# 1- Project Overview

**Super*Duper*Drive Cloud Storage**

You have been hired by Super*Duper*Drive, 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.

Super*Duper*Drive 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](https://github.com/udacity/nd035-c1-spring-boot-basics-project-starter/tree/master/starter/cloudstorage).

### 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](https://git-scm.com/downloads), 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](https://github.com/udacity/nd035-c1-spring-boot-basics-project-starter/tree/master/starter/cloudstorage) 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. (done)
* 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. (done)
* You also need to implement a custom AuthenticationProvider which authorizes user logins by matching their credentials against those stored in the database. (done)

#### 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. (done)
* 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. (done)
* 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! (done)
* 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! (done)

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

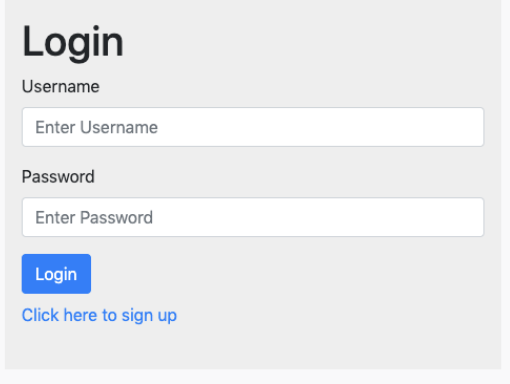
* Since you were provided with a database schema to work with, you can design Java classes to match the data in the database. These should be POJOs (Plain Old Java Objects) with fields that match the names and data types in the schema, and you should create one class per database table. These classes typically are placed in a model or entity package. (done)
* 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. (done)

### 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

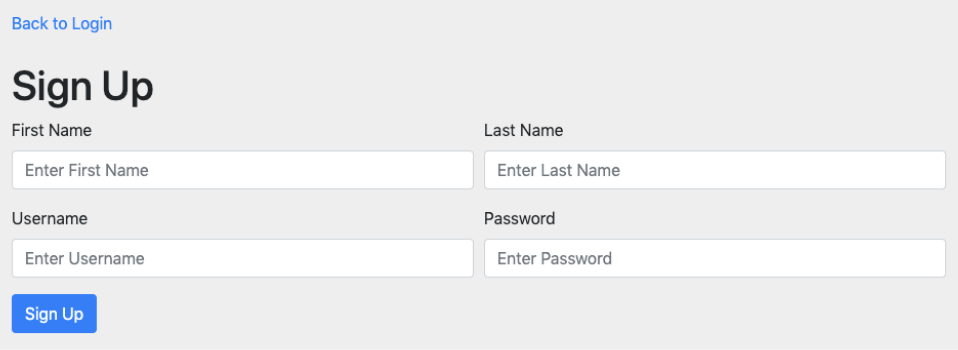
* Everyone should be allowed access to this page, and users can use this page to login to the application. (done)
* Show login errors, like invalid username/password, on this page. (done)



Login page: <http://localhost:8080/login>

#### **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. (done)
* Validate that the username supplied does not already exist in the application, and show such signup errors on the page when they arise. (done)
* Remember to store the user's password securely! (done)



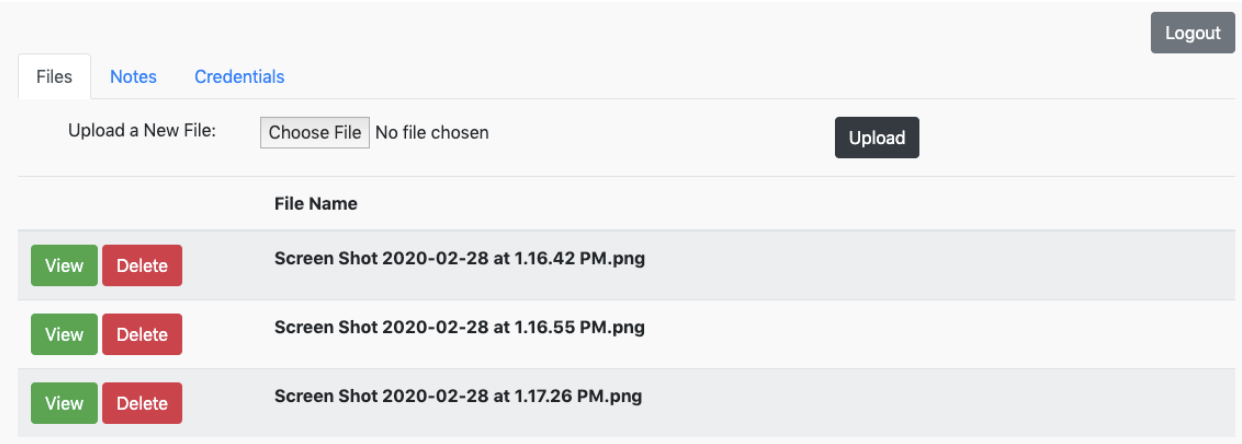
Sign Up page: <http://localhost:8080/signup>

