Problem Statement

You should have learnt about software engineering topics ranging from version control to backend development and using Spring Boot, JPA, and PostgreSQL to build a backend application.

In this assignment, you'll apply everything that you have learnt to build an image hoster similar to Imgur, one of the top 100 most visited websites in the world.

Goals of This Assignment

Through working on the image uploader, you'll get a chance to experience what it is like to join an engineering team as a new, junior software engineer. Specifically, you'll get a chance to familiarize yourself with the codebase by fixing a few bugs in the codebase. You'll also get a chance to implement a few well-defined features to the image uploader and improve its functionalities.

Lastly, this project will also introduce you to new concepts to Spring MVC, JPA, PostgreSQL and unit testing. Some of these concepts will be explained to you, and some concepts, you need to Google and Stack Overflow to learn those concepts on your own. This experience will further enhance the experience of joining a new team as a junior software developer as mentioned above.

Stub file

As all of you have implemented the 'Image Hoster' project given in MVC architecture and Database & ORMs assessments, you will continue working on the same project as a part of this assignment.
Let us discuss in brief, the features implemented in the 'Image Hoster' project till now.

- The application runs on localhost and the user is redirected to the landing page of the application displaying all the images in the application.
- A new user can register in the application by entering the details such as username, password, full name, email address, and mobile number, and after the successful registration, he is redirected to the login page.
- The user can log in the application by entering the username and password, and after he is successfully logged in the application, he is redirected to the user homepage, displaying all the images in the application.
- After a user successfully logs in the application, he can click on the title of any image and the image details will be displayed including the description and tags of the image, along with the option to edit and delete the image.

- A user can also upload the image by entering the details such as image title, description, image file, and tags related to the image.
- You may register in the application and upload the images in the application, to check all the features discussed above.

Stub zipped file here -

https://drive.google.com/drive/u/0/folders/1tw4ij5QGGx1VIZ2OJMdBDDVGjMT73bNs

Instructions:

1. You must manually create a database named **'imageHoster'** in PostgreSQL with the user as **'postgres'** using pgAdmin as the UI.
2. Change the username and password in the 'src/main/java/imageHoster/config/JpaConfig.java' file and 'src/main/resources/META-INF/pertence.xml' file according to your PostgreSQL username and password.
3. Note that it is mandatory to assign at least one tag/category to an image while you upload the image. If you do not assign any tag to the image, the image will get uploaded in the application. But when you try to edit the image, the application throws an error.

You need to download the above-given stub file and run the code on IntelliJ. Run the application on localhost and observe all the working features of the application as mentioned above. Also, to improve the presentation of web pages, some HTML files have been updated using the external CSS file. However, you are only expected to work on the backend development and the details for any changes required in the presentation logic are clearly mentioned in the problem statement and also you can find the relevant comments in the stub file. In this assignment, you need to fix the bugs in the application and add some more features in the application. The details of the bugs to be fixed and the new features to be added in the application are given below.

Additional Notes

Spring, Unit Testing, and Mocking

Although you have learnt about unit testing in this course, you are yet to learn about unit testing in the context of a Spring Boot project.

Therefore, in the Image Uploader project, we have written the unit tests of the controllers for you. We have also added comments to those unit tests to make the tests legible, so you can read the tests and understand what those tests are trying to accomplish.

If you would like to learn more about unit testing and mocking with Spring Boot, please refer to the following resources:

- Spring MVC Test Framework
- Spring: Testing the Web Layer
- Spring Boot - Unit Testing and Mocking with Mockito and JUnit

Uncomment the unit tests written for the controller classes and confirm that all the test cases are successfully passed. Lastly, please feel to experiment with the unit tests, and write additional unit tests on your own.

HTTP Session

In the Image Uploader project, you'll see that we are storing a User object (representing the currently logged in user) in a HttpSession object. This allows us to check the details of the currently logged in user.

Please refer to the following resources to read more about HTTP Sessions, and how to use them in Spring:

- What are sessions? How do they work?
- Using Http Session With Spring Based Web Applications

Version Control Best Practices

Please follow the following best practice as you are using Git and Github to conduct version control of your code. This will make you a more effective software engineer.

