Web-Based Evaluations

Final Report

Group Number: 3

Class CPSC 488 Section 1

Problem:

4/25/2022

Revised:

11/30/2022

Tanuj Rane [txr1029@sru.edu](mailto:txr1029@sru.edu), ranetanuj@gmail.com

Dalton Stenzel [drs1030@sru.edu](mailto:drs1030@sru.edu), daltonrstenzel@gmail.com

Logan Racer [lsr1006@sru.edu](mailto:lsr1006@sru.edu), loganracer27@gmail.com

Anthony Cinicola [ajc1033@sru.edu](mailto:ajc1033@sru.edu)

Revisors:

Duncan Lawrence [dal1017@sru.edu](mailto:dal1017@sru.edu) duncanlawrence2000@gmail.com

J Abbigail Rowe [jhr1002@sru.edu](mailto:jhr1002@sru.edu) jakerowe@protonmail.com

Zachary Freilino zlf1001@sru.edu, zfreilino@gmail.com

David Gillette dgg1001@sru.edu, davidgsgillette@gmail.com

Contents

[References 1](#_Toc120650011)

[Completion 1](#_Toc120650012)

[Contribution 1](#_Toc120650013)

[Glossary 1](#_Toc120650014)

[Problem Explanation 1](#_Toc120650015)

[Caveats/Minefields 1](#_Toc120650016)

[Code Reusability 1](#_Toc120650017)

[Testing 1](#_Toc120650018)

[Post-Mortem Analysis 1](#_Toc120650019)

[Project Accomplishments 1](#_Toc120650020)

[Project Issues & Solutions 1](#_Toc120650021)

[Secure Login and Password Encryption 1](#_Toc120650022)

[Searching and Sorting Users 1](#_Toc120650023)

[Uploading groups 1](#_Toc120650024)

[Ability to generate charts and PDF reports 1](#_Toc120650025)

[Deriving the evaluation form from an excel file 1](#_Toc120650026)

[Expansion/Improvement Goals 1](#_Toc120650027)

# References

[1] Baeldung *Get All Data From a Table With Hibernate*. <https://www.baeldung.com/hibernate-select-all>

[2] W3school. Bootstrap https://www.w3schools.com/bootstrap/bootstrap\_ref\_all\_classes.asp

[3] Baeldung *Spring Security Roles and Privileges* <https://www.baeldung.com/role-and-privilege-for-spring-security-registration>

Previous group 3

[4] <https://www.baeldung.com/spring-boot-testing>

[5] <https://spring.io/guides>

[6] <https://reflectoring.io/spring-boot-test/>

[7] <https://www.arhohuttunen.com/spring-boot-webmvctest/>

[8] <https://semaphoreci.com/community/tutorials/stubbing-and-mocking-with-mockito-2-and-junit>

[9] <https://stackoverflow.com/questions/41770156/spring-add-custom-user-details-to-spring-security-user>

[10] <https://stackoverflow.com/questions/14268451/spring-security-userdetailsservice-implementation-login-fails>

[11] <https://howtodoinjava.com/spring-security/inmemory-jdbc-userdetails-service/>

[12] <https://www.baeldung.com/spring-security-authentication-with-a-database>

[13] <https://www.baeldung.com/circular-dependencies-in-spring>

[14] <https://stackoverflow.com/questions/46297832/required-a-bean-of-type-org-springframework-security-core-userdetails-userdetai>

[15] <https://github.com/itzg/spring-security-spa/issues/1>

[16] <https://www.javainuse.com/webseries/spring-security-jwt/chap5>

[17] <https://stackoverflow.com/questions/36824973/cant-find-securitymockmvcconfigurers>

[18] <https://www.baeldung.com/spring-security-integration-tests>

[19] <https://stackabuse.com/get-http-post-body-in-spring/>

[20] <https://blog.devgenius.io/spring-boot-deep-dive-on-unit-testing-92bbdf549594>

