

[< Return to Classroom](#)

SuperDuperDrive

REVIEW

CODE REVIEW

HISTORY

Requires Changes

7 specifications require changes

Dear Student,

You have done a great job so far. I appreciate your efforts in making it to this level.

Most of the test cases implemented are correct and as per requirements.

However, there are some changes that are required. The project requirements are not fully implemented in the web-ui. Please test the web-ui to make sure it passes all the tests.

Please find more details in the review.

Good luck with your next submission!

Basic Functionality



There are Spring Boot annotations like `@Controller`, `@RestController`, `@RequestBody`, `@RequestParam`, etc. in the Java classes.

✓ All the above mentioned annotations are used well in place during the course of implementation. Keep it up!

- @Controller ✓
- @PostMapping ✓
- @RequestBody ✓
- @RequestParam ✓
- @GetMapping ✓

Suggestions:

- Please read more about [spring boot annotations](#) which you can make use of in your application.
- [Refer this official documentation](#) to know more about implementation of controllers in details.



There are Thymeleaf attributes in the HTML files like th:action, etc.

✓ thymeleaf attributes have been used perfectly.

- th:action ✓
- th:href ✓
- th:if ✓
- th:text ✓

[Here](#) is a nice resource to learn more about thymeleaf form handling



There are annotations like @Mapper , @Select , @Insert , @Update , and @Delete in the Java classes and/or imports from MyBatis/iBatis API.

✓ All the above annotations are used well in place for making associated operations with database. Good work pal!

- @Mapper ✓
- @Select ✓
- @Insert ✓
- @Delete ✓
- @Update ✓

You may know!

Hibernate is another framework that deals with SQL type databases. To know more about the difference between mybatis and hibernate, [here is a nice article](#) to know more about that.

Additional resources:

- [Mybatis springboot example with mysql](#)
- [Mybatis: Selects and inserts](#)



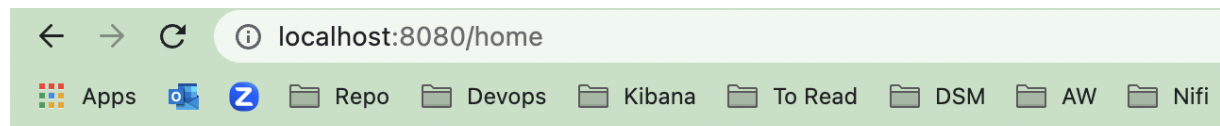
If invalid or improper inputs are given to the system, it should not crash or display raw error information. Error messages should be shown or users should be disallowed from sending invalid or improper input. Make sure your implementation passes the `testBadUrl()` and `testLargeUpload()` test cases provided by Udacity.

Nice work handling invalid inputs. However, the code breaks when a user wants to upload a very large file. In order to make the code very robust, please handle that situation as well.

Below is some config you can leverage in your application.properties file (you can set the file size limit or make it to be indefinite. But it's advisable to set a limit) alongside some exception handling in your files controller.

```
spring.servlet.multipart.max-file-size=-1
spring.servlet.multipart.max-request-size=-1
```

Also handle a situation where the user does not select any file and tries to upload.