- Commit often
- Make small, incremental commits

- Write good commit messages
- Make sure your code works before committing it
- Use branches -- Keep your code for bug fixes and feature development in different branches.

Here are additional readings on best Git practices:

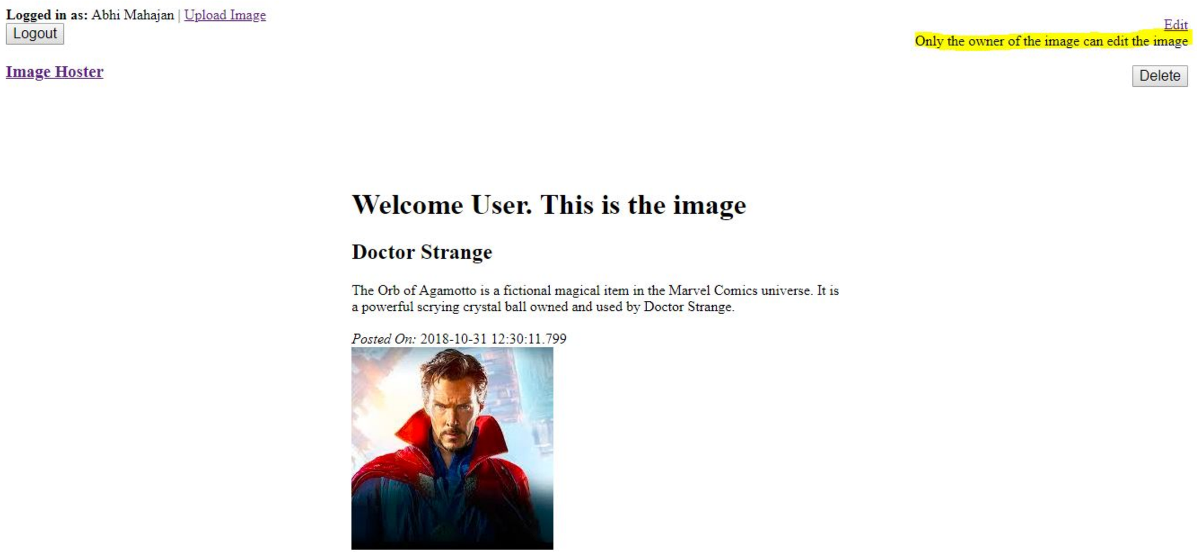
- Git Common Practices
- Commit Often, Fix it Later, Publish Once: Git Best Practices

To-Do Tasks:

Part A: Fixing Issues

1. Image upload issues:
    - If you upload an image with the same exact title as a previously uploaded image, it will get uploaded. But then, if you try to navigate to one of the images with the same title, the image uploader will display an error.
    - Please fix this issue, so that it should not show an error and take you to respective image's page.
    - Please see this issue on Github.

2. After logging into the application, it is possible to edit/delete the image which is posted by some other user. This is a bug in the application.
   Now, fix this bug in the application, such that only the owner of the image can edit/delete the image.
    - If the owner of the image is trying to edit the image, you must return the 'images/edit.html' file, thus enabling the user to edit the image. But if the non-owner of the image is trying to edit the image, return the 'images/image.html' file displaying all the details of the image and print the message **'Only the owner of the image can edit the image'**. Carefully add all the required attributes in the Model type object needed by 'images/image.html' file.
    - Observe the message when non-owner of the image is trying to edit the image as shown below:


