

# Project: SuperDuperDrive

Date Started: 12/31/2020

Date Completed: TBD

## 1. Project Overview

### SuperDuperDrive Cloud Storage

You have been hired by SuperDuperDrive, which is a brand new company aiming to make a dent in the Cloud Storage market and is already facing stiff competition from rivals like Google Drive and Dropbox. That hasn't dampened their spirits at all, however. They want to include personal information management features in their application to differentiate them from the competition, and the minimum viable product includes three user-facing features:

1. **Simple File Storage:** Upload/download/remove files
2. **Note Management:** Add/update/remove text notes
3. **Password Management:** Save, edit, and delete website credentials.

SuperDuperDrive wants you to focus on building the web application with the skills you acquired in this course. That means you are responsible for developing the server, website, and tests, but other tasks like deployment belong to other teams at the company.

## 2. Project Directions

### Starter Project

A senior developer is assigned to be your tech lead and mentor, and they put together a starter project for you. It's a Maven project configured for all the dependencies the project requires, though you should feel free to add any additional dependencies you might require. [You can download or clone the starter repository here.](#)

## Project: SuperDuperDrive

Note: I forked the project over to my Github account.

### How to download a Git repository

In general, the steps for downloading (clone) a Github repository are:

**Step 1.** Download and install the [Git for your OS](#), if not already.

**Step 2.** Open terminal (macOS/Linux) or Git Bash (Windows).

**Step 3.** Run the following commands:

```
# Download the repository
git clone https://github.com/udacity/nd035-c1-spring-boot-
basics-project-starter.git

# Go inside the downloaded repository
cd nd035-c1-spring-boot-basics-project-starter

# Go to the project starter code
cd starter/cloudstorage/
```

**Once downloaded, you can import the downloaded repository (or a subfolder) into the IntelliJ IDE to have the starter code ready to work upon.**

*Remember, Github does not support downloading only a sub-folder from a Github repository. Instead, Github allows us to download the entire repository.*

## Scenario

Your tech lead already designed a database schema for the project and has added it to the `src/main/resources` directory. That means you don't have to design the database, only develop the Java code to interact with it.

Your tech lead also created some HTML templates from the design team's website mockups, and they placed them in the `src/main/resources/templates` folder. These are static pages right now, and you have to configure them with Thymeleaf to add functionality and real data from the server you develop. You may also have to change them to support testing the application.

You can download [the starter code](#) and open it as a Maven project in IntelliJ.

## Requirements and Roadmap

Your tech lead is excited to work with you and has laid out a development roadmap with requirements and milestones. They tell you that there are three layers of the application you need to implement:

1. The back-end with Spring Boot
2. The front-end with Thymeleaf
3. Application tests with Selenium

### The Back-End

The back-end is all about security and connecting the front-end to database data and actions.

#### *1. Managing User Access with Spring Security*

- You have to restrict unauthorized users from accessing pages other than the login and signup pages. To do this, you must create a security configuration class that extends the `WebSecurityConfigurerAdapter` class from Spring. Place this class in a package reserved for security and configuration. Often this package is called `security` or `config`. [CHECK]
- Spring Boot has built-in support for handling calls to the `/login` and `/logout` endpoints. You have to use the security configuration to override the default login page with one of your own, discussed in the front-end section. [CHECK]
- You also need to implement a custom `AuthenticationProvider` which authorizes user logins by matching their credentials against those stored in the database. [CHECK]

#### *2. Handling Front-End Calls with Controllers*

- You need to write controllers for the application that bind application data and functionality to the front-end. That means using Spring MVC's application model to identify the templates served for different requests and populating the view model with data needed by the template.
- The controllers you write should also be responsible for determining what, if any, error messages the application displays to the user. When a controller processes front-end requests, it should delegate the individual steps and logic of those requests to other services in the application, but it should interpret the results to ensure a smooth user experience.
- It's a good idea to keep your controllers in a single package to isolate the controller layer. Usually, we simply call this package `controller`!

- If you find yourself repeating tasks over and over again in controller methods, or your controller methods are getting long and complicated, consider abstracting some methods out into services! For example, consider the `HashService` and `EncryptionService` classes included in the starter code package `service`. These classes encapsulate simple, repetitive tasks and are available anywhere dependency injection is supported. Think about additional tasks that can be similarly abstracted and reused, and create new services to support them!

### *3. Making Calls to the Database with MyBatis Mappers*

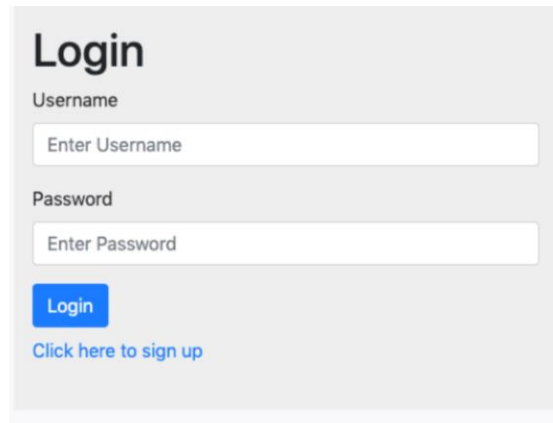
- Since you were provided with a database schema to work with, you can design Java classes to match the data in the database. These should be POJOs (Plain Old Java Objects) with fields that match the names and data types in the schema, and you should create one class per database table. These classes typically are placed in a `model` or `entity` package.
- To connect these model classes with database data, implement MyBatis mapper interfaces for each of the model types. These mappers should have methods that represent specific SQL queries and statements required by the functionality of the application. They should support the basic CRUD (Create, Read, Update, Delete) operations for their respective models at the very least. You can place these classes in (you guessed it!) the `mapper` package.

## The Front-End

Your tech lead has done a thorough job developing HTML templates for the required application pages. These templates are provided to you in the starter resources in the link above. They have included fields, modal forms, success and error message elements, as well as styling and functional components using Bootstrap as a framework. You must edit these templates and insert Thymeleaf attributes to supply the back-end data and functionality described by the following individual page requirements:

### *1. Login Page*

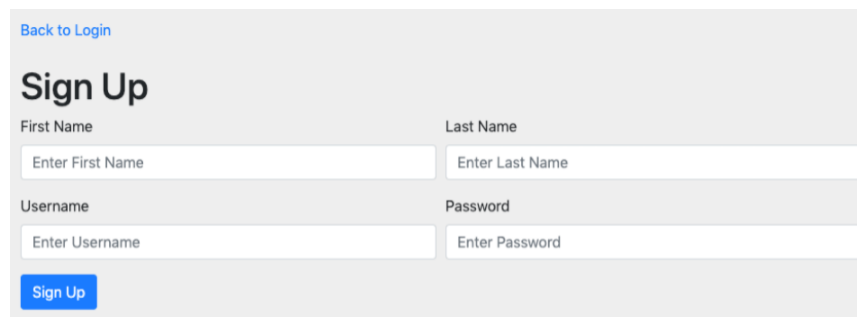
- Everyone should be allowed access to this page, and users can use this page to login to the application.
- Show login errors, like invalid username/password, on this page.



A login form with a light gray background. At the top, the word "Login" is written in a large, bold, black font. Below it, the label "Username" is followed by a white input field with the placeholder text "Enter Username". Underneath, the label "Password" is followed by a white input field with the placeholder text "Enter Password". At the bottom left, there is a blue button with the word "Login" in white. To the right of the button, there is a blue link that says "Click here to sign up".

## 2. Signup Page

- Everyone should be allowed access to this page, and potential users can use this page to sign up for a new account.
- Validate that the username supplied does not already exist in the application, and show such signup errors on the page when they arise.
- Remember to store the user's password securely!



A sign up form with a light gray background. At the top left, there is a blue link that says "Back to Login". Below it, the words "Sign Up" are written in a large, bold, black font. The form is divided into four sections: "First Name" with a white input field and placeholder "Enter First Name", "Last Name" with a white input field and placeholder "Enter Last Name", "Username" with a white input field and placeholder "Enter Username", and "Password" with a white input field and placeholder "Enter Password". At the bottom left, there is a blue button with the words "Sign Up" in white.

Sign Up page: <http://localhost:8080/signup>

## 3. Home Page

- The home page should have a logout button that allows the user to log out of the application and keep their data private.
- The home page is the center of the application and hosts the three required pieces of functionality. The existing template presents them as three tabs that can be clicked through by the user:

### i. Files

## Project: SuperDuperDrive

- The user should be able to upload files and see any files they previously uploaded.

The screenshot shows the 'Files' tab of the SuperDuperDrive application. At the top right is a 'Logout' button. Below the navigation tabs (Files, Notes, Credentials), there is an 'Upload a New File:' section with a 'Choose File' button, the text 'No file chosen', and an 'Upload' button. Below this is a table of uploaded files.

		File Name
<a href="#">View</a>	<a href="#">Delete</a>	Screen Shot 2020-02-28 at 1.16.42 PM.png
<a href="#">View</a>	<a href="#">Delete</a>	Screen Shot 2020-02-28 at 1.16.55 PM.png
<a href="#">View</a>	<a href="#">Delete</a>	Screen Shot 2020-02-28 at 1.17.26 PM.png

- The user should be able to view/download or delete previously-uploaded files.
- Any errors related to file actions should be displayed. For example, a user should not be able to upload two files with the same name, but they'll never know unless you tell them!

### ii. Notes

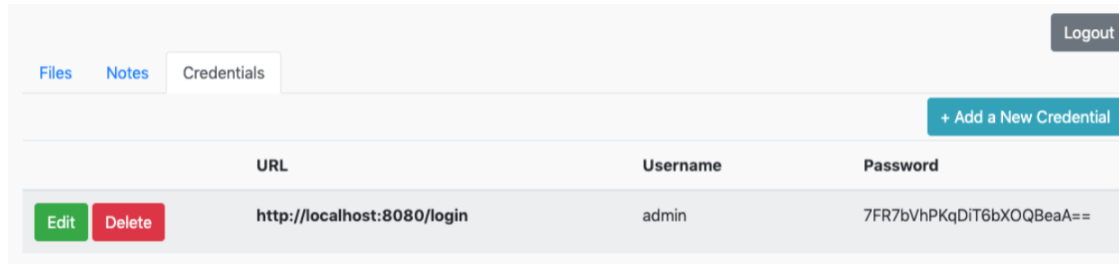
- The user should be able to create notes and see a list of the notes they have previously created.