HTTP Status 403 – Forbidden

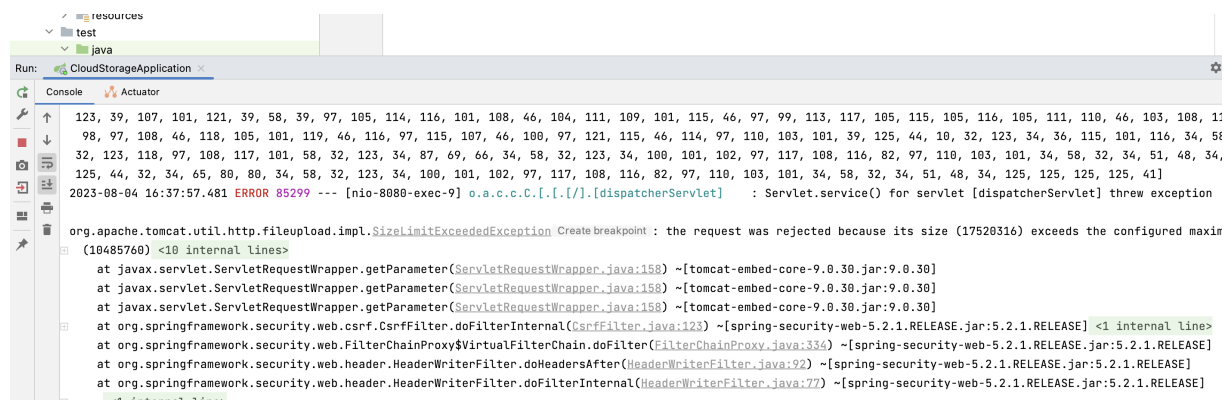
Type Status Report

Message Forbidden

Description The server understood the request but refuses to authorize it.

Apache Tomcat/9.0.30

Logs:



Front-End



The signup page already has input fields for all the data you need from the user, including username and password fields.

Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission.

✅ Thymeleaf attributes are used in place to bind the data provided to login models to authenticate the user and allow the functionality to be accessed. Great work!



On a successful signup, the user should be taken to the login page with a message indicating their registration was successful. Otherwise, an error message should be shown on the sign-up page. An error message is already present in the template, but should only be visible if an error occurred during signup. Make sure your implementation passes the `testRedirection()` test case provided by Udacity.

✅ Proper thymeleaf attributes are used in place to bind the input data to appropriate java dtos used in place. Good work!

Suggestion: However, try adding confirmPassword textbox in the signup page. It is there now-a-days in almost every website.

❌ Nice work with the user signup. However, the user needs to be re-directed to the login page upon successful signup. Think of it as the user successfully signed up, so you don't want him to click on a link before taken to the login page(that's more work for the user right?), why not just take him directly to the login page so that he can login.

In order to achieve this, you can use Redirect Attributes, something like the following:

```
@PostMapping()  
    public String signupUser(@ModelAttribute Users users, Model model, RedirectAttributes redirectAttributes) {  
        if (signupError == null) {  
            redirectAttributes.addFlashAttribute("SuccessMessage", "Sign Up Successfully");  
            return "redirect:/login";  
        }  
    }  
}
```

[Back to Login](#)

Sign Up

You successfully signed up! Please continue to the [login](#) page.

First Name

Last Name

Username

Password

Sign Up



The login page already has the username and password fields.

Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission.



Thymeleaf attributes are used in place to bind the data provided to login models to authenticate the user and allow the functionality to be accessed. Great work!



On a successful login, the user should be taken to their home page.

An error message is already present in the template, but should only be visible if an error occurred during signup.

On logout, the user should no longer have access to the home page.



Successful login takes the user to the homepage. Proper error messages are shown wherever required.



ON logout, user is no longer able to access homepage



The home page should have three tabs:

1. The user should be able to upload new files on this tab and download/remove existing files
2. The user should be able to add new notes and edit/remove existing ones
3. The user should be able to add new credentials, view existing credentials unencrypted and remove them as well

The home template already has the forms required by this functionality. Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission

Details on individual features are documented in Section 3.

- ✓ User is able to add/download/remove file
- ✓ User is able to add/remove/edit notes
- ✓ I can add new credentials, edit them and remove them. decrypted password is shown wherever required.
- ✓ All the above tabs and UI components are in place

User-Facing Features



When a user logs in, they should see the data they have added to the application.

Good work here 🙌

I logged in from user A. Created a note N1

I logged out.

I logged in again from user A.

- ✓ User is able to see the saved data after log out and again logging in

Additional Resources:

- Please [refer this article](#) to know more about spring security.



