ECSE 321 - Intro to Software Engineering
Design Specification Document - Deliverable 3

Harley Wiltzer
Camilo Garcia La Rotta
Jake Shnaidman
Robert Attard
Matthew Lesko

March 19, 2017

Contents

Part I

Unit Tests

1 Course Unit Test

METHOD UNDER TEST: Constructor
*Note: For cases in which a message, stating for a wrong input, is output from the system, then the changes/creations/deletions of objects should not persist.
Test Cases For: className Inputs

Test Cases for Constructor	className Input	cdn Input	TimeBudget Input
Test Case no.1	null	101	10.00
Test Case no.2	””	101	10.00
Test Case no.3	”1234567890”	101	10.00
Test Case no.4	” ”	101	10.00

Test Cases for Constructor	Expected Output
Test Case no.1	Course name cannot be empty!
Test Case no.2	Course name cannot be empty!
Test Case no.3	(className gets saved in persistence layer)
Test Case no.4	Course name cannot be empty!

Test Cases For: cdn Inputs

Test Cases for Constructor	className Input	cdn Input	TimeBudget Input
Test Case no.5	ECSE	101 and 101	10.00
Test Case no.6	ECSE	null	10.00
Test Case no.7	ECSE	-1	10.00
Test Case no.8	ECSE	”one hundred and one”	10.00
Test Case no.9	ECSE	” ”	10.00

Test Cases for Constructor	Expected Output
Test Case no.5	Cannot input Non-Unique CDN!
Test Case no.6	Cannot input empty CDN!
Test Case no.7	CDN must be positive!
Test Case no.8	Cannot input alphabetical characters for CDN!
Test Case no.9	Cannot input empty CDN!

Test Cases For: graderTimeBudget/taTimeBudget Inputs

Test Cases for Constructor	className Input	cdn Input	TimeBudget Input
Test Case no.10	ECSE	101	null
Test Case no.11	ECSE	101	””
Test Case no.12	ECSE	101	”ten”
Test Case no.13	ECSE	101	-10.00

Test Cases for Constructor	Expected Output
Test Case no.10	Cannot input empty TimeBudget!
Test Case no.11	Cannot input empty TimeBudget!
Test Case no.12	Cannot input alphabetical characters for TimeBudget!
Test Case no.13	Grader Budget must be positive! TA Budget must be positive!

METHOD UNDER TEST: addJob
Test Cases For: addJob Inputs

Test Cases	startTime	endTime	Day	Salary	Requirements	Instructor
Test Case no.14	null	null	null	null	null	null
Test Case no.15	10:00	15:00	Monday	15.00	Bachelors	Default Instructor

Test Cases	Expected Output
Test Case no.14	Invalid Input for Job!
Test Case no.15	(Job gets saved in persistence layer)

METHOD UNDER TEST: addJobAt
Test Cases For: addJobAt Inputs

Test Cases	Job1 Input	Index1 Input	Job2 Input	Index2 Input
Test Case no.16	Job1	null	Job2	null
Test Case no.17	Job1	1	Job2	1

Test Cases	Expected Output
Test Case no.16	Cannot add Job at null index!
Test Case no.17	Cannot add Job at occupied index!

METHOD UNDER TEST: addOrMoveJobAt
Test Cases For: addOrMoveJobAt Inputs

Test Cases	Job Input	Index Input	New Index Input
Test Case no.18	Job	1	2
Test Case no.19	Job	1	0

Test Cases	Expected Output
Test Case no.18	(Job gets moved to a new index successfully)
Test Case no.19	(Job gets moved to a new index successfully)

2 Job Unit Test

***Note:** Since the Classes Tutorial and Laboratory inherit from the Job class; the test cases for these two classes will be identical to the Job Class. The inputs aCourse and aInstructor are simply valid Course and Instructor inputs.
METHOD UNDER TEST: Constructor
Test Cases For: startTime and endTime Inputs

Test Cases	startTime	endTime	Day	Salary	Requirements	Course	Instructor
Test Case no.1	null	null	Monday	10.00	Bachelors	aCourse	aInstructor
Test Case no.2	””	””	Monday	10.00	Bachelors	aCourse	aInstructor
Test Case no.3	10:00	9:00	Monday	10.00	Bachelors	aCourse	aInstructor
Test Case no.4	21:00	22:00	Monday	10.00	Bachelors	aCourse	aInstructor

Test Cases	Expected Output
Test Case no.1	Input Time cannot be empty!
Test Case no.2	Input Time cannot be empty!
Test Case no.3	startTime cannot be before endTime!
Test Case no.4	Times cannot be outside working hours!

Test Cases For: Requirements Inputs

Test Cases	startTime	endTime	Day	Salary	Requirements	Course	Instructor
Test Case no.5	10:00	12:00	Monday	10.00	null	aCourse	aInstructor
Test Case no.6	10:00	12:00	Monday	10.00	””	aCourse	aInstructor
Test Case no.7	10:00	12:00	Monday	10.00	” ”	aCoursee	aInstructor

Test Cases	Expected Output
Test Case no.5	Requirements cannot be empty!
Test Case no.6	Requirements cannot be empty!
Test Case no.7	Requirements cannot be empty!