The screenshot shows the 'Notes' tab of the SuperDuperDrive application. At the top right is a 'Logout' button. Below the navigation tabs (Files, Notes, Credentials), there is a '+ Add a New Note' button. Below this is a table of notes.

		Title	Description
<a href="#">Edit</a>	<a href="#">Delete</a>	To Do	- Design models - Create mappers - Implement services - Connect endpoints

### iii. Credentials:

- The user should be able to store credentials for specific websites and see a list of the credentials they've previously stored. If you display passwords in this list, make sure they're encrypted!



- The user should be able to view/edit or delete individual credentials. When the user views the credential, they should be able to see the unencrypted password.

## Testing

Your tech lead trusts you to do a good job, but testing is important whether you're an excel number-cruncher or a full-stack coding superstar! The QA team at SuperDuperDrive carries out extensive user testing. Still, your tech lead wants you to write some simple Selenium tests to verify user-facing functionality and prove that your code is feature-complete before the testers get their hands on it.

### 1. Write Tests for User Signup, Login, and Unauthorized Access Restrictions.

- Write a test that verifies that an unauthorized user can only access the login and signup pages.
- Write a test that signs up a new user, logs in, verifies that the home page is accessible, logs out, and verifies that the home page is no longer accessible.

### 2. Write Tests for Note Creation, Viewing, Editing, and Deletion.

- Write a test that creates a note, and verifies it is displayed.
- Write a test that edits an existing note and verifies that the changes are displayed.
- Write a test that deletes a note and verifies that the note is no longer displayed.

### 3. Write Tests for Credential Creation, Viewing, Editing, and Deletion.

- Write a test that creates a set of credentials, verifies that they are displayed, and verifies that the displayed password is encrypted.
- Write a test that views an existing set of credentials, verifies that the viewable password is unencrypted, edits the credentials, and verifies that the changes are displayed.
- Write a test that deletes an existing set of credentials and verifies that the credentials are no longer displayed.

## Final Tips and Tricks

### Password Security

Make sure not to save the plain text credentials of the application's users in the database. That's a recipe for data breach disaster! Use a hashing function to store a scrambled version instead. Your tech lead gave you a class called `HashService` that can hash passwords for you. When the user signs up, you only store a hashed version of their password in the database, and on login, you hash the password attempt before comparing it with the hashed password in the database. Your tech lead knows that can be a little confusing, so they provided this code sample to help illustrate the idea:

```
byte[] salt = new byte[16];
random.nextBytes(salt);
String encodedSalt = Base64.getEncoder().encodeToString(salt);
String hashedPassword = hashService.getHashedValue(plainPassword, encodedSalt);
return hashedPassword;
```

For storing credentials in the main part of the application, we can't hash passwords because it's a one-way operation. The user needs access to the unhashed password, after all! So instead, you should encrypt the passwords. Your tech lead provided you with a class called `EncryptionService` that can encrypt and decrypt passwords. When a user adds new credentials, encrypt the password before storing it in the database. When the user views those credentials, decrypt the password before displaying it. Here's a little code snippet on how to use `EncryptionService`:

```
SecureRandom random = new SecureRandom();
byte[] key = new byte[16];
random.nextBytes(key);
String encodedKey = Base64.getEncoder().encodeToString(key);
String encryptedPassword = encryptionService.encryptValue(password, encodedKey);
String decryptedPassword = encryptionService.decryptValue(encryptedPassword, encodedKey);
```

You aren't required to understand hashing or encryption and that's why your tech lead provided these code samples for you. If you're curious and want to learn a little more, you can do a quick Google search or follow the links below:

- [Hash Function](#)
- [Encryption](#)



Project: SuperDuperDrive

All of us here at SuperDuperDrive wish you good luck with the project!

### 3. Project: SuperDuperDrive

Double-check the rubric

Make sure you have completed all the rubric items [here](#).

Submit your project

You can submit your project as either a zip file or a Github repository link.

### Project Submission Checklist

**Before submitting your project, please review and confirm the following items.**

- ☐ I am confident all rubric items have been met and my project will pass as submitted. (If not, I will discuss with my mentor prior to submitting.)
- ☐ Project builds correctly without errors and runs.
- ☐ All required functionality exists and my project behaves as expected per the project's specifications.

**Once you have checked all these items, you are ready to submit!**

## Project: SuperDuperDrive

### Rubric

Basic Functionality	
CRITERIA	MEETS SPECIFICATIONS
Utilize Spring Boot annotations and their functions	There are Spring Boot annotations like <code>@Controller</code> , <code>@RestController</code> , <code>@RequestBody</code> , <code>@RequestParam</code> , etc. in the Java classes.
Utilize Thymeleaf standard dialects in the application	There are Thymeleaf attributes in the HTML files like <code>th:action</code> , etc.
Integrate MyBatis into the application	There are annotations like <code>@Mapper</code> , <code>@Select</code> , <code>@Insert</code> , <code>@Update</code> , and <code>@Delete</code> in the Java classes and/or imports from MyBatis/iBatis API.
Write an application that will fail gracefully	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	
CRITERIA	MEETS SPECIFICATIONS
Develop a signup page	<p>The signup page already has input fields for all the data you need from the user, including username and password fields.</p> <p>Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission.</p>
Create a user signup workflow	<p>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.</p> <p>An error message is already present in the template, but should only be visible if an error occurred during signup.</p>
Develop a login page	<p>The login page already has the username and password fields.</p> <p>Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission.</p>
Create a user login/logout workflow	<p>On a successful login, the user should be taken to their home page.</p> <p>An error message is already present in the template, but should only be visible if an error occurred during signup.</p>
	<p>On logout, the user should no longer have access to the home page.</p>
Create a home page	<p>The home page should have three tabs:</p> <ol style="list-style-type: none"><li>1. The user should be able to upload new files on this tab and download/remove existing files</li><li>2. The user should be able to add new notes and edit/remove existing ones</li><li>3. The user should be able to add new credentials, view existing credentials unencrypted and remove them as well</li></ol> <p>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</p> <p>Details on individual features are documented in Section 3.</p>

## Project: SuperDuperDrive

User-Facing Features	
CRITERIA	MEETS SPECIFICATIONS
Implement persistent storage for users' important data	When a user logs in, they should see the data they have added to the application.
Implement note storage, edit, and removal	<p>Creation: On successful note creation, the user should be shown a success message and the created note should appear in the list.</p> <p>Deletion: On successful note deletion, the user should be shown a success message and the deleted note should disappear from the list.</p> <p>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.</p> <p>Errors: Users should be notified of errors if they occur.</p>
Implement file storage, download, and removal	<p>Upload: On successful file upload, the user should be shown a success message and the uploaded file should appear in the list.</p> <p>Deletion: On successful file deletion, the user should be shown a success message and the deleted file should disappear from the list.</p> <p>Download: On successful file download, the file should download to the user's system.</p> <p>Errors: Users should be notified of errors if they occur.</p>
Implement secure credential storage	<p>Creation: On successful credential creation, the user should be shown a success message and the created credential should appear in the list.</p> <p>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.</p> <p>Deletion: On successful credential deletion, the user should be shown a success message and the deleted credential should disappear from the list.</p> <p>Errors: Users should be notified of errors if they occur.</p>
Back-End	
CRITERIA	MEETS SPECIFICATIONS
Perform data validation and sanitization	The application should not allow duplicate usernames or duplicate filenames attributed to a single user.
Secure the application	<p>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.</p> <p>If someone isn't logged in, they must be redirected to the login page.</p>
A user can access only their own data	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.

## Project: SuperDuperDrive

The credentials are kept encrypted in the database	<p>All the passwords should be stored as encrypted in the database and shown as encrypted when the user retrieves them.</p> <p>The user should only see the decrypted version when they want to edit it.</p>
Implement an ORM model that maps to the database using MyBatis	<p>Create Java classes to model the tables in the database (specified in <code>src/main/resources/schema.sql</code>) and create <code>@Mapper</code> annotated interfaces to serve as Spring components in your application.</p> <p>You should have one model class and one mapper class per database table.</p>

### Testing

CRITERIA	MEETS SPECIFICATIONS
Test signup and login flow	<p>Write a Selenium test that verifies that the home page is not accessible without logging in.</p> <p>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.</p>
Test adding, editing, and deleting notes	<p>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.</p> <p>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.</p> <p>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.</p>
Test adding, editing and deleting credentials	<p>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.</p> <p>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.</p> <p>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.</p>

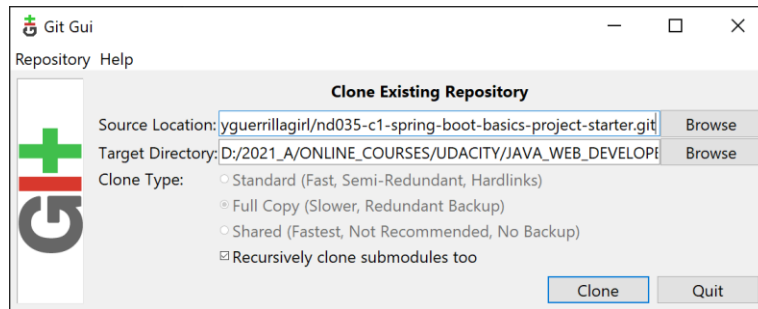
### Suggestions to Make Your Project Stand Out!

