Web-Based Evaluations

Testing Document

Group Number: 3

Class CPSC 488 Section 1

4/25/2022

10/31/2022

Tanuj Rane txr1029@sru.edu

Dalton Stenzel drs1030@sru.edu

Logan Racer lsr1006@sru.edu

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

Revisors:

J Abbigail Rowe

Contents

[Section 1) Overview of Testing 3](#_Toc118127286)

[Section 2) Unit Testing 3](#_Toc118127287)

[2.1) AdminMethodService 3](#_Toc118127288)

[2.2) Group Testing 3](#_Toc118127289)

[2.3) Evaluation Testing 3](#_Toc118127290)

[2.4) 4](#_Toc118127291)

[2.5) 4](#_Toc118127292)

[2.6) 4](#_Toc118127293)

[Section 3) Integration Testing 4](#_Toc118127294)

[3.1) Integration Testing Troubles 4](#_Toc118127295)

[Section 4) Black Box Testing Examples 5](#_Toc118127296)

[4.1) Logging In 5](#_Toc118127297)

[4.2) Additions/Deletions/Edits 5](#_Toc118127298)

[4.3) Searching/Sorting 5](#_Toc118127299)

[4.4) More Tests 6](#_Toc118127300)

[Section 5) White Box Testing Example 6](#_Toc118127301)

[5.1) PageCalc() Method 6](#_Toc118127302)

[5.2) File Upload Handling 6](#_Toc118127303)

# Section 1) Overview of Testing

The document within this file is tasked with covering the procedures used for testing the Web-Based Evaluations System. The testing types included are:

* Unit Testing
* Integration Testing
* Black Box Testing
* White Box Testing

# Section 2) Unit Testing

## 2.1) AdminMethodService

* capitalize() - Test to see whether returned String has been capitalized or not
* hasSpace() - Test to determine if String sent has a space located at any point in the String
* checkAndUpdate() - Test to determine the different conditions of the method are met. The conditions being where each attribute of a user is appropriate to be saved. Attribute checking would include searching for null, spaces, empty, and length requirements

## 2.2) Group Testing

* tested the creation of group with reviewee and evaluators
* tested group,reviewee and evaluators repository(retrieving data based on certain perimeter like evaluator group number ).

## 2.3) Evaluation Testing

* Each test is first initialized by creating a new Evaluation object and populating it with testing data. This is the initialize() method annotated with the ‘@BeforeEach’ tag.
* testEvalSectionCount() - Ensure that the number of Sections returned by the Evaluation.getSectionCount() method is correct.
* testEvalQuestionCount() - Ensures that the Section.getQuestionCount() method returns the correct number of questions which are present.
* testSectionHasDropdown() - Checks validity of the Section.hasDropDownQuestions() method which returns a boolean.
* testComputeRanges() - Validates the logic of the Question.computeResultsString() method. Based on the assigned ComputeRanges, number values are passed in to ensure that the correct range name string is returned. Values which fall outside the defined ranges should return “NO SCORE AVAILABLE”.
* testDropdownOptionsMaxPoints() - Ensures that the string values of the response options are correctly parsed to integers. Also checks the logic of the Question.getQuestionMaxPoints() method which returns the largest point value of the response options.
* testDropdownOptionsEarnedPoints() - Based on the string value for Question.qResponse, Question.getEarnedPoints() should return the value of the string converted to an integer. N/A or blank responses should return 0.
* testSectionScoringWithDropdownsQResponse3() - Checks the correct functionality of Evaluation.updateCompute(). After setting question responses as 3 and updating the compute information, the correct compute string should be returned.
* testSectionScoringWithDropdownsQResponseNA() - Checks the correct functionality of Evaluation.updateCompute(). After setting question responses as “N/A” and updating the compute information, the correct compute string should be returned.
* testSectionScoringWithDropdownsValueInvalidQResponse() - Checks the correct functionality of Evaluation.updateCompute(). After setting question responses to an invalid value and updating the compute information, the correct compute string should be returned.
* testSectionTooltipProcessing() - Verifies that the Evaluation.processToolTips() method functions correctly. The provided Section and Question tooltip information should be applied to the Question text and Section description respectively.

## 2.4) Domain Testing

### Archive Test

* getIdTest()
* getDateEditedTest()
* getPathTest()
* getRevieweeTest()
* getRoleTest()
* getEvaluatorTest()
* nullValuesTest()

### EvalRole Test

* getIdTest()
* getNameTest()
* getEvaluatorTest()

### EvalTemplates Test

* getNameTest()
* getEvalTest()
* getExcelFileTest()

### EvaluationLog Test

* getEvaluatorTest()
* getRevieweeTest()
* getPathTest()
* getId()
* getDateEdited()
* getAuthTest()
* getCompletedTest()
* getAttachTest()
* getAttachnameTest()

### EvaluatorId Test

* getGroupIdTest()
* getLevelTest()

### Evaluator Test

* getIdTest()
* getUserTest()
* getGroupTest()
* getEvalutationLogTest()
* appendEvalutationLogTest()
* getLevelTest()
* isSyncTest()
* isPreviewTest()

### Group Test

* getIdTest()
* getRevieweeTest()
* getEvaluatorTest()
* getEvalTemplatesTest()
* getEvalstartTest()
* getSelfevalTest()

