @Controller when you need to return HTML views for navigation and rendering web pages.
* **Use** @RestController when you need to return JSON or XML data directly for RESTful web services.

Then create All HTML files: under resource/template folder   
==============================================  
your-project

│

├── src

│ ├── main

│ │ ├── java

│ │ │ └── com

│ │ │ └── k7it

│ │ │ └── controller

│ │ │ └── HomeController.java

│ │ ├── resources

│ │ ├── template

│ │ │ ├── home.html

│ │ │ ├── create\_student.html

│ │ │ ├── list\_students.html

│ │ │ ├── edit\_student.html

│ │ │ └── view\_student.html

│ │ └── application.properties

│ └── ...  
  
🡪When working with Spring Boot or Spring MVC ,the template folder is used to store HTML for Template Rendering, Resource Mapping, Maintainability and organization…

🡪The template folder is for dynamic content that may be processed by the server like Html..  
🡪The static folder is for static content like CSS, JavaScript….

home.html  
===============  
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home</title>

</head>

<body>

<h1>Home Page</h1>

<button onclick="location.href='/create-student'">Create Student</button>

<button onclick="location.href='/students'">List Students</button>

</body>

</html>

create\_student.html  
===================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Create Student</title>

<style>

.error {

color: red;

}

</style>

</head>

<body>

<h1>Create Student</h1>

<form id="createForm" action="/student/upload" method="post" enctype="multipart/form-data">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required minlength="2" maxlength="50"><br><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br><br>

<label for="phone">Phone:</label>

<input type="tel" id="phone" name="phone" pattern="^\+?[0-9]{10,15}$" required><br><br>

<label for="age">Age:</label>

<input type="number" id="age" name="age" required min="0" max="100"><br><br>

<label for="resume">Resume:</label>

<input type="file" id="resume" name="resume" required><br><br>

<label for="photo">Photo:</label>

<input type="file" id="photo" name="photo" required><br><br>

<button type="submit">Create Student</button>

</form>

<script>

document.getElementById('createForm').addEventListener('submit', **function**(event) {

// Front-end validation

**var** form = event.target;

**var** name = form.name.value;

**var** email = form.email.value;

**var** phone = form.phone.value;

**var** age = form.age.value;

**var** resume = form.resume.value;

**var** photo = form.photo.value;

**var** errorMessages = [];

// Validate name

if (!name || name.length < 2 || name.length > 50) {

errorMessages.push("Name must be between 2 and 50 characters.");

}

// Validate email

if (!email || !/^\S+@\S+\.\S+$/.test(email)) {

errorMessages.push("Please enter a valid email address.");

}

// Validate phone

if (phone && !/^\+?[0-9]{10,15}$/.test(phone)) {

errorMessages.push("Phone number is invalid.");

}

// Validate age

if (!age || age < 0 || age > 100) {

errorMessages.push("Age must be between 0 and 100.");

}

// Validate resume

if (!resume) {

errorMessages.push("Resume is mandatory.");

}

// Validate photo

if (!photo) {

errorMessages.push("Photo is mandatory.");

}

if (errorMessages.length > 0) {

event.preventDefault();

alert(errorMessages.join("\n"));

}

});

</script>

</body>

</html>

list\_students.html  
================================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>List of Students</title>

<style>

table {

width: 100%;

border-collapse: collapse;

}

table, th, td {

border: 1px solid black;

}

th, td {

padding: 10px;

text-align: left;

}

th {

background-color: #f2f2f2;

}

.action-icons {

cursor: pointer;

}

</style>

</head>

<body>

<h1>List of Students</h1>

<table>

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Email</th>

<th>Phone</th>

<th>Age</th>

<th>Actions</th>

</tr>

</thead>

<tbody id="studentTableBody">

<!-- Student rows will be dynamically inserted here -->

</tbody>

</table>

<script>

// Sample data to display (this would come from a backend service in a real application)

**const** students = [

{id: 1, name: "John Doe", email: "john.doe@example.com", phone: "+1234567890", age: 20},