1. If a user knows the file, note, or credential ID of another user, make sure they can't make a direct request through the browser to view, edit, or delete that file, note, or credential.
2. Use test-driven-development.  
Write your selenium tests before implementing the functionality they're testing, and watch you tests go from red to green as you finish features!  
Use page objects to abstract selenium element selection and actions.  
Test file upload and download with selenium. This will require some extra research!  
Test everything! Verify all the requirements above with selenium tests, down to expected successes and failures in specific
3. Make it your own! You can replace the bootstrap CSS and JS libraries with a design framework of your choosing, and redesign the HTML templates to customize and redesign the website. Note: this could take a long time!

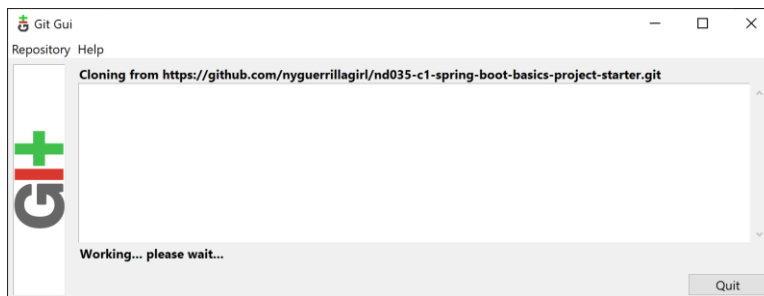
## How can I create an STS Workspace and Project from Github files?

- Open Git GUI and enter the Source and Target locations:

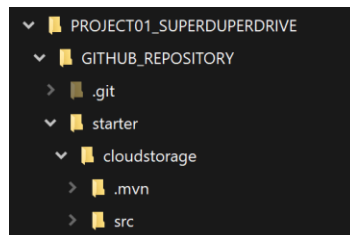
## Project: SuperDuperDrive



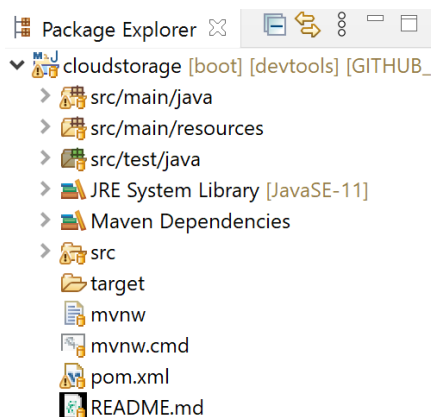
The Transfer should start:



The files on my hard drive:



I will try to create a Workspace at **starter** location and then import the Java Maven project CloudStorage from there.



It worked!

## What is the DBSchema?

We have 4 tables.

```
CREATE TABLE IF NOT EXISTS USERS (
  userid INT PRIMARY KEY auto_increment,
  username VARCHAR(20),
  salt VARCHAR,
  password VARCHAR,
  firstname VARCHAR(20),
  lastname VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS NOTES (
  noteid INT PRIMARY KEY auto_increment,
  notetitle VARCHAR(20),
  notedescription VARCHAR (1000),
  userid INT,
  foreign key (userid) references USERS(userid)
);

CREATE TABLE IF NOT EXISTS FILES (
  fileId INT PRIMARY KEY auto_increment,
  filename VARCHAR,
  contenttype VARCHAR,
  filesize VARCHAR,
  userid INT,
  filedata BLOB,
  foreign key (userid) references USERS(userid)
);

CREATE TABLE IF NOT EXISTS CREDENTIALS (
  credentialid INT PRIMARY KEY auto_increment,
  url VARCHAR(100),
  username VARCHAR (30),
  key VARCHAR,
  password VARCHAR,
  userid INT,
  foreign key (userid) references USERS(userid)
);
```

The USERS table is quite similar to the one we created for our chat app.

## How to create a persistent H2 database

I could not figure out how to create a persistent H2 database for the Chat application. I would like to have a persistent H2 database so I can:

- Test things from one run of the application to the next
- Keep user logins in place

I think I finally figured out the application.properties file I will need to make my H2 database persistent.

I worked out an example today using the project SPRING\_BOOT\_MYBATIS\_H2. This project demonstrated several features I would like to use for my project:

## Project: SuperDuperDrive

- H2 Persistence
  - The right settings in application.properties
- Using CommandLineRunner to test out my iBatis stuff
- Setting up for logging

The application.properties file I will use is the following:

```
# H2 Persistent DB
spring.datasource.url=jdbc:h2:file:./db/testdb;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

# Enabling H2 Console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.h2.console.settings.trace=false
spring.h2.console.settings.web-allow-others=true

# Set up logging
logging.file.name=./logs/spring-boot-app.log
logging.level.web=DEBUG
```

I am not sure if it is required but I also added the following to my pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
```

I was thinking of leaving out security but I think this is the easiest part to add since so much comes from the course project we have done.

## Adding Security

The plan shall be:

1. Add User Model and UserMapper ☼
2. Add UserService ☼
3. Add Security ☼
4. Add Thymeleaf elements to login.html and signup.html ☼
5. Add LoginController and SignupController (Empty HomeController) ☼
6. Test logging in and db entries are made ☼
7. Add Test cases for Login and Signup

## Add User Model and UserMapper

```
package com.udacity.jwdnd.coursel.cloudstorage.model;

public class User {
    private Integer userid;
```

```
private String username;
private String salt;
private String password;
private String firstname;
private String lastname;

public String toString() {
    StringBuffer sb = new StringBuffer();
    sb.append("userid: " + userid);
    sb.append(";username: " + username);
    sb.append(";salt: " + salt);
    sb.append(";password: " + password);
    sb.append(";firstname: " + firstname);
    sb.append(";lastname: " + lastname);

    return sb.toString();
}

public User() {
    // empty constructor
}

public User(Integer userid, String username, String salt, String password, String
firstname, String lastname) {
    this.userid = userid;
    this.username = username;
    this.salt = salt;
    this.password = password;
    this.firstname = firstname;
    this.lastname = lastname;
}

public Integer getUserid() {
    return userid;
}

public void setUserid(Integer userid) {
    this.userid = userid;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getSalt() {
    return salt;
}

public void setSalt(String salt) {
    this.salt = salt;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}
```



## Project: SuperDuperDrive

```
public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}
}
```

## The UserMapper

```
package com.udacity.jwdnd.course1.cloudstorage.mapper;

import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Options;
import org.apache.ibatis.annotations.Select;

import com.udacity.jwdnd.course1.cloudstorage.model.User;

@Mapper
public interface UserMapper {

    @Select("SELECT * FROM USERS WHERE username = #{username}")
    User getUserByName(String username);

    @Select("SELECT * FROM USERS WHERE userid = #{userid}")
    User getUserById(Integer userid);

    @Insert("INSERT INTO USERS (username, salt, password, firstname, lastname) VALUES (#{username},
#{salt}, #{password}, #{firstname}, #{lastname})")
    @Options(useGeneratedKeys = true, keyProperty = "userid")
    Integer insert(User user);

    @Delete("DELETE FROM USERS WHERE userid = #{userid}")
    void delete(Integer userid);
}
```

I plan on testing the above via a UserService (using Spring Boot CommandLineRunner) in order to test the above works.

## UserService

This class is the main way to interface with the USER table.

```
package com.udacity.jwdnd.course1.cloudstorage.services;

import java.security.SecureRandom;
import java.util.Base64;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.udacity.jwdnd.course1.cloudstorage.mapper.UserMapper;
import com.udacity.jwdnd.course1.cloudstorage.model.User;

public class UserService {
```

```

private static Logger logger = LoggerFactory.getLogger(UserService.class);

private final UserMapper userMapper;
private final HashService hashService;

public UserService(UserMapper userMapper, HashService hashService) {
    this.userMapper = userMapper;
    this.hashService = hashService;
}

public boolean isUsernameAvailable(String username) {
    logger.debug("==> isUsernameAvailable invoked with username: " + username);
    User aUser = userMapper.getUserByName(username);
    // TODO: Remove or comment out this debugging code
    if (aUser == null) {
        logger.debug("==> user record NOT FOUND.");
    } else {
        logger.debug("==> user record FOUND.");
    }
    return userMapper.getUserByName(username) == null;
}

public int createUser(User user) {
    logger.debug("==> createUser invoked with user: " + user.toString());
    SecureRandom random = new SecureRandom();
    byte[] salt = new byte[16];
    random.nextBytes(salt);
    String encodedSalt = Base64.getEncoder().encodeToString(salt);
    String hashedPassword = hashService.getHashedValue(user.getPassword(), encodedSalt);

    logger.debug("==> createUser inserting record using userMapper");
    return userMapper.insert(new User(null, user.getUsername(), encodedSalt,
hashedPassword, user.getFirstname(),
                                user.getLastname()));
}

public User getUser(String username) {
    logger.debug("==> getUser where username: " + username);
    return userMapper.getUserByName(username);
}

public List<User> getAllUsers() {
    logger.debug("==> getAllUsers");
    return userMapper.getAllUsers();
}
}

```

Let's test our UserService using a CommandLineRunner

```

package com.udacity.jwdnd.course1.cloudstorage;

import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.udacity.jwdnd.course1.cloudstorage.model.User;
import com.udacity.jwdnd.course1.cloudstorage.services.UserService;

/**
 * Use this to test UserMapper
 * @author lafig
 */

```

## Project: SuperDuperDrive

```
*/
@Component
public class UserCommandLineRunner implements CommandLineRunner {

    private static Logger logger = LoggerFactory.getLogger(UserCommandLineRunner.class);

    @Autowired
    private UserService userService;

    @Override
    public void run(String... args) throws Exception {
        logger.info("===> UserCommandLineRunner <===");
        //User firstUser = new User(null, "nyguerrillgirl", null, "password123", "Lorraine",
"Figueroa");
        User firstUser = new User(null, "samro65", null, "password123", "Samantha", "Neill");
        userService.createUser(firstUser);

        List<User> users = userService.getAllUsers();
        logger.info("===> users: " + users.toString());
    }
}
```

I ran twice and added two users:

