CourseSpotter

Validation Test Plan and Results

**Contents**

**Part 1 Unit Testing**

* 1. **Methodology (Unit)**
  2. **Results**

**Part 2 Component Testing**

**2.1 Methodology (Component)**

**2.2 Results**

**Part 3 System Testing**

**3.1 Methodology (System)**

**3.2 Results**

**1.1 Methodology**

Unit, component, and system testing were carried out to ensure that the software met all requirements according to the requirements specification and that it could perform correctly as well as reliably. The JUnit unit testing framework for Java was employed for the purpose of carrying out these tests. All of the tests written are located within the “src/test” directory and all of the test functions have a suffix of “Test” along with the test type (A or B) added to the end, so that they can be easily identified.

**1.2 Results**

* + 1. **Parser**

**Unit Test #1:**

|  |  |
| --- | --- |
| Method Name | acquireSessionInfo |
| Class Name | Parser |
| Test Type | A |
| Parameters | None |
| Results | The function successfully returns a linked hash map containing the appropriate session values, including the ICSID & ICStateNum. |

**Unit Test #2:**

|  |  |
| --- | --- |
| Method Name | acquireSessionInfo |
| Class Name | Parser |
| Test Type | B |
| Parameters | None |
| Results | System is able to avoid an exception when the linked hash map for session values is empty. If the session info is not successfully returned, then the other functions will not be able to get their desired output, so in the chance of empty session key, the fetchSubjects and others do not send requests. By doing this we can avoid wasting resources and a system failure since the resulting response by the external system (CUNYfirst) would have been erroneous. |

**Unit Test #3:**

|  |  |
| --- | --- |
| Method Name | fetchSubjects |
| Class Name | Parser |
| Test Type | A |
| Parameters | String icsID, int icstateNum, String institution, String term |
| Results | The function outputs a list of subjects, represented by strings for the appropriate college and term.  List Elements (first 9):  AACS - Asian Amer Commun. Std  ACCT - Accounting  AFST - Africana Studies  AMST - American Studies  ANTH - Anthropology  ARAB - Arabic  ARTH - Art History  ARTS - Studio Art  ASTR - Astronomy  …. |

**Unit Test #4:**

|  |  |
| --- | --- |
| Method Name | fetchSubjects |
| Class Name | Parser |
| Test Type | B |
| Parameters | String icsID, int icstateNum, String institution, String term |
| Results | System successfully remains in operation in the event of an unexpected situation, such as when there is an empty subjects list, fetchByClassNumber and fetchCourseInfo will not attempt to send requests. Our system responds this way because the CUNY system will issue an error if it is in the wrong state. So, if we cannot obtain the subjects then we won’t proceed to the next step. |

**Unit Test #5:**

|  |  |
| --- | --- |
| Method Name | fetchCourseInfo |
| Class Name | Parser |
| Test Type | A |
| Parameters | String icsID, int icstateNum, String institution, String term, String subject, int course |
| Results | The function successfully outputs a complete list of courses based on the inputted college, term, subject, and course number. Each object in the list is a CourseInfo object. |

**Unit Test #6:**

|  |  |
| --- | --- |
| Method Name | fetchCourseInfo |
| Class Name | Parser |
| Test Type | B |
| Parameters | String icsID, int icstateNum, String institution, String term, String subject, int course |
| Results | The system is successfully able to stay in operation even though the course info list is empty. This is because there are safeguards in place to check if the list is empty first before trying to present the items within. |

**Unit Test #7:**

|  |  |
| --- | --- |
| Method Name | fetchByClassNumber |
| Class Name | Parser |
| Test Type | A |
| Parameters | String institution, String term, String classNumber |
| Results | The method successfully returns a single CourseInfo object which contains all of the course information for that course retrieved via class number (after having inputted the institution and term). |

**Unit Test #8:**

|  |  |
| --- | --- |
| Method Name | fetchByClassNumber |
| Class Name | Parser |
| Test Type | B |
| Parameters | String institution, String term, String classNumber |
| Results | If an unexpected scenario occurs due to fetchByClassNumber not being able to retrieve the correct result due, for instance, due to it not being able to retrieve the subjects, then the system returns an empty object and doesn’t attempt to send the request over to CUNYfirst. |

**1.2.2 Message Service**

**Unit Test #9:**

|  |  |
| --- | --- |
| Method Name | sendEmail |
| Class Name | MessageService |
| Test Type | A |
| Parameters | String from, String subject, String to, Content content |
| Results | The method works successfully as we receive a return code of 202, which means the following ACCEPTED - Your message is both valid, and queued to be delivered. |

**Unit Test #10:**

|  |  |
| --- | --- |
| Method Name | sendEmail |
| Class Name | MessageService |
| Test Type | B |
| Parameters | String from, String subject, String to, Content content |
| Results | The system is able to successfully stay in operation during the case of a problem. The return code of this unsuccessful attempt is 0. |