[21] <https://spring.io/guides/gs/testing-web/>

# Completion

Project Status: Complete

**Completed**:

* Upload, create, edit, and delete groups
* Upload, preview, save, and delete evaluation forms/templates. Form ID enable multiple unique evaluation form uploads.
* Features of the evaluation file are fully implemented
  + Sections and all question types
  + Pre load information about reviewee
  + Required questions marked with asterisk, cannot submit if unanswered
  + Fully functional tool tips for questions and sections
  + Tool tip marker can be changed to three different options
  + Compute sections display scores from dropdowns
  + Questions which control visibility of another question
* Preview uploaded evaluation file before saving, question inputs disabled
* Check for errors in uploaded file before saving, display errors on page
* When an evaluation excel file is submitted with errors, the original file is available for download with a new sheet containing the errors encountered.
* Source excel file can be re-downloaded from previously uploaded evaluations.
* Evaluation forms can be deleted if they are currently assigned to no groups.
* The Groups page for EvalAdmin is inaccessible until at least one evaluation is uploaded.
* Info and errors are logged to a file which is downloadable by the Admin.
* Loading spinners on potentially lengthy actions, such as uploading excel files
* Evaluators fill out and submit completed evaluation
* Save incomplete evaluations for later
* Reviewee can view evaluations done on them
* Eval Admin can view evaluations done for all groups and members
* Access deleted group evaluations through the archive
* Complete self evaluation where applicable
* Evaluation have async and sync option
* Evaluation have preview and no preview option
* Groups can be assigned their own unique evaluation form
* After some evaluations have been submitted, the EvalAdmin can:
  + Generate individual charts (Pie charts, Bar charts, Ring charts, Area charts)
  + Generate average individual score charts
  + Generate average group score charts
  + Generate and download PDF reports for entire group
  + Generate and download analysis excel documents containing averages and totals of all completed evaluations for a given evaluation form.
  + Generate and download analysis excel documents containing averages and totals of all completed evaluations for a given group.
* Active page stays highlighted in the navbar
* Default password reset for first time logins
* Reset password option at login screen
* Filtering and sorting on Admin users page
* Consistent web-page design layout
* Ability to upload additional files (PDF) when submitting and evaluations
* Analysis excel file should use Apache POI instead of Aspose Cells
* Added a company structure
* Added Role objects for access control
* Implemented the roles into the existing program
* Added privileges to the roles to have granular control
* Added departments/locations/location/groups to company
* Implemented the display of departments in various locations of the program
* Implemented way to add companies/company structures/roles
* Updated evaluations to take advantage of departments
* Created a split between companies for the program so multiple companies can exist at the same time.
* Updated GUI to reflect the presence of the company
* Modified user upload/group upload/template upload to all use post requests.
* Modified the user/group/eval template uploads to reflect the company changes and updated the templates themselves.
  + User now uses some if the info that was previously just strings to add it to certain objects. Company dept etc.
    - Also can be assigned as a department head from the upload template.
  + Group template can now include departments as “members” of the group.
    - Also updated the group object to hold a collection of the the departments that are present in the group. I.E. a list of all the different departments the members in that group belong to.
  + Eval template now is added to one company and the second sheet is used to assign it to specific departments within that company (it is added to the company of the user adding the template).
* Implemented automatic schema creation.
* Implemented automated adding of admin user and initial company.
* Unit testing
  + Number of tests from previous group total <10
* Integration testing
* Overhauled old logging system to use log4j
* Added explicit error logging

**Incomplete**:

# Contribution

Spring security login/websecurity: 65% from Daily Code Buffer, Java Brains, & Sharma Manish

Front-end to back-end communication: 40% from Thymeleaf & Eugen Paraschiv

Searching/retrieving data from database: 45% from Baeldung, Oliver Gierke, & Thanh Tran

CSS-Bootstrap for web page design: 90% from Mark Otto

Displaying charts on webpage: 80% from ZetCode

General chart generation: 40% from TutorialsPoint

Pdf report generation: 65% from ThinkTibits

Testing Concepts: 90% from [20, 8, 7, 6, 5, 4]

MockMvc: 100% from [21, 17, 7]

Integration Tests: 90% from [21, 20, 19, 18, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 4]

Copy-pasted code in integration tests: 100% from [6]

**Note: All testing was taught at end of year, so testing was self-taught, and all resources listed were used in some capacity, even if not explicitly in code.**

# Glossary

ADMIN - Administrative user

EVAL\_ADMIN - Evaluation Administrator user

EVALUATOR\_EVAL - Evaluator and evaluee/reviewee

EVALUATOR - Evaluator user

USER - Reviewee user

Evaluation File - Excel file which contains instructions for the program to assemble an Evaluation Form

Evaluation Template - Evaluation form containing blank responses from which evaluations are conducted.

pieDataset - Dataset to be used for pie chart and ring chart generation

categoryDataset - Dataset to be used for bar chart and area chart generation

EvalGroup - Group

eval-evaluationLog

rev-reviewee

role-EvalRole

id-Identification number

sort- A particular arrangement of values based on an arrangement term such as by first name, or email.

sortOr -The order in which the arrangement of values is placed in such as ascending or descending.