```
2020-12-29 19:59:11.840 INFO 19924 --- [ restartedMain] c.u.j.c.c.UserCommandLineRunner : ===> UserCommandLineRunner <===
2020-12-29 19:59:11.892 INFO 19924 --- [ restartedMain] c.u.j.c.c.UserCommandLineRunner : ===> users: [[userid: 1;username:
nyguerrillgirl;salt: ilAXF6fLHawg+9oZJXYDEw==;password: X7hLbry1Ke7AvEQZWEIXRg==;firstname: Lorraine;lastname: Figueroa], [userid:
2;username: samro65;salt: Lfr4MIELAa6CLgUTvD3Jlw==;password: gxUQht2bY1UAd3WONHclGg==;firstname: Samantha;lastname: Neill]]
```

The UserMapper works fine.

Issue: I don't see logger.debug logging to the log file! Added to application.properties:

```
logging.level.com.udacity.jwdnd.course1.cloudstorage=DEBUG
```

## Add Security

First we will add AuthenticationService class:

```
package com.udacity.jwdnd.course1.cloudstorage.services;

import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.stereotype.Service;

import com.udacity.jwdnd.course1.cloudstorage.mapper.UserMapper;
import com.udacity.jwdnd.course1.cloudstorage.model.User;

@Service
public class AuthenticationService implements AuthenticationProvider {
```

## Project: SuperDuperDrive

```
@Autowired
private UserMapper userMapper;

@Autowired
private HashService hashService;

@Override
public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
    String username = authentication.getName();
    String password = authentication.getCredentials().toString();

    User user = userMapper.getUserByName(username);
    if (user != null) {
        String encodedSalt = user.getSalt();
        String hashedPassword = hashService.getHashedValue(password, encodedSalt);
        if (user.getPassword().equals(hashedPassword)) {
            return new UsernamePasswordAuthenticationToken(username, password, new
ArrayList<>());
        }
    }
    return null;
}

@Override
public boolean supports(Class<?> authentication) {
    return authentication.equals(UsernamePasswordAuthenticationToken.class);
}
}
```

Then we will add SecurityConfig class under config package:

```
package com.udacity.jwdnd.course1.cloudstorage.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import com.udacity.jwdnd.course1.cloudstorage.services.AuthenticationService;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private AuthenticationService authenticationService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) {
        auth.authenticationProvider(this.authenticationService);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/h2-console", "/signup", "/css/**", "/js/**").permitAll()
            .anyRequest().authenticated();
    }
}
```

```
        http.formLogin()
            .loginPage("/login")
            .permitAll();

        http.formLogin()
            .defaultSuccessUrl("/home", true);
    }
}
```

## Add Thymeleaf elements

### Signup

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <link rel="stylesheet" type="text/css" media="all" th:href="@{/css/bootstrap.min.css}">

    <title>Sign Up</title>
</head>
<body class="p-3 mb-2 bg-light text-black">
    <div class="container justify-content-center w-50 p-3" style="background-color: #eeeeee;
margin-top: 5em;">
        <div class="form-group">
            <label><a th:href="@{/Login}">Back to Login</a></label>
        </div>

        <h1 class="display-5">Sign Up</h1>
        <form action="#" th:action="@{/signup}" method="POST">
            <div id="success-msg" th:if="${signupSuccess}" class="alert alert-dark">
                You successfully signed up! Please continue to the <a>login</a> page.
            </div>
            <div id="error-msg" th:if="${signupError}" class="alert alert-danger">
                <span th:text="${signupError}">Example Signup Error Message</span>
            </div>
            <div class="form-row">
                <div class="form-group col-md-6">
                    <label for="inputFirstName">First Name</label>
                    <input type="input" name="firstname" class="form-control" id="inputFirstName"
placeholder="Enter First Name" maxlength="20" required>
                </div>
                <div class="form-group col-md-6">
                    <label for="inputLastName">Last Name</label>
                    <input type="input" name="lastname" class="form-control" id="inputLastName"
placeholder="Enter Last Name" maxlength="20" required>
                </div>
            </div>
            <div class="form-row">
                <div class="form-group col-md-6">
                    <label for="inputUsername">Username</label>
                    <input type="input" name="username" class="form-control" id="inputUsername"
placeholder="Enter Username" maxlength="20" required>
                </div>
                <div class="form-group col-md-6">
                    <label for="inputPassword">Password</label>
                    <input type="password" name="password" class="form-control" id="inputPassword"
placeholder="Enter Password" maxlength="20" required>
                </div>
            </div>
        </form>
    </div>
```

## Project: SuperDuperDrive

```
        <button id="submit-button" type="submit" class="btn btn-primary">Sign Up</button>
    </form>
</div>
</body>
</html>
```

## Login

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

        <link rel="stylesheet" type="text/css" media="all" th:href="@{/css/bootstrap.min.css}">

        <title>Login</title>
    </head>
    <body class="p-3 mb-2 bg-light text-black">
        <div class="container justify-content-center w-25 p-3" style="background-color: #eeeeee;
margin-top: 5em;">
            <h1 class="display-5">Login</h1>
            <form action="#" th:action="@{/Login}" method="POST">
                <div id="error-msg" th:if="${param.error}" class="alert alert-danger">
                    Invalid username or password
                </div>
                <div id="logout-msg" th:if="${param.logout}" class="alert alert-dark">
                    You have been logged out
                </div>
                <div class="form-group">
                    <label for="inputUsername">Username</label>
                    <input type="input" class="form-control" name="username" id="inputUsername"
placeholder="Enter Username" maxlength="20" required>
                </div>
                <div class="form-group">
                    <label for="inputPassword">Password</label>
                    <input type="password" class="form-control" name="password" id="inputPassword"
placeholder="Enter Password" maxlength="20" required>
                </div>
                <button id="submit-button" type="submit" class="btn btn-primary">Login</button>
            </form>

            <div class="form-group" style="margin-top: 0.5em;">
                <label><a id="signup-link" th:href="@{/signup}">Click here to sign up</a></label>
            </div>
        </div>
    </body>
</html>
```

- Create package “controller”

## SignupController

```
package com.udacity.jwdnd.course1.cloudstorage.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
```

```

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

import com.udacity.jwdnd.course1.cloudstorage.model.User;
import com.udacity.jwdnd.course1.cloudstorage.services.UserService;

@Controller
@RequestMapping("/signup")
public class SignupController {
    private static Logger logger = LoggerFactory.getLogger(SignupController.class);

    private final UserService userService;

    public SignupController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping
    String signupScreen(Model model) {
        logger.info("SignupController - signupScreen(GET)");
        return "signup";
    }

    @PostMapping
    public String signupUser(@ModelAttribute User user, Model model) {

        logger.info("SignupController - signupUser(POST)");
        String signupError = null;

        if (!userService.isUsernameAvailable(user.getUsername())) {
            signupError = "The username already exists.";
        }

        if (signupError == null) {
            logger.info("signupUser - creating the user...");
            int rowsAdded = userService.createUser(user);
            logger.info("==> signupUser - rowsAdded: " + rowsAdded);
            if (rowsAdded < 0) {
                signupError = "There was an error signing you up. Please try again.";
            }
        }

        if (signupError == null) {
            logger.info("signupUser - signupSuccess");
            model.addAttribute("signupSuccess", true);
        } else {
            logger.info("signupUser - signupError");
            model.addAttribute("signupError", signupError);
        }
        return "signup";
    }
}

```

## LoginController

```

package com.udacity.jwdnd.course1.cloudstorage.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/login")
public class LoginController {

```

## Project: SuperDuperDrive

```
@GetMapping
public String loginView() {
    return "login";
}

}
```

The above basically does nothing. We use Spring login capability and authentication via the AuthenticationService.

## HomeController

```
package com.udacity.jwdnd.course1.cloudstorage.controller;

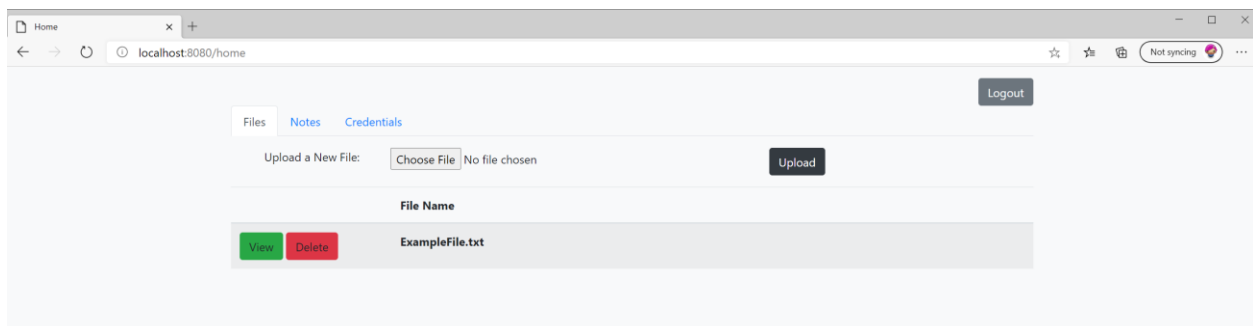
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/home")
public class HomeController {

    @GetMapping
    public String getHomePage() {
        return "home";
    }
}
```

## Building the HomeController

This is the main page that looks like this:



The first thing we will work on is introducing a /logout and have Spring handling all the heavy lifting. Using this reference: <https://stackoverflow.com/questions/36557294/spring-security-logout-does-not-work-does-not-clear-security-context-and-auth> we will add the following code to SecurityConfig:

```
http.logout().logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
.logoutSuccessUrl("/login").deleteCookies("JSESSIONID")
.invalidateHttpSession(true);
```



## Project: SuperDuperDrive

The code will execute Spring logout process and clear the session cookie and invalidate the session. Upon success the user shall be taken to the login screen.

Note: We are persisting the data for this project.

We need to be able to “show” the current list of:

- Files
- Notes
- Credentials

That belong to the user.

At this time they should be ALL empty and not showing the examples. So the first thing we need to do for all three tab sections:

- Add Thymeleaf notation for all three List of items
- Add to the HomeController ModelAttribute code so that all responses for logged in user shall always include the list of Files, Notes and Credential data

On the home.html page we have to find the html to generate the lists for each item. We will create a Service class for each list item: FileService, NoteService, and CredentialService. Each Service will have a corresponding model class – File, Note and Credential.