**Unit Test #11:**

|  |  |
| --- | --- |
| Method Name | sendText |
| Class Name | MessageService |
| Test Type | A |
| Parameters | String to, String from, String body |
| Results | The system successfully makes the request using Twilio’s API. The Message SID is a valid value, so everything is working on our end.  Outputted value: SM30fa7efb999d4960b917744385a19105 |

**Unit Test #12:**

|  |  |
| --- | --- |
| Method Name | sendText |
| Class Name | MessageService |
| Test Type | B |
| Parameters | String to, String from, String body |
| Results | The test is successful in an unexpected situation as we receive an empty SID when we tried to send a text from and to the same number. |

**1.2.3 DB Worker**

**Unit Test #13:**

|  |  |
| --- | --- |
| Method Name | insertData |
| Class Name | DBWorker |
| Test Type | A |
| Parameters | String email, String phone, String college, String courseName, int courseNumber, int classID |
| Results | The system successfully inserts the data into the database and receives a return code of 1.  Values Entered:  test@example.com 1112223333 QNS01 CSCI 211 50460 |

**Unit Test #14:**

|  |  |
| --- | --- |
| Method Name | insertData |
| Class Name | DBWorker |
| Test Type | B |
| Parameters | String email, String phone, String college, String courseName, int courseNumber, int classID |
| Results | The system successfully operates when the row was not able to be inserted into the database. The return code for this process is 0. |

**2.1 Methodology (Component)**

For the component testing we look at some of the different interactions in the CourseSpotter system. Here we take some of the various subsystems that interact with each other and verify that the behavior is what we want.

**2.2 Results**

|  |  |
| --- | --- |
| Interaction Name | MainView -> Parser |
| Test Type | A |
| Description | In this scenario, we looked at whether the view successfully makes use of the Parser when retrieving details, it needs like the course subjects for example, which have to be received from CUNYfirst. |
| Results | The view is able to successfully use the methods of Parser in order to obtain the CUNYfirst data which is parsed and passed on to the view in a timely manner. The output is the list of subjects as well as the course results retrieved later on. |

|  |  |
| --- | --- |
| Interaction Name | MainView -> Parser |
| Test Type | B |
| Description | In this scenario, we looked at whether the view utilizes the Parser but is unable to retrieve the necessary information from CUNYfirst. |
| Results | The view is not able to retrieve the info from CUNYfirst while using the methods of Parser, although the application still remains operational. This results in there being an empty select box on the main view. |

|  |  |
| --- | --- |
| Interaction Name | DBWorker -> MessageService |
| Test Type | A |
| Description | We tested whether the DBWorker is able to successfully call the MessageService to send the actual email and text message to a user. |
| Results | The DB Worker is able to successfully invoke the methods of the MessageService in order to submit the appropriate email and text message. The output is the email message and text message sent to the CourseSpotter user. |

|  |  |
| --- | --- |
| Interaction Name | DBWorker -> MessageService |
| Test Type | B |
| Description | We tested a scenario in which the DB Worker did call the MessageService, but the information given was not valid |
| Results | The MessageService does attempt to submit the information via the SendGrid and Twilio APIs but this is not sufficient. The email and text message request is rejected by their respective API’s and the message does not make it to the user. |

**3.1 Methodology (System)**

For the system testing phase we verified that each of the functional requirements of the Business Requirements document was fulfilled. This was done by hosting a live version of the system which is accessed through a web browser and tested by various users to find out whether the system as a whole behaves as expected. In the results, we will list each functional requirement and the appropriate test result.

* 1. **Results**

**Functional Requirement #1**: Form Displayed with multiple input fields including College, Subject, Course Number, Phone, and Email as well as a submit button

This requirement was fully met since we have the appropriate fields displayed on the screen. For instance, we have a select box for the college that allows the user to choose the appropriate college. There is a select box for the subject as well in addition to text fields for the user to input the course number and phone number. The email address uses a custom email field which will help with email validation.

**Functional Requirement #2**: System catches errors in the input fields (left blank, option never selected, invalid input) and alerts the user.

The system is able to catch errors in the input fields. For instance, the system shows the message “Please provide a valid Email Address” when an incorrect email address is displayed, and the user moves away from the email address field. If they get the error and resolve the issue, the message is no longer displayed. If for some reason the user does not enter a phone number or email address, the system will notify them that at least one of the two have to be entered. If the phone number itself is not a numeric number and is instead some string of characters like “Phone Number”, then it will display an error as well.

**Functional Requirement #3**: The system should provide the correct results containing the list of courses for that particular class in a pop-up dialog.

The system successfully provides a modal (pop-up dialog window) that contains a grid of the appropriate courses obtained from the CUNYfirst system, according to the inputs entered by the user on the forum. The results shown there are exactly the results that would be achieved had the user just searched for these classes using CUNYfirst.

**Functional Requirement #4**: The CourseSpotter system notifies the user via email or text message or both when the course opens.

The system successfully notifies the user with an email message or text message or both of these messages when the course opens. This generally happens within seconds after the system detects that the class is in fact open. It is up to the user though to register for the course themselves in a timely manner through the CUNYfirst system, since our system doesn’t deal with the whole registration process.