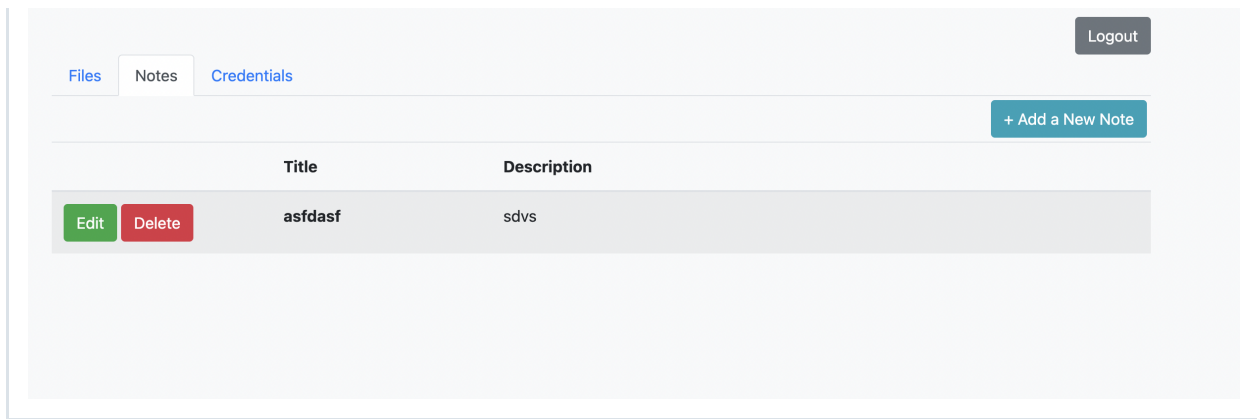
Creation: On successful note creation, the user should be shown a success message and the created note should appear in the list.

Deletion: On successful note deletion, the user should be shown a success message and the deleted note should disappear from the list.

Edit/Update: When a user selects edit, they should be shown a view with the note's current title and text. On successful note update, the user should be shown a success message and the updated note should appear from the list.

Errors: Users should be notified of errors if they occur.

- ✓ Note Creation Flow is working properly
- ✓ Note Edit Flow is working properly
- ✓ Note Removal Flow is working properly



Files Notes **Credentials** Logout

+ Add a New Note

	Title	Description
Edit Delete	asfdasf	sdvs



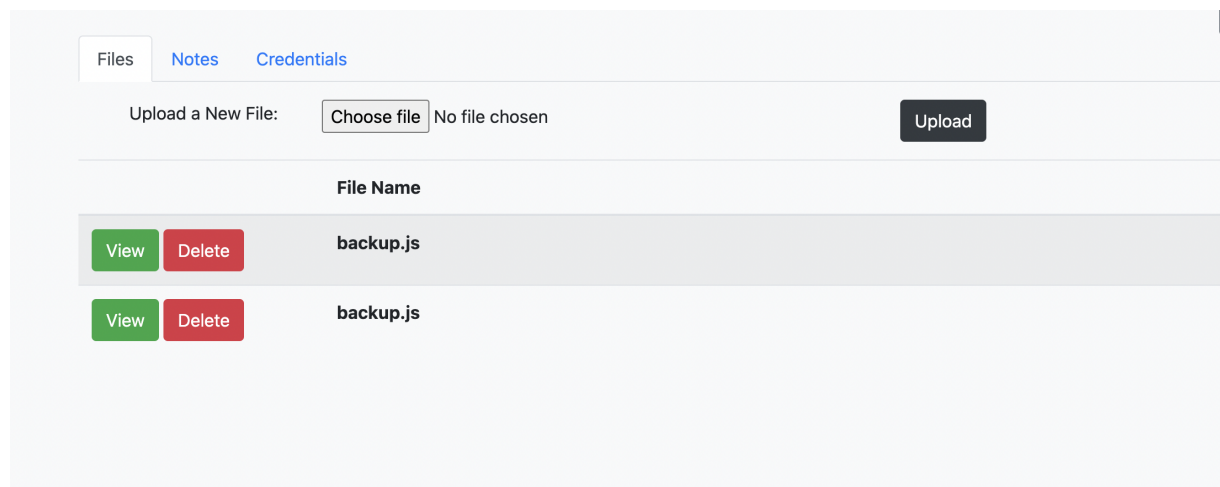
Upload: On successful file upload, the user should be shown a success message and the uploaded file should appear in the list.