```
package com.udacity.jwdnd.course1.cloudstorage.model;

import java.sql.Blob;

public class File {

    private Integer fileId;
    private String filename;
    private String contenttype;
    private String filesize;
    private Integer userid;
    private Blob filedata;

    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("[fileId: " + fileId);
        sb.append(";filename: " + filename);
        sb.append(";contenttype: " + contenttype);
        sb.append(";filesize: " + filesize);
        sb.append(";userid: " + userid + "];");
        :
        :
    }
}
```

```
package com.udacity.jwdnd.course1.cloudstorage.model;

public class Note {

    private Integer noteid;
    private String notetitle;
    private String notedescription;
```

## Project: SuperDuperDrive

```
private Integer userid;

public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("[noteid: " + noteid);
    sb.append(";notetitle: " + notetitle);
    sb.append("\nnotedescription:\n" + notedescription + "\n");
    sb.append(";userid: " + userid + "]");

    return sb.toString();
}

:
:
```

```
package com.udacity.jwdnd.course1.cloudstorage.model;

public class Credential {

    private Integer credentialid;
    private String url;
    private String username;
    private String key;
    private String password;
    private Integer userid;

    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("[credentialid: " + credentialid);
        sb.append(";url: " + url);
        sb.append(";username: " + username);
        sb.append(";key: " + key);
        sb.append(";password: " + password);
        sb.append(";userid: " + userid + "]");
        return sb.toString();
    }

    :
    :
}
```

The above shall be used in the Mapper classes.

```
package com.udacity.jwdnd.course1.cloudstorage.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import com.udacity.jwdnd.course1.cloudstorage.model.File;

@Mapper
public interface FileMapper {

    @Select("SELECT * FROM FILES WHERE userid=#{userid}")
    public List<File> getAllUserFiles(Integer userid);
}
```

## Project: SuperDuperDrive

```
package com.udacity.jwdnd.course1.cloudstorage.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import com.udacity.jwdnd.course1.cloudstorage.model.Note;

@Mapper
public interface NoteMapper {

    @Select("SELECT * FROM CREDENTIALS WHERE userid=#{userid}")
    public List<Note> getAllUserNotes(Integer userid);

}
```

```
package com.udacity.jwdnd.course1.cloudstorage.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import com.udacity.jwdnd.course1.cloudstorage.model.Credential;

@Mapper
public interface CredentialMapper {

    @Select("SELECT * FROM CREDENTIALS WHERE userid=#{userid}")
    public List<Credential> getAllUserCredentials(Integer userid);

}
```

I added a similar method to each Service class:

```
@Service
public class FileService {

    @Autowired
    private FileMapper mapper;

    public List<File> getUsersStoredData(Integer userid) {
        return mapper.getAllUserFiles(userid);
    }

    :
    :
```

I updated the HomeController to populate the Model with each list:

```
package com.udacity.jwdnd.course1.cloudstorage.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
```

## Project: SuperDuperDrive

```
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

import com.udacity.jwdnd.course1.cloudstorage.common.CloudStorageConstants;
import com.udacity.jwdnd.course1.cloudstorage.model.User;
import com.udacity.jwdnd.course1.cloudstorage.services.UserService;
import com.udacity.jwdnd.course1.cloudstorage.services.storage.CredentialService;
import com.udacity.jwdnd.course1.cloudstorage.services.storage.FileService;
import com.udacity.jwdnd.course1.cloudstorage.services.storage.NoteService;

@Controller
@RequestMapping("/home")
public class HomeController {
    private static Logger logger = LoggerFactory.getLogger(HomeController.class);

    @Autowired
    private FileService fileService;

    @Autowired
    private NoteService noteService;

    @Autowired
    private CredentialService credentialService;

    @Autowired
    private UserService userService;

    @GetMapping
    public String getHomePage() {
        return "home";
    }

    @ModelAttribute
    public void allUserFiles(Authentication authentication, Model model) {
        String userName = authentication.getName();
        logger.info("HomeController - allUserFiles: " + userName);
        User userRecord = userService.getUser(userName);
        if (userRecord != null) {
            logger.info("HomeController.allUserFiles - userRecord found.");
            Integer userId = userRecord.getUserId();
            model.addAttribute(CloudStorageConstants.FILE_LIST,
                fileService.getUsersStoredData(userId));
            model.addAttribute(CloudStorageConstants.CREDENTIAL_LIST,
                credentialService.getUsersStoredData(userId));
            model.addAttribute(CloudStorageConstants.NOTE_LIST,
                noteService.getUsersStoredData(userId));
        }
    }
}
```

Update home.html with Thymeleaf to display each list or a message

Since these lists are currently empty (they should be) we will display a message in each tab for a null list.

“0 files stored under user account.”

“0 notes stored under user account.”

“0 credentials stored under user account.”

How to check if a list is empty in Thymeleaf:

Reference: <https://stackoverflow.com/questions/33106391/how-to-check-if-list-is-empty-using-thymeleaf>

You can do as follows:

112

```
<div th:if="${not #lists.isEmpty(tblUserList)}">
  --content--
</div>
```



`<div th:unless="${#lists.isEmpty(tblUserList)}">` is a bit more readable in my opinion. – [Watercolours](#) Feb 7 '17 at 12:11

With **Thymeleaf 3.x.x** you can validate list more elegant:

47

```
<div th:if="${tblUserList!=null and !tblUserList.empty}"></div>
```

or

```
<div th:if="${tblUserList!=null and !tblUserList.isEmpty()}"></div>
```

This is how the file section appears to the user:

Files

Notes

Credentials

Upload a New File:  No file chosen

File Name

View

Delete

ExampleFile.txt

In the code we have:

```
<div class="table-responsive">
  <table class="table table-striped" id="fileTable">
    <thead>
      <tr>
        <th style="width: 20%" scope="col"></th>
```

## Project: SuperDuperDrive

```
        <th style="width: 80%" scope="col">File Name</th>
      </tr>
    </thead>
    <tbody>
      <!-- THIS IS WHERE WE DISPLAY THE USER'S FILES -->

      <tr>

        <td>
          <a target="_blank" class="btn btn-success">View</a>
          <a class="btn btn-danger">Delete</a>
        </td>

        <th scope="row">ExampleFile.txt</th>
      </tr>
    </tbody>
  </table>
</div>
```

We will work on displaying either a message across the entire row or list the elements in the fileList.

Note: I created a CloudStorageConstants.java file with all the View/Model names that should remain consistent.

```
package com.udacity.jwdnd.course1.cloudstorage.common;

public class CloudStorageConstants {

    public static final String FILE_LIST = "fileList";
    public static final String NOTE_LIST = "noteList";
    public static final String CREDENTIAL_LIST = "credentialList";
}
```

Files	Notes	Credentials
<div>+ Add a New Note</div>		
Title	Description	
0 notes stored under user account		

And

Files	Notes	Credentials
<div>+ Add a New Credential</div>		
URL	Username	Password
0 password credentials stored under user account		

For testing purposes we will insert three dummy items to each list to make sure the object element appears in the right column.

## Project: SuperDuperDrive

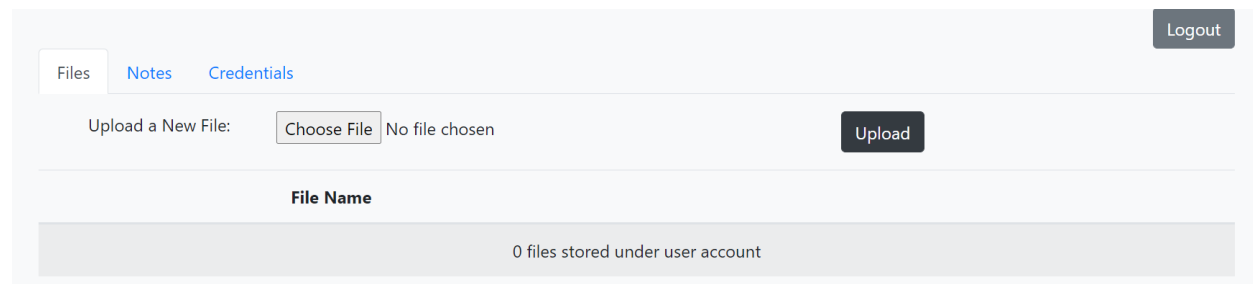
So far we have the following code on home.html

```
<tbody>
<!-- THIS IS WHERE WE DISPLAY THE USER'S FILES -->
<tr th:if="${fileList==null or fileList.size() == 0}">
<td colspan="2" align="center">
0 files stored under user account
</td>
<tr th:if="${fileList!=null and fileList.size() > 0}">

<td>
<a target="_blank" class="btn btn-success">View</a>
<a class="btn btn-danger">Delete</a>
</td>

<th scope="row">ExampleFile.txt</th>
</tr>
</tbody>
```

And we see:



This is what we want. Let's do the same for Notes and Credentials.

I will now add three dummy items to each list to see it displayed. I created a TestSupportUtil.java class to use to test the tabs for file, notes and credentials having three items each to display.

Updated allUserFiles method in HomeController:

```
@ModelAttribute
public void allUserFiles(Authentication authentication, Model model) {
    String userName = authentication.getName();
    Logger.info("HomeController - allUserFiles: " + userName);
    User userRecord = userService.getUser(userName);
    if (userRecord != null) {
        Logger.info("HomeController.allUserFiles - userRecord found.");
        Integer userId = userRecord.getUserId();
        model.addAttribute(CloudStorageConstants.FILE_LIST,
            TestSupportUtil.getDummyFileList());
        model.addAttribute(CloudStorageConstants.CREDENTIAL_LIST,
            TestSupportUtil.getDummyCredentialList());
        model.addAttribute(CloudStorageConstants.NOTE_LIST,
            TestSupportUtil.getDummyNoteList());
        //model.addAttribute(CloudStorageConstants.FILE_LIST,
        fileService.getUsersStoredData(userId));
        //model.addAttribute(CloudStorageConstants.CREDENTIAL_LIST,
        credentialService.getUsersStoredData(userId));
        //model.addAttribute(CloudStorageConstants.NOTE_LIST,
        noteService.getUsersStoredData(userId));
    }
}
```

## Project: SuperDuperDrive

I hard coded values for each section and we see this:

File Tab:

Files

Notes

Credentials

Upload a New File: 

Choose File

 No file chosen 

Upload

File Name	
<div>View</div> <div>Delete</div>	FILE_ENTRY_1.txt
<div>View</div> <div>Delete</div>	FILE_ENTRY_2.txt
<div>View</div> <div>Delete</div>	FILE_ENTRY_3.txt

Notes Tab:

Files

Notes

Credentials

Logout

+ Add a New Note

	Title	Description
<div>Edit</div> <div>Delete</div>	JWDND Note 1	This is a reminder to work on project 1 everyday until it is done.
<div>Edit</div> <div>Delete</div>	JWDND Note 2	Start Lesson 3 on Java Web Services with REST/API/Microservices
<div>Edit</div> <div>Delete</div>	JWDND Note 3	Check all provided references.

Credentials Tab:

Files

Notes

Credentials

+ Add a New Cred

	URL	Username	Password
<div>Edit</div> <div>Delete</div>	http://www.brainycode.com	bob	bobbybrownToday2021
<div>Edit</div> <div>Delete</div>	http://www.youtube.com	samantha	toss_me_up_put_me_down!
<div>Edit</div> <div>Delete</div>	http://www.packtpub.com	freeme	password123



- Restore the HomeController code, that is remove the hardcoded values:

```
package com.udacity.jwdnd.course1.cloudstorage.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

import com.udacity.jwdnd.course1.cloudstorage.common.CloudStorageConstants;
import com.udacity.jwdnd.course1.cloudstorage.model.User;
import com.udacity.jwdnd.course1.cloudstorage.services.UserService;
import com.udacity.jwdnd.course1.cloudstorage.services.storage.CredentialService;
import com.udacity.jwdnd.course1.cloudstorage.services.storage.FileService;
import com.udacity.jwdnd.course1.cloudstorage.services.storage.NoteService;

@Controller
@RequestMapping("/home")
public class HomeController {
    private static Logger logger = LoggerFactory.getLogger(HomeController.class);

    @Autowired
    private FileService fileService;

    @Autowired
    private NoteService noteService;

    @Autowired
    private CredentialService credentialService;

    @Autowired
    private UserService userService;

    @GetMapping
    public String getHomePage() {
        return "home";
    }

    @ModelAttribute
    public void allUserFiles(Authentication authentication, Model model) {
        String userName = authentication.getName();
        logger.info("HomeController - allUserFiles: " + userName);
        User userRecord = userService.getUser(userName);
        if (userRecord != null) {
            logger.info("HomeController.allUserFiles - userRecord found.");
            Integer userId = userRecord.getUserId();
            model.addAttribute(CloudStorageConstants.FILE_LIST,
                fileService.getUsersStoredData(userId));
            model.addAttribute(CloudStorageConstants.CREDENTIAL_LIST,
                credentialService.getUsersStoredData(userId));
            model.addAttribute(CloudStorageConstants.NOTE_LIST,
                noteService.getUsersStoredData(userId));
        }
    }
}
```

All the tabs are now empty.

The changes to the home.html file to support this so far:

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <link rel="stylesheet" type="text/css" media="all" th:href="@{/css/bootstrap.min.css}">

    <title>Home</title>
  </head>
  <body class="p-3 mb-2 bg-light text-black">
    <div class="container">
      <div id="LogoutDiv">
        <form action="#" method="POST" th:action="@{/Logout}">
          <button type="submit" class="btn btn-secondary float-right">Logout</button>
        </form>
      </div>
      <div id="contentDiv" style="clear: right;">
        <nav style="clear: right;">
          <div class="nav nav-tabs" id="nav-tab" role="tablist">
            <a class="nav-item nav-link active" id="nav-files-tab" data-toggle="tab"
href="#nav-files" role="tab" aria-controls="nav-files" aria-selected="true">Files</a>
            <a class="nav-item nav-link" id="nav-notes-tab" data-toggle="tab" href="#nav-
notes" role="tab" aria-controls="nav-notes" aria-selected="false">Notes</a>
            <a class="nav-item nav-link" id="nav-credentials-tab" data-toggle="tab"
href="#nav-credentials" role="tab" aria-controls="nav-credentials" aria-
selected="false">Credentials</a>
          </div>
        </nav>
        <div class="tab-content" id="nav-tabContent">
          <div class="tab-pane fade show active" id="nav-files" role="tabpanel" aria-
labelledby="nav-files-tab">
            <form action="#" enctype="multipart/form-data" method="POST">
              <div class="container">
                <div class="row" style="margin: 1em;">
                  <div class="col-sm-2">
                    <label for="fileUpload">Upload a New File:</label>
                  </div>
                  <div class="col-sm-6">
                    <input type="file" class="form-control-file" id="fileUpload"
name="fileUpload">
                  </div>
                  <div class="col-sm-4">
                    <button type="submit" class="btn btn-dark">Upload</button>
                  </div>
                </div>
              </div>
            </form>
            <div class="table-responsive">
              <table class="table table-striped" id="fileTable">
                <thead>
                  <tr>
                    <th style="width: 20%" scope="col"></th>
                    <th style="width: 80%" scope="col">File Name</th>
                  </tr>
                </thead>
                <tbody>
                  <!-- THIS IS WHERE WE DISPLAY THE USER'S FILES -->
                  <tr th:if="${fileList==null or fileList.size() == 0}">
                    <td colspan="2" align="center">
                      0 files stored under user account
                    </td>
                  </tr>
                  <tr th:each="fileEntry : ${fileList}" th:if="${fileList!=null and
fileList.size() > 0}">
                    <td>

```

```

                <a target="_blank" class="btn btn-success">View</a>
                <a class="btn btn-danger">Delete</a>
            </td>
        </tr>
        <tr>
            <th scope="row">ExampleFile.txt</th>
        </tr>
    </tbody>
</table>
</div>
</div>
<div class="tab-pane fade" id="nav-notes" role="tabpanel" aria-labelledby="nav-
notes-tab">
    <button style="margin: 0.25em;" type="button" class="btn btn-info float-right"
onclick="showNoteModal()">
        + Add a New Note
    </button>

    <div class="table-responsive">
        <table class="table table-striped" id="userTable">
            <thead>
                <tr>
                    <th style="width: 20%" scope="col"></th>
                    <th style="width: 20%" scope="col">Title</th>
                    <th style="width: 60%" scope="col">Description</th>
                </tr>
            </thead>
            <tbody>
                <!-- THIS IS WHERE WE PLACE THE NOTES DATA -->

                <tr th:if="{noteList==null or noteList.size() == 0}">
                    <td colspan="3" align="center">
                        0 notes stored under user account
                    </td>
                </tr>
                <tr th:each="noteEntry : {noteList}" th:if="{noteList!=null and
noteList.size() > 0}">
                    <td>
                        <button type="button" class="btn btn-success">Edit</button>
                        <a class="btn btn-danger">Delete</a>
                    </td>
                    <th scope="row" th:text="{noteEntry.notetitle}">Example Note
Title</th>
                    <td th:text="{noteEntry.notedescription}">Example Note
Description </td>
                </tr>
            </tbody>
        </table>
    </div>

    <!-- THIS IS THE NOTE ENTRY DIALOG WHEN USER PRESSES
"Add a new Note" -->
    <div class="modal fade" id="noteModal" tabindex="-1" role="dialog" aria-
labelledby="noteModalLabel" aria-hidden="true">
        <div class="modal-dialog" role="document">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title" id="noteModalLabel">Note</h5>
                    <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
                        <span aria-hidden="true">&times;</span>
                    </button>
                </div>
                <div class="modal-body">
                    <form action="#" method="POST">
                        <input type="hidden" name="noteId" id="note-id">
                        <div class="form-group">
                            <label for="note-title" class="col-form-
Label">Title</label>

```

```

        <input type="text" name="noteTitle" class="form-
control" id="note-title" maxlength="20" required>
    </div>
    <div class="form-group">
        <label for="note-description" class="col-form-
Label">Description</label>
        <textarea class="form-control" name="noteDescription"
id="note-description" rows="5" maxlength="1000" required></textarea>
    </div>
    <button id="noteSubmit" type="submit" class="d-
none"></button>

    </form>
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-
dismiss="modal">Close</button>
    <button type="button" class="btn btn-primary"
onclick="$('#noteSubmit').click();">Save changes</button>
</div>
</div>
</div>
</div>
<div class="tab-pane fade" id="nav-credentials" role="tabpanel" aria-
labelledby="nav-credentials-tab">
    <button style="margin: 0.25em;" type="button" class="btn btn-info float-right"
onclick="showCredentialModal()">
        + Add a New Credential
    </button>

    <div class="table-responsive">
        <table class="table table-striped" th:object="${credentials}"
id="credentialTable">
            <thead>
                <tr>
                    <th style="width: 20%" scope="col"></th>
                    <th style="width: 35%" scope="col">URL</th>
                    <th style="width: 20%" scope="col">Username</th>
                    <th style="width: 25%" scope="col">Password</th>
                </tr>
            </thead>
            <tbody>
                <tr th:if="${credentialList==null or credentialList.size() == 0}">
                    <td colspan="4" align="center">
                        0 password credentials stored under user account
                    </td>
                </tr>

                <tr th:each="credentialEntry : ${credentialList}"
th:if="${credentialList!=null and credentialList.size() > 0}">
                    <td>
                        <button type="button" class="btn btn-success">Edit</button>
                        <a class="btn btn-danger">Delete</a>
                    </td>
                    <th scope="row" th:text="${credentialEntry.url}">Example Credential
URL</th>
                    <td th:text="${credentialEntry.username}">Example Credential
Username</td>
                    <td th:text="${credentialEntry.password}">Example Credential
Password</td>
                </tr>
            </tbody>
        </table>
    </div>

    <!-- THIS IS THE MODAL DIALOG TO FILL IN WHEN USER
CLICKS ON "Add a New Credential" -->
    <div class="modal fade" id="credentialModal" tabindex="-1" role="dialog" aria-
labelledby="credentialModalLabel" aria-hidden="true">