{id: 2, name: "Jane Smith", email: "jane.smith@example.com", phone: "+1987654321", age: 22},

];

// Function to render student rows

**function** renderStudentRows() {

**const** tableBody = document.getElementById('studentTableBody');

tableBody.innerHTML = ''; // Clear existing rows

students.forEach(student **=>** {

**const** row = document.createElement('tr');

row.innerHTML = `

<td>${student.id}</td>

<td>${student.name}</td>

<td>${student.email}</td>

<td>${student.phone}</td>

<td>${student.age}</td>

<td>

<span class="action-icons" onclick="viewStudent(${student.id})">👁️</span>

<span class="action-icons" onclick="editStudent(${student.id})">✏️</span>

<span class="action-icons" onclick="deleteStudent(${student.id})">🗑️</span>

</td>

`;

tableBody.appendChild(row);

});

}

// Placeholder functions for actions

**function** viewStudent(id) {

location.href = '/view-student?id=' + id;

}

**function** editStudent(id) {

location.href = '/edit-student?id=' + id;

}

**function** deleteStudent(id) {

if (confirm("Are you sure you want to delete this student?")) {

alert('Student with ID ' + id + ' will be deleted (implement delete request in real app).');

}

}

// Initial render

renderStudentRows();

</script>

</body>

</html>

edit\_student.html  
==========================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Edit Student</title>

<style>

.error {

color: red;

}

</style>

</head>

<body>

<h1>Edit Student</h1>

<form id="editForm" action="/student/update" method="post" enctype="multipart/form-data">

<input type="hidden" id="id" name="id">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required minlength="2" maxlength="50"><br><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br><br>

<label for="phone">Phone:</label>

<input type="tel" id="phone" name="phone" pattern="^\+?[0-9]{10,15}$" required><br><br>

<label for="age">Age:</label>

<input type="number" id="age" name="age" required min="0" max="100"><br><br>

<label for="resume">Resume:</label>

<input type="file" id="resume" name="resume"><br><br>

<label for="photo">Photo:</label>

<input type="file" id="photo" name="photo"><br><br>

<button type="submit">Update Student</button>

</form>

<script>

// Get student ID from query string

**const** urlParams = new URLSearchParams(window.location.search);

**const** studentId = urlParams.get('id');

// Fetch student details (simulated with sample data)

**const** student = {

id: studentId,

name: "John Doe",

email: "john.doe@example.com",

phone: "+1234567890",

age: 20

};

// Populate form with student details

document.getElementById('id').value = student.id;

document.getElementById('name').value = student.name;

document.getElementById('email').value = student.email;

document.getElementById('phone').value = student.phone;

document.getElementById('age').value = student.age;

document.getElementById('editForm').addEventListener('submit', **function**(event) {

// Front-end validation

**var** form = event.target;

**var** name = form.name.value;

**var** email = form.email.value;

**var** phone = form.phone.value;

**var** age = form.age.value;

**var** errorMessages = [];

// Validate name

if (!name || name.length < 2 || name.length > 50) {

errorMessages.push("Name must be between 2 and 50 characters.");

}

// Validate email

if (!email || !/^\S+@\S+\.\S+$/.test(email)) {

errorMessages.push("Please enter a valid email address.");

}

// Validate phone

if (phone && !/^\+?[0-9]{10,15}$/.test(phone)) {

errorMessages.push("Phone number is invalid.");

}

// Validate age

if (!age || age < 0 || age > 100) {

errorMessages.push("Age must be between 0 and 100.");

}

if (errorMessages.length > 0) {

event.preventDefault();

alert(errorMessages.join("\n"));

}

});

</script>

</body>

</html>

view\_student.html  
================================================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>View Student</title>

<style>

.student-details {

font-family: Arial, sans-serif;

margin: 20px;

}

.student-details h2 {

border-bottom: 1px solid #ccc;

padding-bottom: 10px;

}

.student-details p {

margin: 10px 0;