# Problem Explanation

The issue at hand is the need for a system that provides the ability to view and perform evaluations on staff or employees at a company or organization. The requirements include the ability to customize the evaluation criteria and include qualitative and quantitative questions. Evaluations must be appropriately assigned to the correct people in particular companies/schools/etc and must include the option to have several evaluators able to perform evaluations on the same person or group. Administrators with the particular power over evaluations must provide the evaluation forms, groups, and order of evaluations. The ability to view completed evaluations must be present depending on determining factors such as rank. User accounts must be present and will be ranked to have some sort of order and power distribution. Information from the program, such as user data and finished evaluation reports, must be held from within a database. It must also be able to generate analysis based on the completed evaluations, including things such as chart generation, PDF reports, and score analysis spreadsheets.

# Caveats/Minefields

When either adding, uploading, or editing a user’s information, there is some inconsistency and loose requirements for what may be entered. The only serious checks are for spaces and null values with most requirements and a length requirement of 5 characters for passwords on adding/editing. Attributes such as names do not check/reject special characters and the Date of Hire attribute accepts strings of any kind. A user can have almost the entirety of their information be a single digit, other than the password and role, as those entered digits are converted to string and aren’t violating the space or null checks in place.

A Caveat regarding the evaluation analysis excel file is also worth mentioning. Upon downloading the analysis file and opening it, the user will be greeted by a blank excel sheet with an Aspose Cells watermark in blue text. When building this aspect of the system, we opted to use an API called Aspose Cells to handle the creation and formatting of new excel documents. Unfortunately, this API is not free of charge and this project has been implemented using the free version. This results in the watermark being added to every output file. When opening a newly generated analysis file for the first time, simply select sheet 1 to view the actual contents of the analysis spreadsheet. This caveat is also referenced in the future work section, which states that an alternative free API which offers similar features (Apache POI) could and should be implemented in its place.

Some Caveats/Minefields that pertain to chart generation is that sections that are not given a score are given a default score of zero, when viewing the charts it may look like sections are missing however these sections are present and their values are just zero. When clicking the chart generation button on a group which does not have any evaluations or self evaluations done there will not be chart generation buttons, once an evaluation is completed these buttons will appear.

# Code Reusability

The software was built with the idea of OOP programming in mind, so the use of code built for reusability is certain. Not all aspects of every class and method provide the most elegant solution for the subject of recycling but there are plenty of instances.

The services package includes classes that contain methods that other classes need in order to properly out what is needed. The UserService is the source for pulling custom user objects from the custom UserRepository class. The User Service class has instances created and used by classes such as the UserController and AddUserController class for obtaining what uses to be displayed and in what format. The UserService class can easily be reused in a situation where there is a different User and UserRepository class as its sole duty is to provide a list of objects from a database.

The chart generation and pdf report generation were built with reusability in mind. The chart generation methods use a method to find the logs that match the id received through the path variable. These logs are then sent to the createDataset class where the section names and information are pulled and placed into a usable dataset. These values are pulled using methods in the evaluation class meaning that no matter the amount of scoring and non-scoring sections the correct values will be pulled. This means that any evaluation template can have their scores converted into a usable dataset. The chart generation page and buttons are displayed based on the group, reviewee, and evaluationLog tables in the database. This allows the list of reviewees on the chart generation page to be dynamic and update when users are added or deleted.

The Evaluation class and related classes lend themselves to reusability. All actions relating to creating and formatting evaluations are encapsulated by the Evaluation class and its subclasses (Section, Question, and ComputeRange). This includes necessary functions such as updateCompute(), processTooltips(), saveResponses(), getSectionByName(), and getQuestionById(). Setters, getters, and List processing methods are also included. Any other project of similar scope could implement these classes into their project with little modification needed. Despite this, there are several utility classes for our specific use case which were delegated to separated classes with implementation specific methods. This includes the ParseEvaluation class which loads the Evaluation object with data from a XML file, and the GenerateEvalReport class which takes evaluations from the database and produces an analysis file.