```

```
<div class="modal-dialog" role="document">  
  <div class="modal-content">  
    <div class="modal-header">  
      <h5 class="modal-title">  
  
id="credentialModalLabel">Credential</h5>  
      <button type="button" class="close" data-dismiss="modal" aria-  
label="Close">  
  
        <span aria-hidden="true">&times;</span>  
      </button>  
    </div>  
    <div class="modal-body">  
      <form action="#" method="POST">  
        <input type="hidden" name="credentialId" id="credential-  
id">  
  
        <div class="form-group">  
          <label for="note-title" class="col-form-  
Label">URL</label>  
  
          <input type="text" name= "url" class="form-control"  
id="credential-url" maxlength="100" required>  
        </div>  
        <div class="form-group">  
          <label for="note-title" class="col-form-  
Label">Username</label>  
  
          <input type="text" name= "username" class="form-  
control" id="credential-username" maxlength="30" required>  
        </div>  
        <div class="form-group">  
          <label for="note-title" class="col-form-  
Label">Password</label>  
  
          <input type="text" name= "password" class="form-  
control" id="credential-password" maxlength="30" required>  
        </div>  
        <button id="credentialSubmit" type="submit" class="d-  
none"></button>  
  
      </form>  
    </div>  
    <div class="modal-footer">  
      <button type="button" class="btn btn-secondary" data-  
dismiss="modal">Close</button>  
  
      <button type="button" class="btn btn-primary"  
onclick="$('#credentialSubmit').click();">Save changes</button>  
    </div>  
  </div>  
</div>  
</div>  
</div>  
</div>  
</div>  
</div>  
  
<script th:src="@{/js/jquery-slim.min.js}"></script>  
<script th:src="@{/js/popper.min.js}"></script>  
<script th:src="@{/js/bootstrap.min.js}"></script>  
  
<!--For opening the note modal-->  
<script type="text/javascript">  
  // For opening the note modal  
  function showNoteModal(noteId, noteTitle, noteDescription) {  
    $('#note-id').val(noteId ? noteId : '');  
    $('#note-title').val(noteTitle ? noteTitle : '');  
    $('#note-description').val(noteDescription ? noteDescription : '');  
    $('#noteModal').modal('show');  
  }  
  
  // For opening the credentials modal  
  function showCredentialModal(credentialId, url, username, password) {  
    $('#credential-id').val(credentialId ? credentialId : '');  
    $('#credential-url').val(url ? url : '');  
    ($('#credential-username').val(username ? username : ''));
```

```
        $('#credential-password').val(password ? password : '');
        $('#credentialModal').modal('show');
    }
</script>
</body>
</html>
```

## Add Testcases for All the page

- Using the same SignupPage we used during the course:

```
package com.udacity.jwdnd.course1.cloudstorage;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class SignupPage {

    @FindBy(css = "#inputFirstName")
    private WebElement firstNameField;

    @FindBy(css = "#inputLastName")
    private WebElement lastNameField;

    @FindBy(css = "#inputUsername")
    private WebElement usernameField;

    @FindBy(css = "#inputPassword")
    private WebElement passwordField;

    @FindBy(css = "#submit-button")
    private WebElement submitButton;

    public SignupPage(WebDriver webDriver) {
        PageFactory.initElements(webDriver, this);
    }

    public void signup(String firstName, String lastName, String username, String password) {
        this.firstNameField.sendKeys(firstName);
        this.lastNameField.sendKeys(lastName);
        this.usernameField.sendKeys(username);
        this.passwordField.sendKeys(password);
        this.submitButton.click();
    }
}
```

- Checked the elements in the above file match the id elements in the file signup.html
- Copy LoginPage from course project:

```
package com.udacity.jwdnd.course1.cloudstorage;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {
```

## Project: SuperDuperDrive

```
@FindBy(css="#inputUsername")
private WebElement usernameField;

@FindBy(css="#inputPassword")
private WebElement passwordField;

@FindBy(css="#submit-button")
private WebElement submitButton;

public LoginPage(WebDriver webDriver) {
    PageFactory.initElements(webDriver, this);
}

public void login(String username, String password) {
    this.usernameField.sendKeys(username);
    this.passwordField.sendKeys(password);
    this.submitButton.click();
}
}
```

- Check elements match the page element ids – check
- Create a HomePage java file

Project hints from forum:

- File size:

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

- DTO Pattern

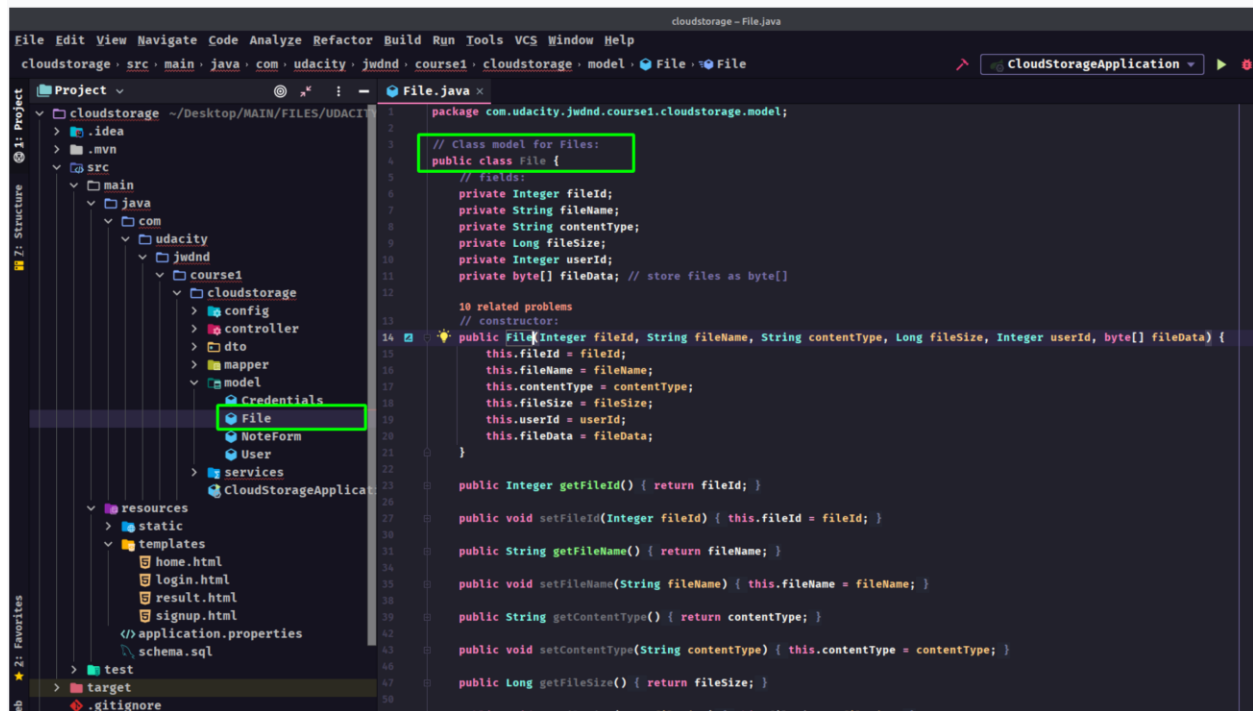
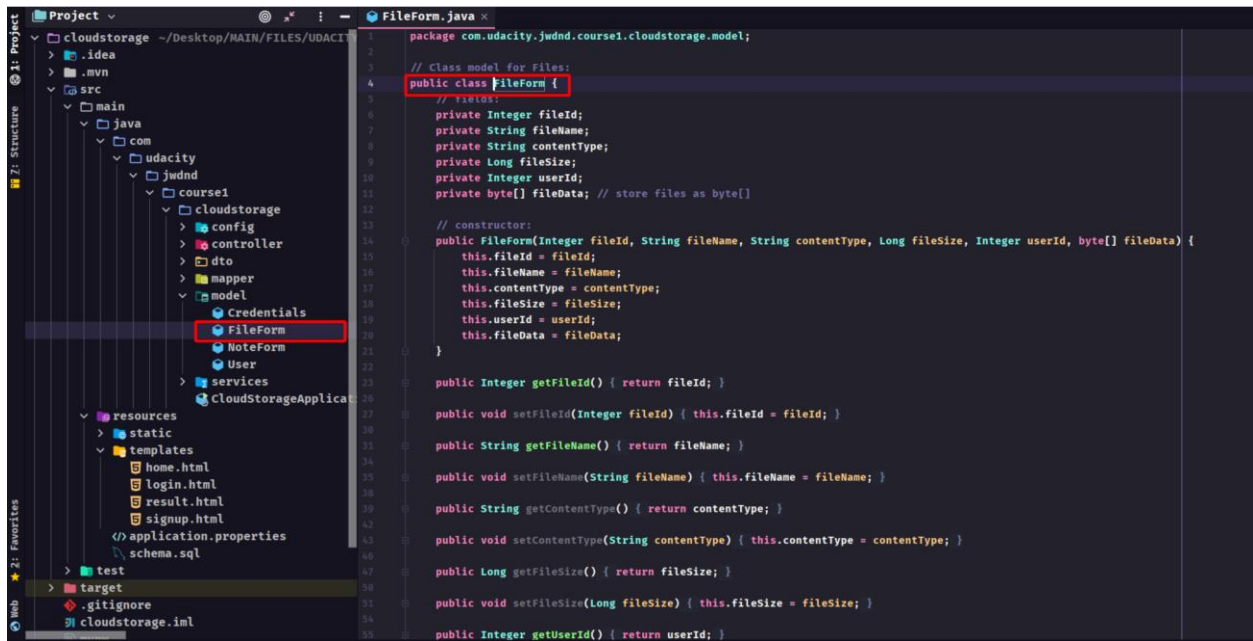
1.1° Use of the DTO pattern (Data Transfer Object)

The **DTO pattern** is widely used in cases where we **need to transport data from one layer to another** and we will make use of it here, for that we will **create a package called dto** and we will **create** the classes that will **serve to transport the frontend data sent by the user through the form to the backend** as you can see **below**:





## Project: SuperDuperDrive



## 2 - Error

### 2.1° Change FileForm to Form

**Before** we can solve the problem with the file upload we need to change all the classes that had FileForm to File in the project, as you can see below:

## 2.2° home.html

At **home.html** we will **put** the **th:object="\${fileDTO}"** and use the **th:field="\*{file}"** to **specify** the **property** that we **want to map to FileDTO** and that will be **sent to the backend**.