}

</style>

</head>

<body>

<div class="student-details">

<h2>Student Details</h2>

<p><strong>ID:</strong> <span id="studentId"></span></p>

<p><strong>Name:</strong> <span id="studentName"></span></p>

<p><strong>Email:</strong> <span id="studentEmail"></span></p>

<p><strong>Phone:</strong> <span id="studentPhone"></span></p>

<p><strong>Age:</strong> <span id="studentAge"></span></p>

<p><strong>Resume:</strong> <a id="studentResume" href="#" download>Download Resume</a></p>

<p><strong>Photo:</strong> <br> <img id="studentPhoto" src="#" alt="Student Photo" style="max-width: 200px;"></p>

<button onclick="location.href='/students'">Back to List</button>

</div>

<script>

// Get student ID from query string

**const** urlParams = new URLSearchParams(window.location.search);

**const** studentId = urlParams.get('id');

// Simulated student data (this would be fetched from a backend service in a real application)

**const** student = {

id: studentId,

name: "John Doe",

email: "john.doe@example.com",

phone: "+1234567890",

age: 20,

resume: "resume.pdf", // Simulated file name for download

photo: "photo.jpg" // Simulated image source

};

// Populate student details

document.getElementById('studentId').textContent = student.id;

document.getElementById('studentName').textContent = student.name;

document.getElementById('studentEmail').textContent = student.email;

document.getElementById('studentPhone').textContent = student.phone;

document.getElementById('studentAge').textContent = student.age;

document.getElementById('studentResume').setAttribute('href', student.resume);

document.getElementById('studentPhoto').setAttribute('src', student.photo);

</script>

</body>

</html>

StudentController.java  
================================================

**package** com.k7it.controller;

**import** java.io.IOException;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.validation.annotation.Validated;

**import** org.springframework.web.bind.annotation.DeleteMapping;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.ModelAttribute;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.PutMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** org.springframework.web.servlet.mvc.support.RedirectAttributes;

**import** com.k7it.dto.StudentDTO;

**import** com.k7it.model.Student;

**import** com.k7it.service.StudentService;

@RestController

@RequestMapping("/student")

**public** **class** StudentController {

@Autowired

**private** StudentService studentService;

@PostMapping("/upload")

**public** ResponseEntity<String> uploadStudent(@Validated @ModelAttribute StudentDTO studentDTO, RedirectAttributes redirectAttributes) {

**try** {

studentService.saveStudent(studentDTO);

**return** ResponseEntity.*ok*("Files uploaded successfully!");

} **catch** (Exception e) {

**return** ResponseEntity.*status*(500).body("Failed to upload files: " + e.getMessage());

}

}

@GetMapping("/all")

**public** List<Student> getAllStudents() {

**return** studentService.getStudents();

}

@GetMapping("/{id}")

**public** Student getStudentById(@PathVariable String id) {

**return** studentService.getStudentById(id);

}

@PutMapping("/{id}")

**public** ResponseEntity<Student> updateStudent(@PathVariable String id, @Validated @ModelAttribute StudentDTO studentDTO) **throws** IOException {

**return** studentService.updateStudent(id, studentDTO);

}

@DeleteMapping("/{id}")

**public** ResponseEntity<Void> deleteStudent(@PathVariable String id) {

**return** studentService.deleteStudent(id);

}

}

StudentService.java  
=============================================================

**package** com.k7it.service;

**import** java.io.IOException;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.stereotype.Service;

**import** org.springframework.web.multipart.MultipartFile;

**import** com.k7it.dto.StudentDTO;

**import** com.k7it.model.Student;

**import** com.k7it.repo.StudentRepository;

@Service

**public** **class** StudentService {

@Autowired

**private** StudentRepository studentRepository;

**public** **void** saveStudent(StudentDTO studentDTO) **throws** IOException {

Student student = **new** Student();

student.setName(studentDTO.getName());

student.setEmail(studentDTO.getEmail());

student.setPhone(studentDTO.getPhone());

student.setAge(studentDTO.getAge());