#### 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. (done)
* 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

* + The user should be able to upload files and see any files they previously uploaded. (done)

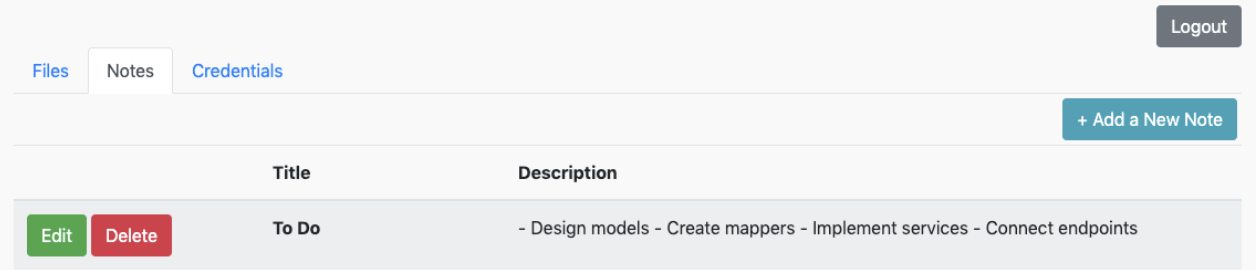


Home page: <http://localhost:8080/home>

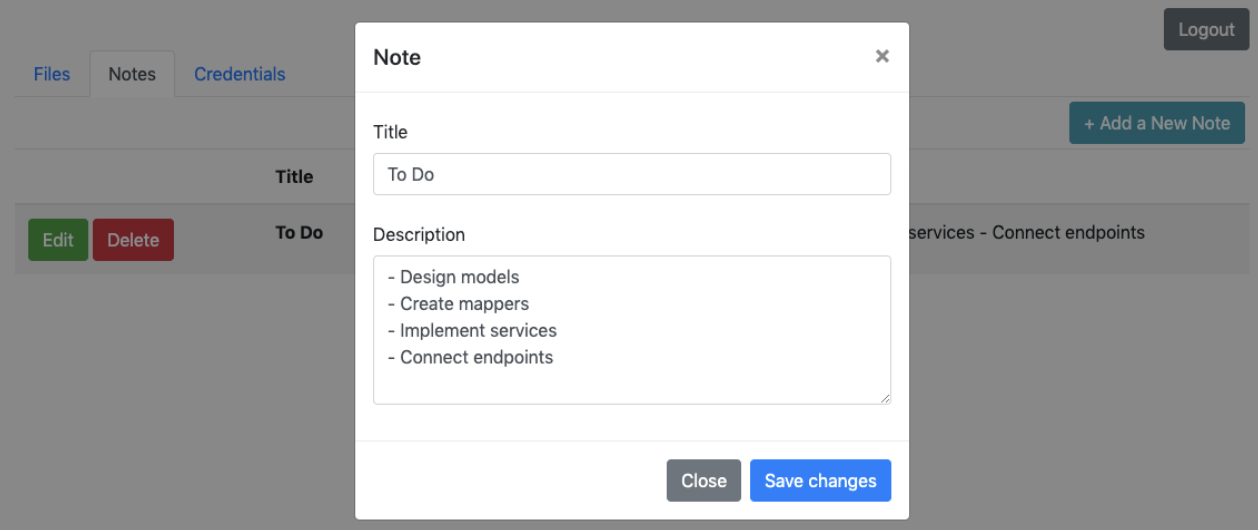
* The user should be able to view/download or delete previously-uploaded files. (done)
* Any errors related to file actions should be displayed. For example, a user should not be able to upload two files with the same name, but they'll never know unless you tell them! (done)

ii. Notes

* The user should be able to create notes and see a list of the notes they have previously created. (done)