Deletion: On successful file deletion, the user should be shown a success message and the deleted file should disappear from the list.

Download: On successful file download, the file should download to the user's system.

Errors: Users should be notified of errors if they occur.

- ✗ Please dont allow duplicate files to be uploaded.
- ✗ As suggested above, please handle the exceptions for large file uploads



Files Notes Credentials Logout

Upload a New File: Choose file No file chosen Upload

	File Name
View Delete	backup.js
View Delete	backup.js



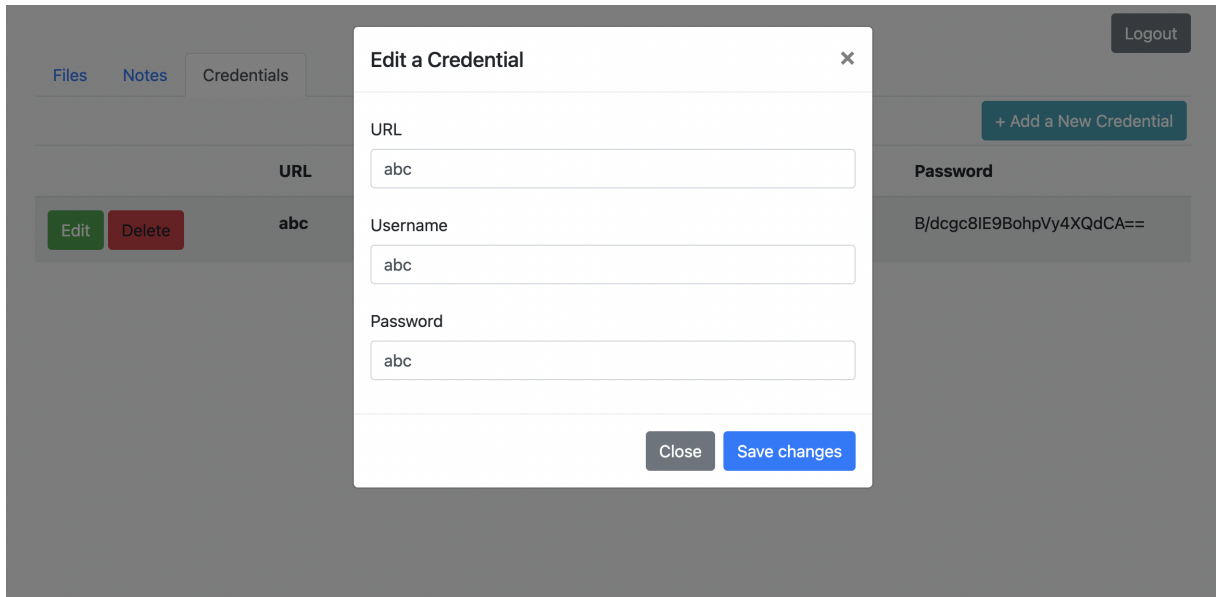
Creation: On successful credential creation, the user should be shown a success message and the created credential should appear in the list.

Edit/Update: When a user selects update, they should be shown a view with the unencrypted credentials. When they select save, the list should be updated with the edited credential details.

Deletion: On successful credential deletion, the user should be shown a success message and the deleted credential should disappear from the list.

Errors: Users should be notified of errors if they occur.

- ✓ Credentials adding Flow is working properly
- ✓ Credentials editing Flow is working properly
- ✓ Credentials removal Flow is working properly