MultipartFile resumeFile = studentDTO.getResume();

MultipartFile photoFile = studentDTO.getPhoto();

**if** (resumeFile != **null** && !resumeFile.isEmpty()) {

student.setResume(resumeFile.getBytes());

}

**if** (photoFile != **null** && !photoFile.isEmpty()) {

student.setPhoto(photoFile.getBytes());

}

studentRepository.save(student);

}

**public** List<Student> getStudents() {

**return** studentRepository.findAll();

}

**public** Student getStudentById(String id) {

**return** studentRepository.findById(id).orElse(**null**);

}

**public** ResponseEntity<Student> updateStudent(String id, StudentDTO studentDTO) **throws** IOException {

Student student = studentRepository.findById(id).orElse(**null**);

**if** (student != **null**) {

student.setName(studentDTO.getName());

student.setEmail(studentDTO.getEmail());

student.setPhone(studentDTO.getPhone());

student.setAge(studentDTO.getAge());

MultipartFile resumeFile = studentDTO.getResume();

MultipartFile photoFile = studentDTO.getPhoto();

**if** (resumeFile != **null** && !resumeFile.isEmpty()) {

student.setResume(resumeFile.getBytes());

}

**if** (photoFile != **null** && !photoFile.isEmpty()) {

student.setPhoto(photoFile.getBytes());

}

studentRepository.save(student);

**return** ResponseEntity.*ok*(student);

} **else** {

**return** ResponseEntity.*notFound*().build();

}

}

**public** ResponseEntity<Void> deleteStudent(String id) {

studentRepository.deleteById(id);

**return** ResponseEntity.*noContent*().build();

}

}

Mini project on Employee Management System  
========================================  
EMS.Application.java  
==================  
**package** com.k7it;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** EmsApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EmsApplication.**class**, args);

}

}

EmployeeController.java  
========================

**package** com.k7it.controller;

**import** java.io.IOException;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.HttpHeaders;

**import** org.springframework.http.MediaType;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.validation.annotation.Validated;

**import** org.springframework.web.bind.annotation.DeleteMapping;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.ModelAttribute;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.PutMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** org.springframework.web.servlet.mvc.support.RedirectAttributes;

**import** com.k7it.dto.EmployeeDTO;

**import** com.k7it.model.Employee;

**import** com.k7it.service.EmployeeService;

@RestController

@RequestMapping("/api/v1")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeService employeeService;

@PostMapping("/employee")

**public** ResponseEntity<String> createEmployee(@Validated @ModelAttribute EmployeeDTO employeeDTO, RedirectAttributes redirectAttributes)

**throws** IOException {

**try** {

employeeService.saveEmployee(employeeDTO);

**return** ResponseEntity.*ok*("Employee Created successfully!");

} **catch** (IOException e) {

**return** ResponseEntity.*status*(500).body("Failed to Create Employee: " + e.getMessage());

}

}

@GetMapping("/employee/all")

**public** List<Employee> getAllEmployees() {

**return** employeeService.getEmployees();

}

@GetMapping("/employee/{id}")

**public** Employee getEmployeeById(@PathVariable String id) {

**return** employeeService.getEmployeeById(id);

}

@PutMapping("/employee/{id}")

**public** ResponseEntity<Employee> updateEmployee(@PathVariable String id, @Validated @ModelAttribute EmployeeDTO employeeDTO) **throws** IOException {

**return** employeeService.updateEmployee(id, employeeDTO);

}

@DeleteMapping("/employee/{id}")

**public** ResponseEntity<Void> deleteEmployee(@PathVariable String id) {

**return** employeeService.deleteEmployee(id);

}

@GetMapping("/employee/resume/{id}")

**public** ResponseEntity<**byte**[]> getResume(@PathVariable String id) {

Employee employee = employeeService.getEmployeeById(id);

**if** (employee.getResume() != **null**) {

HttpHeaders headers = **new** HttpHeaders();

headers.set(HttpHeaders.***CONTENT\_DISPOSITION***, "attachment; filename="+employee.getResumeFileName());

headers.setContentType(MediaType.***APPLICATION\_OCTET\_STREAM***);