Logged in as: Abhi Mahajan | Upload Image

Edit

Only the owner of the image can edit the image

Logout

Image Hoster

Delete

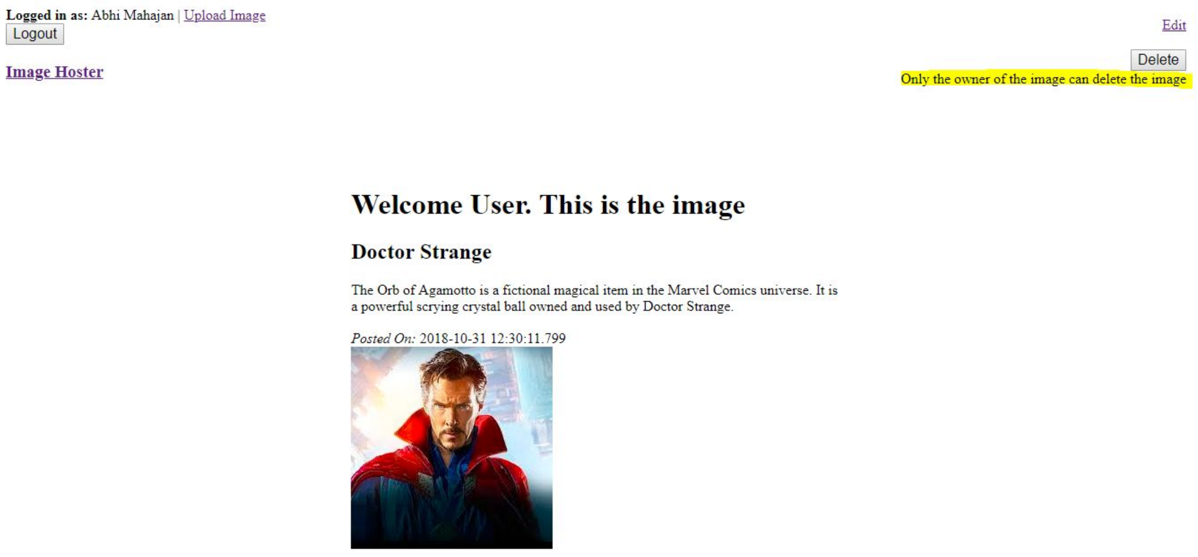
Welcome User. This is the image

Doctor Strange

The Orb of Agamotto is a fictional magical item in the Marvel Comics universe. It is a powerful scrying crystal ball owned and used by Doctor Strange.

Posted On: 2018-10-31 12:30:11.799

- If the owner of the image is trying to delete the image, the user must be redirected to the web page displaying all the images. But if the non-owner of the image is trying to delete the image, return the ‘images/image.html’ file displaying all the details of the image and print the message **‘Only the owner of the image can delete the image’**. Carefully add all the required attributes in the Model type object needed by ‘images/image.html’ file.
- Observe the message when non-owner of the image is trying to edit the image as shown below:


Logged in as: Abhi Mahajan | Upload Image

Edit

Logout

Delete

Image Hoster

Only the owner of the image can delete the image

Welcome User. This is the image

Doctor Strange

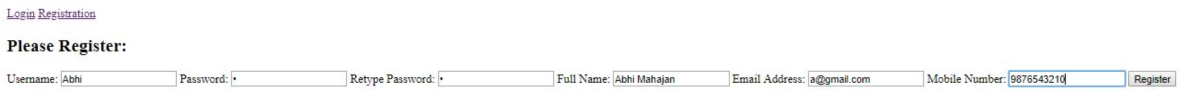
The Orb of Agamotto is a fictional magical item in the Marvel Comics universe. It is a powerful scrying crystal ball owned and used by Doctor Strange.

Posted On: 2018-10-31 12:30:11.799

Part B: Implement a new feature
After fixing the above issues, please implement the following features into the image uploader:

1. **Password Strength**
   Up till now, there is no check on the strength of the password entered by the user at the time of registration. Let us try to keep a check on the strength of the password entered by the user at the time of registration. You need to implement a feature wherein the password entered by the user must contain at least 1 alphabet (a-z and A-Z), 1 number (0-9) and 1 special character (any character other than a-z, A-Z and 0-9). The user must only be registered if the password contains 1 alphabet, 1 number, and 1 special character. And after successful registration, you must directly return 'users/login.html' file. Otherwise, the user must not be registered and again return the 'users/registration.html' file. You also need to display a message **'Password must contain at least 1 alphabet, 1 number & 1 special character'.** Carefully add all the required attributes in the Model type object needed by 'users/registration.html' file.