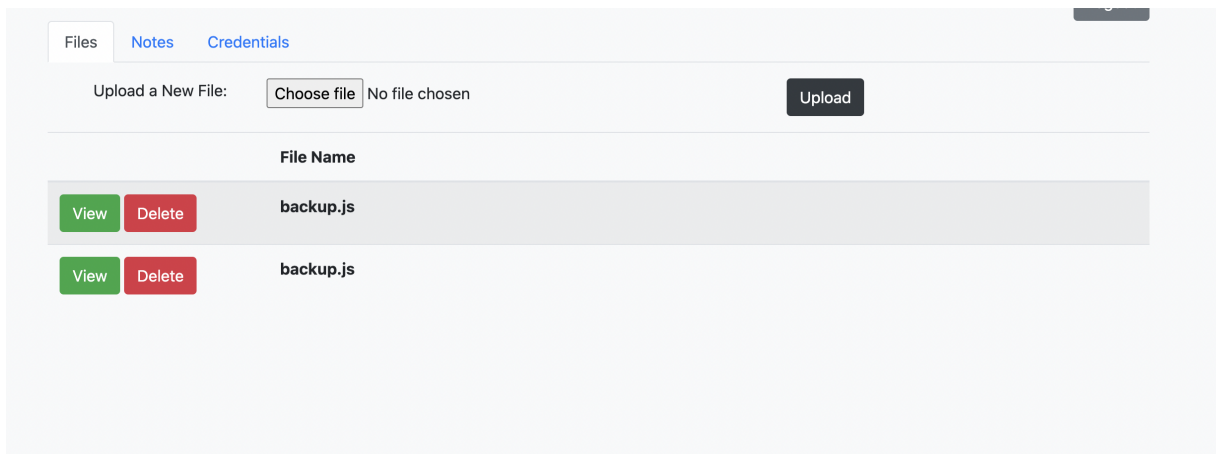


Back-End



The application should not allow duplicate usernames or duplicate filenames attributed to a single user.

✗ User is allowed to upload duplicate files



A user can't access the home page or the three tabs on that page without logging in first. The login and signup page should be visible to all the users without any authentication.

If someone isn't logged in, they must be redirected to the login page.

- ✓ Great job in not allowing the user to access home page without the login page.
- ✓ User is only allowed to access the homepage after successful login

Additional Resources:

Please [refer this article](#) to know more about Spring Security login.



A logged-in user should only be able to view their own data, and not anyone else's data. The data should only be viewable to the specific user who owns it.

Good work here 🙌

I logged in from user A. Created a note N1

I logged out.

I logged in from user B.

User B is not able to see the notes of user A.

User level data separation is implemented correctly in the application



All the passwords should be stored as encrypted in the database and shown as encrypted when the user retrieves them.

The user should only see the decrypted version when they want to edit it.

All the passwords stored are properly encrypted. Great work!

Able to see decrypted password while `Editing Password`

Same password again has a different encrypted value

Logout

Files
Notes
Credentials

+ Add a New Credential

	URL	Username	Password
Edit Delete	abc	abc	B/dcgc8IE9BohpVy4XQdCA==
Edit Delete	def	def	svvWG+YLg917x/yvtcRk8Q==



Create Java classes to model the tables in the database (specified in `src/main/resources/schema.sql`) and create `@Mapper` annotated interfaces to serve as Spring components in your application.

You should have one model class and one mapper class per database table.

Mapper interfaces and schema.sql has been used properly

Testing



Write a Selenium test that verifies that the home page is not accessible without logging in.

Write a Selenium test that signs up a new user, logs that user in, verifies that they can access the home page, then logs out and verifies that the home page is no longer accessible.


 Will be re-evaluated in next submission



Write a Selenium test that logs in an existing user, creates a note and verifies that the note details are visible in the note list.

Write a Selenium test that logs in an existing user with existing notes, clicks the edit note button on an existing note, changes the note data, saves the changes, and verifies that the changes appear in the note list.

Write a Selenium test that logs in an existing user with existing notes, clicks the delete note button on an existing note, and verifies that the note no longer appears in the note list.


 Will be re-evaluated in next submission



Write a Selenium test that logs in an existing user, creates a credential and verifies that the credential details are visible in the credential list.

Write a Selenium test that logs in an existing user with existing credentials, clicks the edit credential button on an existing credential, changes the credential data, saves the changes, and verifies that the changes appear in the credential list.

Write a Selenium test that logs in an existing user with existing credentials, clicks the delete credential button on an existing credential, and verifies that the credential no longer appears in the credential list.

 Will be re-evaluated in next submission

 RESUBMIT

 DOWNLOAD PROJECT

Learn the [best practices for revising and resubmitting your project](#).