**return** ResponseEntity.*ok*().headers(headers).body(employee.getResume());

} **else** {

**return** ResponseEntity.*notFound*().build();

}

}

@GetMapping("/employee/photo/{id}")

**public** ResponseEntity<**byte**[]> getPhoto(@PathVariable String id) {

Employee employee = employeeService.getEmployeeById(id);

**if** (employee.getPhoto() != **null**) {

HttpHeaders headers = **new** HttpHeaders();

headers.set(HttpHeaders.***CONTENT\_DISPOSITION***, "attachment; filename="+employee.getPhotoFileName());

headers.setContentType(MediaType.***APPLICATION\_OCTET\_STREAM***);

**return** ResponseEntity.*ok*().headers(headers).body(employee.getPhoto());

} **else** {

**return** ResponseEntity.*notFound*().build();

}

}

**private** String getFileExtension(String fileName) {

**if** (fileName == **null** || !fileName.contains(".")) {

**return** "";

}

**return** fileName.substring(fileName.lastIndexOf('.') + 1);

}

}

HomeController.java  
====================  
**package** com.k7it.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.GetMapping;

@Controller

**public** **class** HomeController {

@GetMapping("/")

**public** String home() {

**return** "home";

}

@GetMapping("/create-employee")

**public** String createEmployeePage() {

**return** "create\_employee"; // Corresponds to src/main/resources/templates/create\_employee.html

}

@GetMapping("/list-employees")

**public** String listEmployeesPage() {

**return** "list\_employees"; // Corresponds to src/main/resources/templates/list\_Employees.html

}

@GetMapping("/edit-employee")

**public** String editEmployeePage() {

**return** "edit\_employee"; // Corresponds to src/main/resources/templates/edit\_employee.html

}

@GetMapping("/view-employee")

**public** String viewEmployeePage() {

**return** "view\_employee"; // Corresponds to src/main/resources/templates/view\_employee.html

}

}  
  
EmployeeDTO.java  
=================  
**package** com.k7it.dto;

**import** org.springframework.web.multipart.MultipartFile;

**import** jakarta.validation.constraints.Email;

**import** jakarta.validation.constraints.Max;

**import** jakarta.validation.constraints.Min;

**import** jakarta.validation.constraints.NotBlank;

**import** jakarta.validation.constraints.NotNull;

**import** jakarta.validation.constraints.Pattern;

**import** jakarta.validation.constraints.Size;

**import** lombok.Data;

@Data

**public** **class** EmployeeDTO {

@NotBlank(message = "Name is mandatory field")

@Size(min = 3, max = 50, message = "Name must be min 3 chars and max 50 chars")

**private** String name;

@NotNull(message = "Emil is mandatory field")

@Email(message = "invalid emial id")

**private** String email;

@Pattern(regexp = "^\\+?[0-9]{10,15}$", message = "Phone number is invalid. It should be 10 to 15 digits long and may start with a +.")

**private** String phone;

@NotNull(message = "Age is mandatory field")

@Min(value = 0)

@Max(value = 100)

**private** Integer age;

@NotNull(message = "Resume is mandatory field")

**private** MultipartFile resume;

@NotNull(message = "Photo is mandatory field")

**private** MultipartFile photo;

}

EmployeeModel.java  
==================  
**package** com.k7it.model;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.mongodb.core.mapping.Document;

**import** jakarta.validation.constraints.NotNull;

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

**import** lombok.NoArgsConstructor;

**import** lombok.ToString;

@Document(collection = "Employees")

@Data

@AllArgsConstructor

@NoArgsConstructor

@ToString

**public** **class** Employee {

@Id

**private** String id;

**private** String name;

**private** String email;

**private** String phone;

**private** Integer age;

**private** **byte**[] resume;

**private** String resumeFileName; // Field to store the original file name of the resume

**private** **byte**[] photo;

**private** String photoFileName; // Field to store the original file name of the photo

}

EmployeeRepository.java  
========================  
**package** com.k7it.repo;