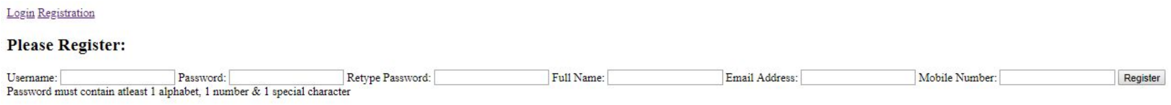
   Observe the registration web page and the password does not contain any number or special character as shown below:


Login Registration

Please Register:

Username: Abhi Password: • Retype Password: • Full Name: Abhi Mahajan Email Address: a@gmail.com Mobile Number: 9876543210 Register

   Observe the message when you try to submit the above details as shown below:


Login Registration

Please Register:

Username: Password: Retype Password: Full Name: Email Address: Mobile Number: Register
Password must contain atleast 1 alphabet, 1 number & 1 special character

   Hint:
   How to display the error message?
   - Declare and initialize a string with the message in the Controller logic as shown below:

```
String error = "Password must contain atleast 1 alphabet, 1 number & 1
special character"
```

- Add the above string specifying the password type error in the Model type object with **‘passwordTypeError’** as the key.

- Display a message in ‘users/registration.html’ file with an if condition. If the Model type object contains the string representing the password type error, only then you should display the message. Else, the message should not be displayed. Uncomment the code given below for displaying the message with if condition

```
<div  th:if="${passwordTypeError}">Password  must  contain  atleast  1
alphabet, 1 number & 1 special character</div>
```

- Note that the test cases are designed in such a way that you need to add the message **"Password must contain at least 1 alphabet, 1 number & 1 special character"** with the key as **‘passwordTypeError’** for the test cases to pass**.**

2. Comments

Now, implement a feature wherein a user can add a comment to any image in the application after he is logged in the application, and, to implement this feature, you’ll have to add the following to the image uploader:

- Create a Comment model class. The model class contains the following attributes:
    - id - Datatype should be int. It should be a primary key. Also explicitly mention the column name as ‘id’ to be created in the database.
    - text - Datatype should be a string. Note that this column can have text-based data that will be longer than 256 characters.
    - createdDate - Datatype should be LocalDate.
    - user - It should be of type User. The ‘comment’ table is mapped to ‘users’ table through this attribute. One user can post multiple comments but one comment should belong to one user. Write the suitable annotation to map the ‘comment’ table to ‘users’ table through this attribute.
    - Image - It should be of type Image. The ‘comment’ table is mapped to ‘images’ table through this attribute. One image can have multiple comments but one comment can only belong to one image. Write the suitable annotation to map the ‘comment’ table to ‘images’ table through this attribute.

- You need to uncomment the HTML code in ‘image.html’ file to display all the comments in the application when you display the details of a particular image as shown below:

```
<div class="comments mt5">
        <article  class="ba  b--black-10  mv4"  th:each="comment  :
${comments}">
               <h1  class="f4  bg-light-gray  black-80  mv0  pv2  ph3"
th:text="${comment.user.username} + ' says'">Title of
            card</h1>
        <div class="pa3 bt b--black-10">
                                            <p  class="f6  f5-ns  lh-copy  measure"
th:text="${comment.text}">
                    text
                </p>
        </div>
    </article>
</div>
```

- Also, modify the code such that before you return ‘images/image.html’ file anywhere in the code, you always need to add the comments of the image in the Model type object in the code, with the key as **‘comments’**.

- You also need to uncomment the HTML code in ‘image.html’ file to allow the user to add the comments to an image when the details of the image are displayed as shown below:

```
<form method="POST" enctype="multipart/form-data"
        th:action="'/image/'+ ${image.id} + '/' + ${image.title} +
'/comments'">
    <fieldset id="sign_up" class="ba b--transparent ph0 mh0">
        <div class="mt3">
                <label class="db fw6 lh-copy f6" for="comment">Write a
comment</label>
                    <textarea class="pa2 input-reset ba w-100" rows="5"
name="comment" id="comment"></textarea>
        </div>
    </fieldset>
    <div>
          <input class="b ph3 pv2 input-reset ba b--black bg-transparent
grow pointer f6 dib" type="submit"
                value="Submit">
    </div>
</form>
```

- Implement a Controller class method that maps to the POST request URL **‘/image/{imageId}/{imageTitle}/comments’** for creating a new comment. After persisting the comment in the database, the controller logic must redirect to the same page displaying all the details of that particular image. Redirect to ‘showImage()’ method in ‘ImageController’ class.