# Testing

Any program of significant size will need to be tested. This is a point that our team prides itself on. These tests are mostly one of two types: unit and integration. Unit tests test any code that is not dependent on other classes or simulates the dependency using mocks provided by Mockito. Integration tests are tests that test multiple classes and ensure those classes work as a group and do not use mocks. In total, there are approximately 450 tests with about 400 of them being unit tests.

Tests outside of coding, like black box tests, were executed either manually or using Burp. While Burp allows you to automate your black box testing, the actual wordlists used to run against the application must be made manually. A sample word list would include a list of common usernames and passwords and each list would be passed into their respective field in Burp to simulate the attack.

# Post-Mortem Analysis

Every project made by anyone is likely to be subject to either convoluted solutions or patch-up jobs in order to function and this project is no exception. Below are a few examples of which less-elegant solutions were provided.

In the AdminMethodsService class, there are two very similar methods that are depended on heavily from particular methods from within the UserController and AddUserController. The two methods provide a series of checks on particular attributes of a user in order to save either a new user or to changes made to an existing user. The summation of the lines of code used by both methods makes up approximately two-thirds of the class’s total code count. While combining both methods into a singular could prove to be a little difficult, removing the redundancy by creating a third method seems easier at first glance.

Another particularly bizarre decision made in the AdminMethodsService class comes from the sortCheck method present. The method is quite redundant as it checks what type of sorting is being asked of it and then forwards the information to an instance of UserService. The issue is apparent upon staring at the sorting method inside the UserService; the sorting method also checks what type of sorting is being requested. The change would be relatively simple but what is done, is done.

The groupUpload method definitely has room for improvement . As we worked on the project we ended up adding more fields in the group upload file and because of that we ended up just adding changes on top of the upload method resulting in the upload method having high time complexity. if we had a better idea of what the group file was supposed to look like in the beginning the group upload method would have been implemented differently.

Looking back on the chart generation and PDF reports there are a few things we would do differently. First, we would have used the setChartTheme method in the ChartFactory class more often. The charts generated are using the default theme, using the setChartTheme method the theme of the charts can be changed to be more visually appealing. The second thing we would do differently pertains to the PDF report generation. We decided to use iText which is a free java PDF library which allows the user to create, convert, and manipulate PDF documents. If I were to go back and redo the project I would opt to use a different library. I would most likely choose to use Jasper Reports. This is a java reporting tool which generates reports in PDF documents. Jasper Reports allows for more versatility and functionality when creating reports.

While the core of the Evaluation object and its subclasses were well designed, there is certainly room for improvement. For example, all section types are represented by the same Section class and all question types are represented by the same Question class. This means that all methods and variables are present in all classes of the same type, even if they are not necessary. For example, the methods pertaining to score computation are only necessary in sections which contain compute questions or dropdown questions. In a perfect implementation, these methods would only be available to the sections and questions that can make use of them. Using principles of inheritance classes, abstract classes, and design patterns, the Evaluation object could be designed more elegantly and logically in terms of proper Object-Oriented programming.

# Project Accomplishments

* Secure login
* Password recovery
* User roles for different tasks/objectives
* Uploading evaluation forms, user information files, group information files
* Ability to manually create users
* User searching/sorting
* Ability for group & user editing/deleting/archiving
* Ability to perform evaluations
* Ability to generate charts, PDF reports, and analysis spreadsheets

# Project Issues & Solutions

## Secure Login and Password Encryption

The need for a logging in was an ability, so that not just anyone could access the contents of the web application, was required and served to be one of the first milestones to accomplish. From the use of online resources and tutorials, a concrete solution was forged from the use of a Spring dependency known as **Spring Security**. To solve the password encryption problem, Spring Security allows for custom password encoders to be assigned to the authentication provider so BCrypt was selected from the ones provided by Spring Security as well. This dependency solved not only the login and encryption requirements, but also provided a way for protecting any webpage produced from non-authorized users who don’t have the required role to access them.