**import** org.springframework.data.mongodb.repository.MongoRepository;

**import** com.k7it.model.Employee;

**public** **interface** EmployeeRepository **extends** MongoRepository<Employee, String> {

}

EmployeeService.java  
====================  
**package** com.k7it.service;

**import** java.io.IOException;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.stereotype.Service;

**import** org.springframework.web.multipart.MultipartFile;

**import** com.k7it.dto.EmployeeDTO;

**import** com.k7it.model.Employee;

**import** com.k7it.repo.EmployeeRepository;

@Service

**public** **class** EmployeeService {

**private** **final** EmployeeRepository employeeRepository;

@Autowired

**public** EmployeeService(EmployeeRepository employeeRepository) {

**this**.employeeRepository = employeeRepository;

}

**public** **void** saveEmployee(EmployeeDTO employeeDTO) **throws** IOException {

Employee employee = **new** Employee();

employee.setAge(employeeDTO.getAge());

employee.setName(employeeDTO.getName());

employee.setEmail(employeeDTO.getEmail());

employee.setPhone(employeeDTO.getPhone());

MultipartFile resumeFile = employeeDTO.getResume();

MultipartFile photoFile = employeeDTO.getPhoto();

**if** (resumeFile != **null** && !resumeFile.isEmpty()) {

employee.setResume(resumeFile.getBytes());

employee.setResumeFileName(resumeFile.getOriginalFilename());

}

**if** (photoFile != **null** && !photoFile.isEmpty()) {

employee.setPhoto(photoFile.getBytes());

employee.setPhotoFileName(photoFile.getOriginalFilename());

}

employeeRepository.save(employee);

}

**public** List<Employee> getEmployees() {

**return** employeeRepository.findAll();

}

**public** Employee getEmployeeById(String id) {

**return** employeeRepository.findById(id).orElse(**null**);

}

**public** ResponseEntity<Employee> updateEmployee(String id, EmployeeDTO employeeDTO) **throws** IOException {

Employee employee = employeeRepository.findById(id).orElse(**null**);

**if** (employee != **null**) {

employee.setName(employeeDTO.getName());

employee.setEmail(employeeDTO.getEmail());

employee.setPhone(employeeDTO.getPhone());

employee.setAge(employeeDTO.getAge());

MultipartFile resumeFile = employeeDTO.getResume();

MultipartFile photoFile = employeeDTO.getPhoto();

**if** (resumeFile != **null** && !resumeFile.isEmpty()) {

employee.setResume(resumeFile.getBytes());

employee.setResumeFileName(resumeFile.getOriginalFilename());

}

**if** (photoFile != **null** && !photoFile.isEmpty()) {

employee.setPhoto(photoFile.getBytes());

employee.setPhotoFileName(photoFile.getOriginalFilename());

}

employeeRepository.save(employee);

**return** ResponseEntity.*ok*(employee);

} **else** {

**return** ResponseEntity.*notFound*().build();

}

}

**public** ResponseEntity<Void> deleteEmployee(String id) {

employeeRepository.deleteById(id);

**return** ResponseEntity.*noContent*().build();

}

}

create\_employee.html  
=====================  
<html>

<div class="navigation">

<button onclick=location.href="/">Home</button>

<button onclick=location.href="/list-employees">List Of Employees</button>

</div>

<form id = "createForm" method="post" enctype="multipart/form-data">

Name:<input type="text" id="name" name="name"/><br>

Age:<input type="number" id="age" name="age"/><br>

Emial:<input type="email" id="email" name="email"/><br>

Phone:<input type="tel" id="phone" name="phone"/><br>

Resume:<input type="file" id="resume" name="resume"/><br>

Photo:<input type="file" id="photo" name="photo"/><br>

<input type="submit" value="SaveEmployee"/>

</form>

<script>

document.getElementById('createForm').addEventListener('submit', **function**(event) {

event.preventDefault(); // Prevent the default form submission

**var** form = event.target;

**var** formData = new FormData(form);

// Perform AJAX request to upload the employee data

fetch('/api/v1/employee', {

method: 'POST',

body: formData

})