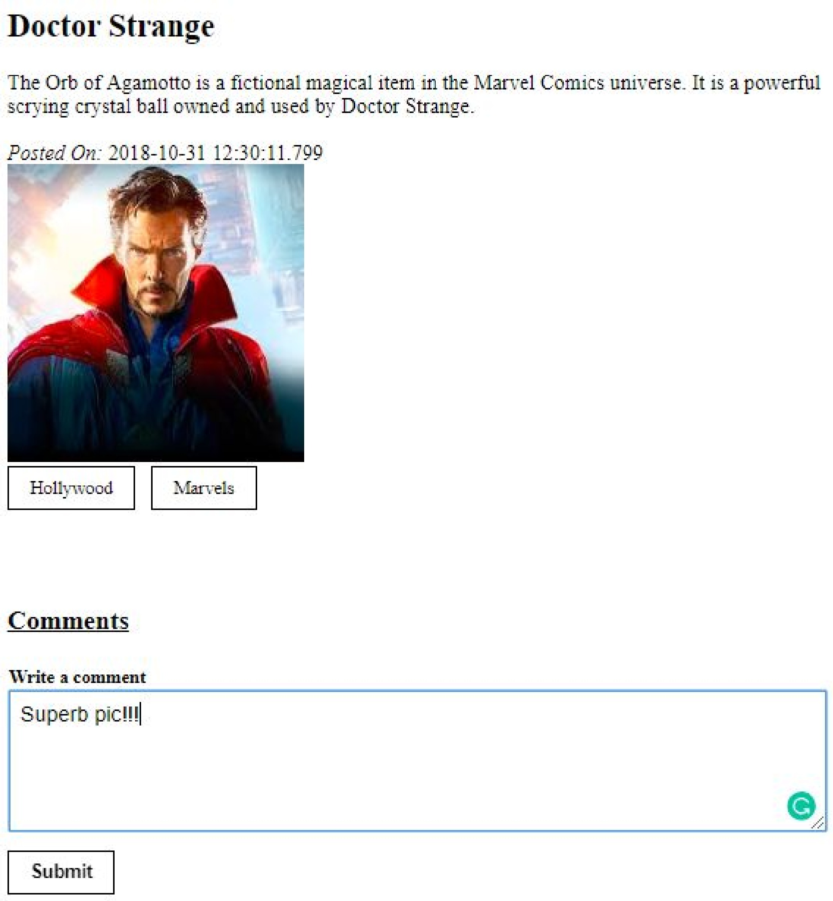
- You can obtain the request parameters from the client request using **@RequestParam** annotation. And obtain the dynamic parameter in the request URI using the annotation **@PathVariable.** You can read more about @PathParam and @RequestParam in the following links:

  @RequestParam vs @PathVariable

[How to explicitly obtain post data in Spring MVC?]

- Implement the required service logic and the Repository logic for the comment feature.

Observe the comments being given to an image by a user in the application as shown below:

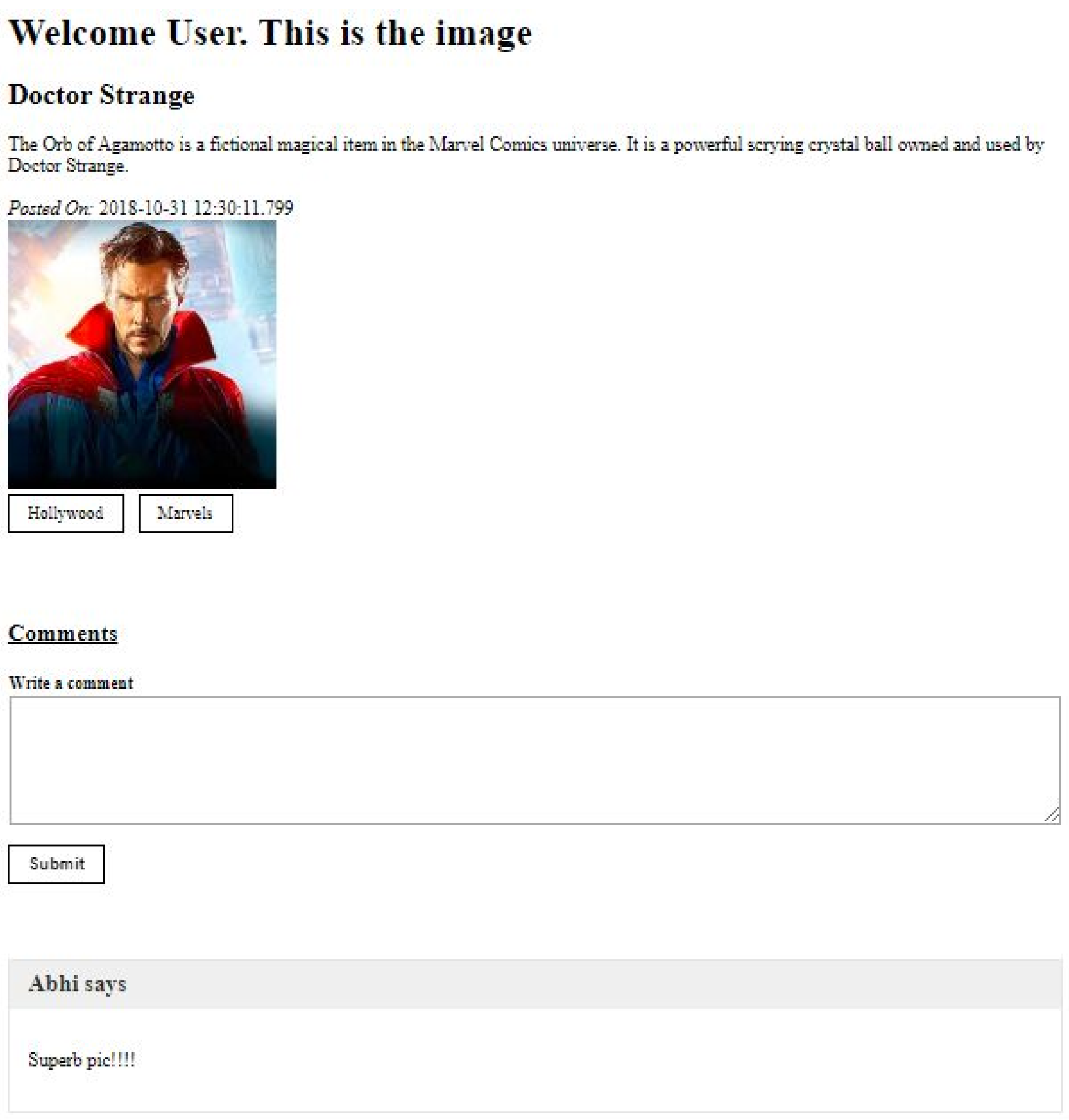


Doctor Strange

The Orb of Agamotto is a fictional magical item in the Marvel Comics universe. It is a powerful scrying crystal ball owned and used by Doctor Strange.

Posted On: 2018-10-31 12:30:11.799

Observe the above given comment being displayed as shown below:



Welcome User. This is the image

Doctor Strange

The Orb of Agamotto is a fictional magical item in the Marvel Comics universe. It is a powerful scrying crystal ball owned and used by Doctor Strange.

Posted On: 2018-10-31 12:30:11.799

Comments

Write a comment

Abhi says

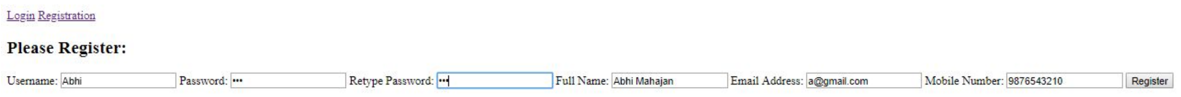
Superb pic!!!!

