UDACITY

Return to Classroom

SuperDuperDrive

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations 👏✨✨

Dear Learner,

You have passed the project "**SuperDuperDrive**" with flying colors.🎯
This is an amazing piece of work you have done here.

I went through all your modules and it clearly demonstrates all the functionalities.
The way you have implemented your logic and organized your code was awesome. Great job.
The corner cases are also handled appropriately such as:-

- ✅ HANDLING INVALID URL
- ✅ HANDLING LARGE FILE UPLOAD
- ✅ REDIRECTING TO THE LOGIN PAGE ON SUCCESSFUL SIGH UP

I really appreciate your hard work and dedication to this project.

Keep Learning and keep doing the good work.
Good luck !!!

Basic Functionality

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

Nice, proper Spring Boot annotations have been used where ever needed.
✅ There are Spring Boot annotations like @Controller, @RestController, @RequestBody, @RequestParams, etc. in the Java classes.

Resources

- Spring Boot annotations1
- Spring Boot annotations2

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

Yes, necessary Thymeleaf attributes could be seen in the HTMl files
✅ There are Thymeleaf attributes in the HTMl files like th:action, etc.

Resources

Handling the command object thymeleaf

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

Nice, Annotations like @Mapper, @Select, @Insert, @Update, and @Delete have been used where ever needed.
✅ There are annotations like @Mapper, @Select, @Insert, @Update, and @Delete in the Java classes and/or imports from MyBatis/iBatis API.

Resources

- mybatis sprint annotations screencast1
- mybatis sprint annotations screencast2

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.

✅ 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.

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.

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

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.

Nice, On a successful signup, the user is redirected to the login page with a message indicating their registration was successful. Otherwise, an error message is shown.
✅ Added the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission.

Resources

- link1
- link2

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.

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

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.

✅ 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.

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.

The user is able to:

- ✅ Upload new files on this tab and download/remove existing files
- ✅ Add new notes and edit/remove existing ones
- ✅ Add new credentials, view existing credentials unencrypted and remove them as well.

User-Facing Features

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

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

Resources

Sprint 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.

All the note functionalities are working as expected:-

- CREATION ✅
- DELETION ✅
- EDIT/UPDATE ✅

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.

All the file functionalities are working as expected:-

- UPLOAD ✅
- DELETE ✅
- DOWNLOAD ✅

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.

All the credential functionalities are working as expected:-

- CREATION ✅
- DELETION ✅
- EDIT/UPDATE ✅

Back-End

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

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

Resources

Registration with spring security

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.

✅ 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.

Resources

Sprint 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.

✅ A logged-in user should only be able to view their own data, and not anyone else's data.

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 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.

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.

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

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.

Successfully wrote test case that:-

- ✅ That verifies that the home page is not accessible without logging in.
- ✅ 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.

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.

Tests were written successfully to check the note functionalities i.e:

- ✅ Create a Note
- ✅ Edit the Note
- ✅ Delete a Note

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.

Tests were written successfully to check the credential functionalities i.e:

- ✅ Creating a credential
- ✅ Editing the credential
- ✅ Deleting a credential

DOWNLOAD PROJECT

RETURN TO PATH