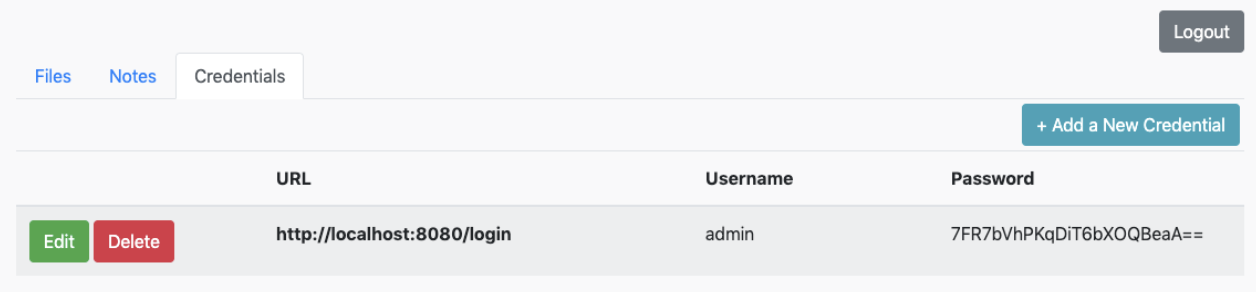


* The user should be able to edit or delete previously-created notes. (done)

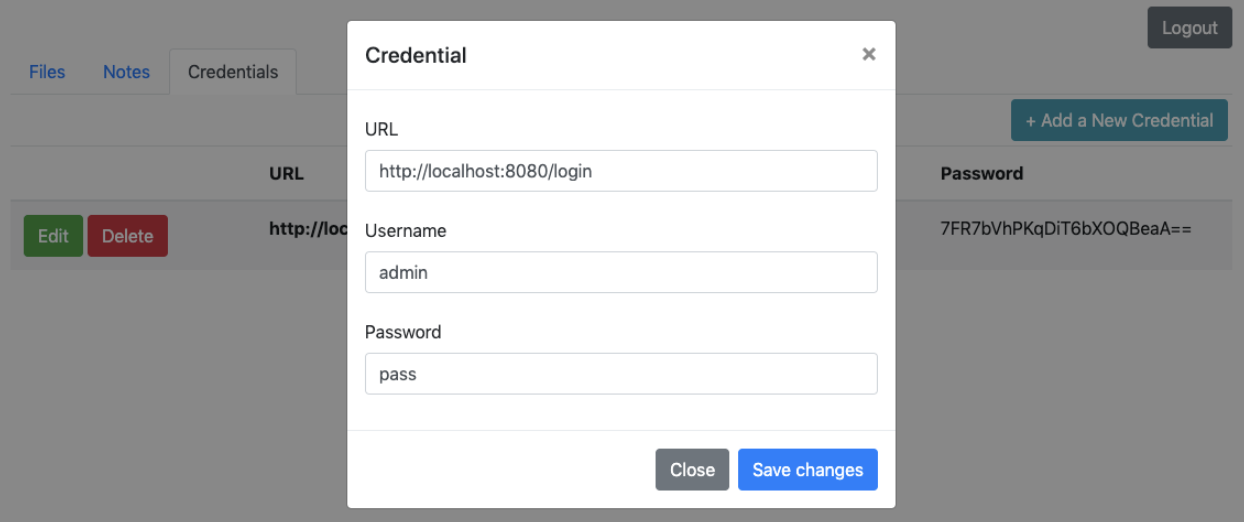


iii. Credentials:

* 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! (done)



* 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. (done)



### 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. (done)
* 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. (done)

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

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

#### 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. (done)

## 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](https://en.wikipedia.org/wiki/Hash_function)
* [Encryption](https://en.wikipedia.org/wiki/Encryption)