## Searching and Sorting Users

The use of searching and sorting users was far too important to ignore for this project when the possibility of dealing with many users was quite possible. The search for an easily digestible and integratable solution for the missing feature became quite the problem as the methods we discovered provided nothing but frustration so we developed our own simple way of gathering and organizing users. By adding on the existing UserService class that grabs the list, we produced the ability to sort by taking advantage of the built in Collection sort object and method mixed with our User attributes for comparison. By checking for specific strings and values, we made the ability to sort by alphabetical, numerical and ascending/descending order.

## Uploading groups

The ability to upload groups was needed to create groups with the proper specification (roles, form, reviewee evaluators, sync evaluation, and preview). After creating groups, evaluators may start their evaluations. The issue that came with the group upload was determining how the evaluation order was going to be set up because each group is different and has its own evaluation order. To solve this for each evaluator in the group, we gave them a role and each role has an id that determines the order the evaluations need to be done in. Another issue that uploading groups came with was determining if the evaluation can be done in async or sync (which means if multiple evaluations can be done at the same time or if they have to wait their turn). The solution provided to the issue was to provide evaluators with a new boolean field named “sync”. If sync is true, the evaluation after the evaluator has to wait their turn. If sync is false then the evaluation after can start at the same time. (For example, if level 1 sync is true then level 2 has to wait for level 1 to finish but if level 1 sync is false then level 2 can also start their evaluation.

## Ability to generate charts and PDF reports

The ability to generate various charts and PDF reports from the evaluation scores was needed to give the evaluation admin the capability to read the results of the evaluations in a clear, concise way. The evaluation admin can generate four different charts, pie charts, bar charts, area charts, and ring charts. These four charts are available for each level of evaluation for each user. These charts can also be generated using the average score of every level of evaluation, and can also be created using the average scores for the entire group. The evaluation admin can also create charts using data pulled from the reviewee’s self evaluations. The evaluation admin also has the option to create a PDF report for the group. This PDF report will allow the viewer to compare all the chart types side by side as well as give the average group scores for each section in a table format.

## Deriving the evaluation form from an excel file

Uploading an evaluation form as an excel file was one of the requirements of this project. This proved to be an implementation challenge because of the dynamic and unique nature of the evaluations. This required all components of an evaluation to work in unison including the excel file layout, reading the data, writing the data to the Evaluation object, rendering the content on the page, and saving user entered responses.

In order to read information from an excel file, the solution involved first converting the excel file to XML and parsing it to extract the data. Specific keywords are used in the excel file which are detected by the XML parser. Rather than reading specific cells, this implementation essentially receives data sequentially and uses the predefined keywords to obtain the appropriate data. This is an approach which, while basic, worked reliably in our test cases. This has the added benefit of allowing more freedom in the positioning of elements in the document without causing errors.

Displaying the evaluation form through HTML was another implementation challenge. The evaluation form could not be statically defined with traditional HTML tags and attributes. Rather than displaying the evaluation data on the page directly from the XML input file, we opted to store all data in an Evaluation object first. Having the data in this structure allowed us to use a composition of Lists having dynamic lengths for sections, questions and responses. These could then be iterated through by Thymeleaf to produce HTML. The Evaluation object also made saving responses and serializing evaluations more accessible.

# Expansion/Improvement Goals

* ADMIN\_LOG role & log protection
* Changing the group upload file so it's easier to read and understand.
* Reduce code redundancy:
  + AdminMethodsService: Two methods, checkAndUpdate() and comparingMethod() share similarities but return different things so an overhaul would be in order to improve code reusability.
  + PDF Chart generation and Excel Analysis generation both require averaging evaluation scores but have unique methods. Methods should be re-written so they are usable for both tasks.
* Add more chart generation types
* Add individual PDF report generation
* Ability for group to have their own role set
* Add ability for EvalAdmin to develop an evaluation template through a page on the site. This would alleviate many potential sources for error which currently exist when drafting the evaluation excel file and would greatly increase usability.
* Ability to download an auxiliary file for group members which shows a summary of who evaluated them and their level
* Save Group number opened when transitioning pages on the Groups & Evaluate pages for the Eval\_admin & Evaluator roles respectively
* Add ability to download evaluations (including self evaluations) as pdf individually & all of them at once
  + Also better way of downloading the attached files as well
* Improve naming convention of Sync/Async evaluation