.then(response **=>** {

if (response.ok) {

alert('Employee created successfully.');

window.location.href = '/list-employees'; // Redirect to the list page

} else {

return response.text().then(text **=>** { throw new Error(text) });

}

})

.catch(error **=>** {

alert('Failed to create Employee: ' + error.message);

});

});

</script>

</html>

edit\_employee.html  
==================  
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Edit Employee</title>

<style>

.error {

color: red;

}

.navigation {

margin-top: 20px;

}

.navigation a {

margin-right: 10px;

}

</style>

</head>

<body>

<h1>Edit Employee</h1>

<div class="navigation">

<button onclick="location.href='/list-employees'">Back to Employee List</button>

<button onclick="location.href='/'">Home</button>

</div>

<br>

<form id="editForm" method="post" enctype="multipart/form-data">

<input type="hidden" id="id" name="id">

Name:<input type="text" id="name" name="name"/><br>

Age:<input type="number" id="age" name="age"/><br>

Emial:<input type="email" id="email" name="email"/><br>

Phone:<input type="tel" id="phone" name="phone"/><br>

Resume:<input type="file" id="resume" name="resume"/><br>

Photo:<input type="file" id="photo" name="photo"/><br>

<input type="submit" value="UpdateEmployee"/>

</form>

<script>

// Get employee ID from query string

**const** urlParams = new URLSearchParams(window.location.search);

**const** employeeId = urlParams.get('id');

// Fetch student details from the backend

**function** fetchEmployeeDetails(id) {

fetch('/api/v1/employee/' + id)

.then(response **=>** response.json())

.then(employee **=>** {

document.getElementById('id').value = employee.id;

document.getElementById('name').value = employee.name;

document.getElementById('email').value = employee.email;

document.getElementById('phone').value = employee.phone;

document.getElementById('age').value = employee.age;

})

.catch(error **=>** {

console.error('Error fetching student details:', error);

});

}

// Handle form submission with PUT request

document.getElementById('editForm').addEventListener('submit', **function** (event) {

event.preventDefault();

**const** form = event.target;

**const** formData = new FormData(form);

fetch(`/api/v1/employee/${employeeId}`, {

method: 'PUT',

body: formData

})

.then(response **=>** {

if (response.ok) {

alert('Employee updated successfully.');

window.location.href = '/list-employees';

} else {

alert('Failed to update employee.');

}

})

.catch(error **=>** {

console.error('Error updating employee:', error);

});

});

// Fetch and populate student details on page load

fetchEmployeeDetails(employeeId);

</script>

</body>

</html>

Home.html  
==========

<html>

<head>

<h1>Employee Home Page</h1>

</head>

</button>

<button onclick=location.href="/create-employee">CreateEmployee</a></button>

<button onclick=location.href="/list-employees">ListOfEmployees</button>

</div>

</html>

list\_emoployees.html  
=====================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>List of Employees</title>

<style>

table {

width: 100%;

border-collapse: collapse;

}

table, th, td {

border: 1px solid black;

}

th, td {

padding: 10px;

text-align: left;

}

th {

background-color: #f2f2f2;

}

.action-icons {

cursor: pointer;

}

.navigation {

margin-bottom: 20px;

}

</style>

</head>

<body>

<div class="navigation">

<a href="/">Home</a>

</div>

<h1>List of Employees</h1>

<table>

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Age</th>

<th>Email</th>

<th>Phone</th>

<th>Actions</th>

</tr>

</thead>

<tbody id="employeeTableBody">

<!-- Employee rows will be dynamically inserted here -->

</tbody>

</table>

<script>

// Fetch student data from the backend

**function** fetchEmployeeData() {

fetch('/api/v1/employee/all')

.then(response **=>** response.json())

.then(data **=>** {

renderEmployeeRows(data);

})