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

**PROJECT SPECIFICATION**

**SuperDuperDrive**

Basic Functionality

|  |  |
| --- | --- |
| CRITERIA | MEETS SPECIFICATIONS |
| Utilize Spring Boot annotations and their functions | There are Spring Boot annotations like @Controller, @RestController, @RequestBody, @RequestParams, etc. in the Java classes. |
| Utilize Thymeleaf standard dialects in the application | There are Thymeleaf attributes in the HTMl files like **th:action**, etc. |
| Integrate MyBatis into the application | There are annotations like @Mapper, @Select, @Insert, @Update, and @Delete 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 | The signup page already has input fields for all the data you need from the user, including username and password fields.  Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission. |
| Create a user signup workflow | On a successful signup, the user should be taken to the login page with a message indicating their registration was successful. Otherwise, an error message should be shown on the sign-up page.  An error message is already present in the template, but should only be visible if an error occurred during signup. |
| Develop a login page | The login page already has the username and password fields.  Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission. |
| Create a user login/logout workflow | On a successful login, the user should be taken to their home page.  An error message is already present in the template, but should only be visible if an error occurred during signup.  On logout, the user should no longer have access to the home page. |
| Create a home page | The home page should have three tabs:   1. The user should be able to upload new files on this tab and download/remove existing files 2. The user should be able to add new notes and edit/remove existing ones 3. The user should be able to add new credentials, view existing credentials unencrypted and remove them as well   The home template already has the forms required by this functionality. Add the proper Thymeleaf attributes to bind the form data to the model and send it to the back-end on submission  Details on individual features are documented in Section 3. |

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 | Creation: On successful note creation, the user should be shown a success message and the created note should appear in the list.  Deletion: On successful note deletion, the user should be shown a success message and the deleted note should disappear from the list.  Edit/Update: When a user selects edit, they should be shown a view with the note's current title and text. On successful note update, the user should be shown a success message and the updated note should appear from the list.  Errors: Users should be notified of errors if they occur. |
| Implement file storage, download, and removal | Upload: On successful file upload, the user should be shown a success message and the uploaded file should appear in the list.  Deletion: On successful file deletion, the user should be shown a success message and the deleted file should disappear from the list.  Download: On successful file download, the file should download to the user's system.  Errors: Users should be notified of errors if they occur. |
| Implement secure credential storage | Creation: On successful credential creation, the user should be shown a success message and the created credential should appear in the list.  Edit/Update: When a user selects update, they should be shown a view with the unencrypted credentials. When they select save, the list should be updated with the edited credential details.  Deletion: On successful credential deletion, the user should be shown a success message and the deleted credential should disappear from the list.  Errors: Users should be notified of errors if they occur. |

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 | A user can’t access the home page or the three tabs on that page without logging in first. The login and signup page should be visible to all the users without any authentication.  If someone isn't logged in, they must be redirected to the login page. |
| A user can 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. |
| The credentials are kept encrypted in the database | All the passwords should be stored as encrypted in the database and shown as encrypted when the user retrieves them.  The user should only see the decrypted version when they want to edit it. |
| Implement an ORM model that maps to the database using MyBatis | Create Java classes to model the tables in the database (specified in src/main/resources/schema.sql) and create @Mapper annotated interfaces to serve as Spring components in your application.  You should have one model class and one mapper class per database table. |

Testing

|  |  |
| --- | --- |
| CRITERIA | MEETS SPECIFICATIONS |
| Test signup and login flow | Write a Selenium test that verifies that the home page is not accessible without logging in.  Write a Selenium test that signs up a new user, logs that user in, verifies that they can access the home page, then logs out and verifies that the home page is no longer accessible. |
| Test adding, editing, and deleting notes | Write a Selenium test that logs in an existing user, creates a note and verifies that the note details are visible in the note list.  Write a Selenium test that logs in an existing user with existing notes, clicks the edit note button on an existing note, changes the note data, saves the changes, and verifies that the changes appear in the note list.  Write a Selenium test that logs in an existing user with existing notes, clicks the delete note button on an existing note, and verifies that the note no longer appears in the note list. |
| Test adding, editing and deleting credentials | Write a Selenium test that logs in an existing user, creates a credential and verifies that the credential details are visible in the credential list.  Write a Selenium test that logs in an existing user with existing credentials, clicks the edit credential button on an existing credential, changes the credential data, saves the changes, and verifies that the changes appear in the credential list.  Write a Selenium test that logs in an existing user with existing credentials, clicks the delete credential button on an existing credential, and verifies that the credential no longer appears in the credential list. |

**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!