These **changes** can be seen **below**:

```

1: Project
2: Structure
3: home.html
4:
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965:
966:
967:
968:
969:
970:
971:
972:
973:
974:
975:
976:
977:
978:
979:
980:
981:
982:
983:
984:
985:
986:
987:
988:
989:
990:
991:
992:
993:
994:
995:
996:
997:
998:
999:
1000:

```

## 2.3° FilesController

In **FilesController** we will **create** the **method getFileDTO** that will be **responsible for including our FileDTO** in the **home.html** page so that we can **use it in the form** and we will **use @ModelAttribute("fileDTO")** in the **file property** of the **method uploadNewFile** so that we can **map the data coming from the form correctly**.

These **changes** can be seen **below**:

```

1: Project
2: Structure
3: FilesController.java
4:
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965:
966:
967:
968:
969:
970:
971:
972:
973:
974:
975:
976:
977:
978:
979:
980:
981:
982:
983:
984:
985:
986:
987:
988:
989:
990:
991:
992:
993:
994:
995:
996:
997:
998:
999:
1000:

```

```

package com.udacity.jwndnd.course1.cloudstorage.controller;
2
3
import com.udacity.jwndnd.course1.cloudstorage.dto.FileDTO;
4
import com.udacity.jwndnd.course1.cloudstorage.services.FilesService;

```

## Project: SuperDuperDrive

```
5 import com.udacity.jwdnd.coursel.cloudstorage.services.UserService;
6 import org.springframework.security.core.Authentication;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.web.bind.annotation.ModelAttribute;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.multipart.MultipartFile;
12
13 import java.io.IOException;
14
15 @Controller
16 public class FilesController {
17     // fields, use FileService, UserService:
18     private FileService fileService;
19     private UserService userService;
20
21     // constructor:
22     public FilesController(FileService fService, UserService uService) {
23         this.fileService = fService;
24         this.userService = uService;
25     }
26
27     @ModelAttribute("fileDTO")
28     public FileDTO getFileDTO() {
29         return new FileDTO();
30     }
31
32     // POST to upload new file:
33     @PostMapping("/home/file/newFile")
34     public String uploadNewFile(Authentication auth, Model model, @ModelAttribute("fileDTO")
35         MultipartFile file) throws IOException {
36         String errorMsg = null;
37
38         int currentUserId = this.userService.getUserById(auth.getName());
39
40         System.out.println("NAME: " + file.getName());
41
42         // handle edge cases:
43
44         // when empty file is uploaded:
```

```
45
46 if (file.isEmpty()) {
47     errorMsg = "File should not be empty!";
48 }
49
50 if (errorMsg == null) {
51     // upload file to Files db by fileId:
52     // return current fileId if success:
53     int currentFileId = this.filesService.uploadFile(file, currentUserId);
54     if (currentFileId < 0) {
55         errorMsg = "There was error uploading this file!";
56     }
57 }
58
59 // show result.html page with success/fail message:
60 if (errorMsg == null) {
61     model.addAttribute("updateSuccess", true);
62 } else {
63     model.addAttribute("updateFail", errorMsg);
64 }
65
66 return "result";
67
68 }
```

## 2.4° HomeController

In **HomeController** you will also **have to include the getFileDTO method** so that you do **not receive the BindingResultError error**.

This **modification** can be seen **below**:

```

1: Project
2: Favorites
HomeController.java x
26 private FileService fileService;
27
28 // constructor:
29 public HomeController(UserService userService, NotesService nService, FileService fService) {
30     this.userService = userService;
31     this.notesService = nService;
32     this.fileService = fService;
33 }
34
35 // initialize variables:
36 private List<NoteForm> noteList = new ArrayList<>();
37 private List<File> fileList = new ArrayList<>();
38
39 @ModelAttribute("fileDTO")
40 public FileDTO getFileDTO() { return new FileDTO(); }
41
42
43
44 @GetMapping
45 public String getHomePage(Authentication auth, Model model) {
46
47     // get userId using UserService and Authentication by username:
48     int userId = this.userService.getUserById(auth.getName());
49
50     // use NotesService to get all notes from Notes db by userId:
51     noteList = this.notesService.getAllNotes(userId);
52     // use FileService to get all files from Files db by userId:
53     fileList = this.fileService.getAllFiles(userId);
54
55     // display to View layer where th:object=${notes}
56     model.addAttribute(s: "notes", noteList);
57     model.addAttribute(s: "files", fileList);
58
59     return "home";
60 }
61

```

```

package com.udacity.jwdnd.coursel.cloudstorage.controller;
1
2
3
import com.udacity.jwdnd.coursel.cloudstorage.dto.FileDTO;
4
import com.udacity.jwdnd.coursel.cloudstorage.model.File;
5
import com.udacity.jwdnd.coursel.cloudstorage.model.NoteForm;
6
import com.udacity.jwdnd.coursel.cloudstorage.services.FileService;
7
import com.udacity.jwdnd.coursel.cloudstorage.services.NotesService;
8
import com.udacity.jwdnd.coursel.cloudstorage.services.UserService;
9
import org.springframework.security.core.Authentication;
10
import org.springframework.stereotype.Controller;
11
import org.springframework.ui.Model;
12
import org.springframework.web.bind.annotation.GetMapping;
13
import org.springframework.web.bind.annotation.ModelAttribute;
14
import org.springframework.web.bind.annotation.RequestMapping;
15

```

## Project: SuperDuperDrive

```
16 import java.util.ArrayList;
17
18 import java.util.List;
19
20 @Controller
21 @RequestMapping("/home")
22
23 public class HomeController {
24
25     // fields include all services:
26
27     private UserService userService;
28
29     private NotesService notesService;
30
31     private FilesService fileService;
32
33     // constructor:
34
35     public HomeController(UserService uService, NotesService nService, FilesService fService) {
36
37         this.userService = uService;
38
39         this.notesService = nService;
40
41         this.fileService = fService;
42
43     }
44
45     // initialize variables:
46
47     private List<NoteForm> noteList = new ArrayList<>();
48
49     private List<File> fileList = new ArrayList<>();
50
51     @ModelAttribute("fileDTO")
52
53     public FileDTO getFileDTO() {
54
55         return new FileDTO();
56
57     }
58
59     @GetMapping
60
61     public String getHomePage(Authentication auth, Model model) {
62
63         // get userId using UserService and Authentication by username:
64
65         int userId = this.userService.getUserById(auth.getName());
66
67         // use NotesService to get all notes from Notes db by userId:
68
69         noteList = this.notesService.getAllNotes(userId);
70
71         // use FilesService to get all files from Files db by userId:
72
73         fileList = this.fileService.getAllFiles(userId);
74
75         // display to View layer where th:object=${notes}
```

```
56
model.addAttribute("notes", noteList);
57
model.addAttribute("files", fileList);
58
59
return "home";
60
}
61
}
```

CHECK: <https://spring.io/guides/gs/uploading-files/>

<https://knowledge.udacity.com/questions/377483>

## Cannot redirect to result screen after File Upload Size Limit Exception exceeded



Veaceslav G about 2 months ago

Hi All,

Like a lot of people here, I also faced the `FileSizeLimitExceededException` when working on the cloudstorage app. From what I read, this can be mitigated by increasing the `max-file-size/max-request-size`, however, files that exceed the maximum size will still throw the `FileSizeLimitExceededException`.

What I want, is to handle the exception and redirect the user to the `/result.html` page, with a boolean flag (`success/true`) and an error message saying that the file could not be uploaded.

So I created a new `ControllerAdvice` that implements `HandlerExceptionResolver`, and specifically handle the `MaxUploadSizeExceededException` exception:



```
1  @ControllerAdvice
2  public class ExceptionController implements
   HandlerExceptionResolver {
3      @Override
4      public ModelAndView resolveException(
5          HttpServletRequest httpServletRequest,
6          HttpServletResponse httpServletResponse,
7          Object o,
8          Exception e) {
9
10         System.out.println("Exception: ");
11         System.out.println(e);
12
13         if (e instanceof MaxUploadSizeExceededException) {
14             System.out.println("Updating the model");
15
16             ModelAndView modelAndView = new
17 ModelAndView("result.html");
18             modelAndView.addObject("result_success", false);
19             modelAndView.addObject("error_message", "Not
20 possible to upload as the file size limit exceeded.");
21             return modelAndView;
22         }
23
24         return new ModelAndView("home.html");
25     }
26 }
```

I have added some logs, and I can see that the exception is handled, but despite that, I am not redirected to the result.html page, instead the browser says that the request could not be processed (connection was reset).

Why is it that I am unable to redirect to result.html page?  
Thank you.

This is an **error** that is **not related** to **Spring** but to the **Tomcat** default **maxSwallowSize connector**.

So in order to **solve this error** you need to **add** in **application.properties** the

```
server.tomcat.max-swallow-size=-1
```