.catch(error **=>** {

console.error('Error fetching employee data:', error);

});

}

// Function to render student rows

**function** renderEmployeeRows(employees) {

**const** tableBody = document.getElementById('employeeTableBody');

tableBody.innerHTML = ''; // Clear existing rows

employees.forEach(employee **=>** {

**const** row = document.createElement('tr');

row.innerHTML = `

<td>${employee.id}</td>

<td>${employee.name}</td>

<td>${employee.age}</td>

<td>${employee.email}</td>

<td>${employee.phone}</td>

<td>

<span class="action-icons" onclick="viewEmployee('${employee.id}')">👁️</span>

<span class="action-icons" onclick="editEmployee('${employee.id}')">✏️</span>

<span class="action-icons" onclick="deleteEmployee('${employee.id}')">🗑️</span>

</td>

`;

tableBody.appendChild(row);

});

}

// Placeholder functions for actions

**function** viewEmployee(id) {

location.href = '/view-employee?id=' + id;

}

// edit employee

**function** editEmployee(id) {

location.href = '/edit-employee?id=' + id;

}

// Function to delete student data

**function** deleteEmployee(id) {

if (confirm("Are you sure you want to delete this employee?")) {

fetch(`/api/v1/employee/${id}`, {

method: 'DELETE'

})

.then(response **=>** {

if (response.ok) {

alert(`Employee with ID ${id} deleted successfully.`);

fetchEmployeeData(); // Refresh the employee list

} else {

alert(`Failed to delete employee with ID ${id}.`);

}

})

.catch(error **=>** {

console.error('Error deleting student:', error);

});

}

}

// Initial fetch and render

fetchEmployeeData();

</script>

</body>

</html>

View\_employee.html  
===================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>View Employee</title>

<style>

.employee-details {

font-family: Arial, sans-serif;

margin: 20px;

}

.employee-details h2 {

border-bottom: 1px solid #ccc;

padding-bottom: 10px;

}

.employee-details p {

margin: 10px 0;

}

</style>

</head>

<body>

<div class="employee-details">

<h2>Employee Details</h2>

<p><strong>ID:</strong> <span id="employeeId"></span></p>

<p><strong>Name:</strong> <span id="employeeName"></span></p>

<p><strong>Age:</strong> <span id="employeeAge"></span></p>

<p><strong>Email:</strong> <span id="employeeEmail"></span></p>

<p><strong>Phone:</strong> <span id="employeePhone"></span></p>

<p><strong>Resume:</strong> <a id="employeeResume" href="#">Download Resume</a></p>

<p><strong>Photo:</strong> <br> <br>

<a id="employeePhotoDownload" href="#"><img id="employeePhoto" src="#" alt="Employee Photo" style="max-width: 200px;"></a> </p>

<button onclick="location.href='/list-employees'">Back to List</button>

<button onclick="location.href='/'">Home</button>

</div>

<script>

// Get employee ID from query string

**const** urlParams = new URLSearchParams(window.location.search);

**const** employeeId = urlParams.get('id');

// Fetch employee details from the backend

**function** fetchEmployeeDetails(id) {

fetch(`/api/v1/employee/${id}`)

.then(response **=>** response.json())

.then(data **=>** {

document.getElementById('employeeId').textContent = data.id;

document.getElementById('employeeName').textContent = data.name;

document.getElementById('employeeEmail').textContent = data.email;

document.getElementById('employeePhone').textContent = data.phone;

document.getElementById('employeeAge').textContent = data.age;

document.getElementById('employeeResume').setAttribute('href', `/api/v1/employee/resume/${data.id}`);

document.getElementById('employeePhoto').setAttribute('src', 'data:image/jpeg;base64,' + data.photo);

document.getElementById('employeePhotoDownload').setAttribute('href', `/api/v1/employee/photo/${data.id}`);

})

.catch(error **=>** {

console.error('Error fetching employee details:', error);

});

}

// Initial fetch and render

fetchEmployeeDetails(employeeId);

</script>

</body>

</html>