Part C (Optional, but Good to Have)

Retype Password

While a user registers in the application, he may do some typing mistake while entering the password. Implement a feature wherein a user will be retyping the password and will be registered only if both the passwords match. If both the passwords do not match, the user must not be registered and again return the ‘users/registration.html’ file. You also need to display a message **‘Password does not match’.** Carefully add all the required attributes in the Model type object needed by ‘users/registration.html’ file.

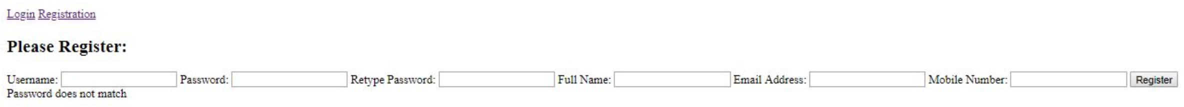
Observe the registration web page and the retype password does not match the password as shown below:


Login Registration

Please Register:

Username: Abhi Password: ••• Retype Password: ••• Full Name: Abhi Mahajan Email Address: a@gmail.com Mobile Number: 9876543210 Register

Observe the message when you try to submit the above details as shown below:


Login Registration

Please Register:

Username: Password: Retype Password: Full Name: Email Address: Mobile Number: Register
Password does not match

Hint:
How to display the error message?

- Declare and initialize a string with the message in the Controller logic as shown below:
```
String error = "Password does not match";
```

- Add the above string specifying the password error in the Model type object with **‘passwordError’** as the key.

- Display a message in ‘users/registration.html’ file with an if condition. If the Model type object contains the string representing the password error, only then you should display the message. Else, the message should not be displayed. Uncomment the code given below for displaying the message with if condition
```
<div th:if="${passwordError}">Password does not match</div>
```

- Note that the test cases are designed in such a way that you need to add the message **"Password does not match"** with the key as **‘passwordError’** for the test cases to pass.

====================
Currently, the user can edit/delete the image which has been uploaded by some other user.

Reproduction Step:
1. Get the details of the image which has been uploaded by some other user
2. Try to edit/delete the image
3. The image gets edited/deleted

The issue:
There is no check on the details of the user when the image is edited/deleted. Before editing/deleting the image, the owner of the image is not compared to the user, who is trying to edit/delete the image.

Possible Solution:
1. There must be a check on the user details who is trying to edit/delete the image. The details of the owner of the user can be compared to the details of the user who is trying to edit/delete the image.
2. If the non-owner tries to edit/delete the image, print the error message.

Hint
How to print the error message?
Let us know how to print the error message in case the non-owner of the image tries to edit the image,
1. Declare and initialize a string with the message in the Controller logic as shown below:
`String error = "Only the owner of the image can edit the image";
`
2. Add the string in the Model type object as shown below:
`model.addAttribute("editError", error);`
Note that the key is ‘editError’.

3. Since we are returning the ‘image.html’ file, you need to provide the instructions to print the message in this HTML file, but with an if condition that displays the message only when the non-owner of the image is trying to edit the image. Therefore, we will add this message in the ‘image.html’ file with an if condition.
4. Since you have added the error string in the Model type object with ‘editError’ as the key in case the non-owner of the image is trying to edit the image, you need to use this as the if condition. If the Model type object contains the ‘editError’ attribute, display the message ‘Only the owner of the image can edit the image’ as shown below:
`<div th:if="${editError}">Only the owner of the image can edit the image</div>`
Uncomment the above code in the ‘image.html’ file.
5. If the Model type object does not contain the ‘editError’ attribute, this message will not be displayed.

6. Note that the test cases are designed in such as way that you need to add the message **"Only the owner of the image can edit the image"** with the key as **‘editError’** for the test cases to pass.
7. Similarly, you need to restrict the deletion of the image by the non-owner of the image. The error message to be added in the Model type object when the non-owner of the image is trying to delete the image should be **“Only the owner of the image can delete the image”** and the key should be **'deleteError'**.