### MyUserDetails Test

* getAuthoritiesTest()
* getPassword()
* getIDTest()
* getRolesTest()
* getUsernameTest()
* isAccountNonExpiredTest()
* isAccountNonLockedTest()
* isCredentialsNonExpiredTest()
* isEnabledTest()

### PasswordResetToken Test

* getTokenTest()
* getUserTest()
* getUserIdResetTest()
* getExpireTimeTest()
* getExpiredDateTest()

### ResetPassword Test

* getEmailTest()
* getPasswordTest()
* getPasswordCheckTest()

### Reviewee Test

* getIdTest()
* getGroupTest()
* getNameTest()
* getUserTest()
* getEvalutationLogTest()
* getSelfEvaluationTest()

### SelfEvaluation Test

* getRevieweeTest()
* getPathTest()
* getIdTest()
* getDateEdited()
* getCompleted()

### User Test

* getPasswordTest()
* getIdTest()
* getNameTest()
* getEmailTest()
* getRolesTest()
* getFirstNameTest()
* getLastNameTest()
* getSuffixNameTest()
* getResetTest()
* isResetPTest()
* getDateOfHireTest()
* getJobTitleTest()
* getSupervisorTest()
* getCompanyNameTest()
* getDivisionBranchTest()

## 2.5) Service Testing

## MyUserDetailsServiceTest.loadUserByUsernameTests()

* tested the creation of group with reviewee and evaluators
* tested group,reviewee and evaluators repository(retrieving data based on certain perimeter like evaluator group number ).

## 2.6) Excel Testing

newSheet()

loadFileTest()

checkStringTest()

checkIntTest()

checkLongTest()

# Section 3) Integration Testing

## 3.1) Integration Testing Troubles

While unit testing was achievable in a rather simple manner, the act of automating a checking system for the relationships between classes proved to be difficult when trying to apply such a technique part way through development. With the way our program is structured and connected provides seemingly endless errors upon trying to call a class that calls multiple others due to the plethora of instances, objects, and services included in the project. An example being when trying to test the AddUserController with a proper link to our user repository/database and the requirements for an instance provided errors with other instances being called from inside the AddUserController. More time could have provided us the necessary knowledge and abilities to overcome this challenge, but for now, we’ve been bested.

The previous group built their program in such a way that integration testing is not possible without reworking a majority of the project. The controller classes must be rewritten to have @PostMapping and @GetMapping methods, and the classes should be annotated @RequestMapping. Additionally, something is broken in the Beans for the project causing a need for a cyclic call in construction of beans when the SecurityConfiguration is called. This causes errors in loading the ApplicationContext and forces the program to terminate. Note that the implementation of the Beans and controllers will affect the code in other classes and will therefore require additional work on the existing code not mentioned to be reworked/refactored to meet the new standards. This will not affect every class but will affect any who utilize the functions the Beans and controllers provide.

The main packages/classes to be focused on are as follows:

* Entire controller package
* Configuration/SecurityConfiguration
* Repository
* Service
* Domain potentially
* Excel potentially

# Section 4) Black Box Testing Examples

## 4.1) Logging In

A user’s credentials and security are a primary concern of the program. Users need to be able to access their own accounts without having troubles such as being redirected to a wrong page. Users of various roles were logged into their accounts both at different and same times in order to see what would happen. It is expected that users would not have troubles accessing their own respected accounts and webpages. Logging into different users at the same time protects from users having their logins merge or swap as long as the test occurs in different browsers or instances of the browser. Having two or more instances of different users logged in on the same browser will merge the cookies, and thus the users will get mixed up, but such behavior is to be expected. Users with different instances of browser cookies, which would be the normal situation as multiple people using one laptop to browse the internet isn’t common, will work as it should.

## 4.2) Additions/Deletions/Edits

Any sort of tinkering with the database entries were able to be tested with ease with the ability to see our data being present to us on the respective webpages. Any changes to users can be seen by viewing the edit buttons or by seeing the default admin users page’s table of users. Feedback is provided on all pages that have actions that provide changes to the database such as notifying if a user was deleted or if the changes attempted failed to be completed.

## 4.3) Searching/Sorting

Users with the “ADMIN” role can search and sort through users on the admin user page via a search bar and a couple of drop-down buttons as well as just normal buttons as well right about the user table. Changing the options of the search should result in the expected changes, and rightly do so.

## 4.4) More Tests

Black Box testing is only to be done once the most recent version is available and working (key word working).

# Section 5) White Box Testing Example

## 5.1) PageCalc() Method

The pageCalc() method located in the AdminMethodsService class is one that could have easily left room for problems if left unsupervised. The method relies on calculations relating to modulus and division which can easily lead to “division by zero” errors. Through analysis and trial and error lead to a seemingly robust defense against any input and regardless of the size of the file or list used. One situation the method takes into consideration is when the user is loading the final page, but there aren’t enough users to fill the list. Another problem taken care of is when the current page is the last page of users and the users per page is set to a low value such as 10; when the user changes the sort/search to a significantly higher value of users per page, the user will be booted to the first page as the previous page would be out of the range of pages available.

## 5.2) File Upload Handling

The ability to upload files arrives on a few occasions, such as on the admin users page, or on the evaluator-admin group or evaluation form page for uploading user, group, or evaluation form information files to the database. Those files must be called by their respective upload methods from each controller. Those methods each are designed in a way to either check whether a file has been uploaded, whether the file contains the right tag/information, and whether some (or any) of the information is even valid. This is done with a series of checks and try/catch exceptions.