Test Cases For: Course and Instructor Inputs

Test Cases	startTime	endTime	Day	Salary	Requirements	Course	Instructor
Test Case no.8	10:00	12:00	Monday	10.00	PHD Student	null	aInstructor
Test Case no.9	10:00	12:00	Monday	10.00	PHD Student	aCourse	null

Test Cases	Expected Output
Test Case no.8	Invalid Course Input!
Test Case no.9	Invalid Instructor Input!

Test Cases For: Salary Inputs

Test Cases	startTime	endTime	Day	Salary	Requirements	Course	Instructor
Test Case no.10	10:00	12:00	Monday	null	PHD Student	aCourse	aInstructor
Test Case no.11	10:00	12:00	Monday	” ”	PHD Student	aCourse	aInstructor
Test Case no.12	10:00	12:00	Monday	”ten10”	PHD Student	aCourse	aInstructor
Test Case no.13	10:00	12:00	Monday	-10	PHD Student	aCourse	aInstructor

Test Cases	Expected Output
Test Case no.10	Salary cannot be empty!
Test Case no.11	Salary cannot be empty!
Test Case no.12	Salary cannot have alphabetical characters!
Test Case no.13	Salary cannot be negative!

Test Cases For: Day Inputs

Test Cases	startTime	endTime	Day	Salary	Requirements	Course	Instructor
Test Case no.14	10:00	12:00	null	10.00	PHD Student	aCourse	aInstructor
Test Case no.15	10:00	12:00	Saturday	10.00	PHD Student	aCourse	aInstructor

Test Cases	Expected Output
Test Case no.14	Date cannot be empty!
Test Case no.15	Date cannot be a weekend day!

3 Profile

Part II

Integration Tests

The integration tests are split into two parts. One part is the Java which tests the functionality of the controllers that operate in the Desktop and Mobile application in order to test the integration of multiple entity classes. The other is the PHP which tests how numerous classes can be integrated together.

4 Java

All together, the Java code has tested that all classes function harmoniously as they are needed to within our Mobile and Desktop application.

4.1 Profile Controller

This controller tests the integration of the profile class with the courses class as well as persistence and profile manager class. This test achieved 100% code coverage using ECLEmma.

It tested for null and empty input in all fields within the controller and ensured that it was handled. It also ensured that profiles that had incorrect input were not added to the system (were not persisted in the XML)

4.2 Course Controller

This controller tests the integration of the course class with the course controller, course manager and persistence. This test achieved 99.2% code coverage.

It tested for null and empty input in all String fields in the controller and ensured that it was handled. It also checked to see that CDNs, and time budgets could not be negative. Most importantly, the integration test ensured that exceptional input did not propagate into the persistence layer.

4.3 Application Controller

This controller tests the integration of the application class with the application controller, application manager, profile manager, profile controller, persistence, course class, profile class, and job class. This test achieved 100% code coverage

It tested that null and empty input in all String fields in the controller as well as negative salaries were handled by exceptions. It also tested that these errors did not propagate into persistence.

5 PHP

5.1 Job Integration

Given that the web client is inteded to be used by an instructor, the integration testing relates the job class to all the other classes in various scenarios and use-cases by utilising their controllers. While the unit test for profile and course used some implementation of their respective controllers, here we try to use only the controllers as much as possible for all interactions with the job class and the application controller, which handles the operations needed for the job class to be created and used.

5.2 Test Cases

The integration testing for the application controller and job class run through various input cases. These include creating a valid job with the controller, creating a job with no linked course, creating a job with an invalid course CDN, creating a job with no instructor, and the various combinations of remaining inputs for start and end time, salary and position.

As of yet integration testing for the web client is not complete and no coverage metrics can be generated. Ideally, after testing all possible use cases that apply to the web client (the model generated code that may not be used), coverage should be around 95%.

Part III

System Tests

System tests have been carried out incrementally as new use cases for the system have been added. Since the desktop app is the most mature of the three platforms in development, it has experienced by far the most system testing, as it currently implements approximately twice as many use cases as the other platforms combined. Therefore, the system tests for the desktop app will be described below, and will be structured by the use cases that they examine.

6 The Register Profile Use Case

Goal: To ensure that profiles are created properly using the RegisterView

Process

This use case was tested by entering various inputs to the RegisterView class, observing the output messages on the view, and verifying the persisted XML upon completion. Inputs were decided according to equivalence partitioning, so an input was tested for each class of error that can occur from user input, and of course a valid input was tested as well.

Results

All tests were passed successfully. Invalid input test cases all showed proper error messages on the view, and proper input test cases showed correct confirmation messages. Furthermore, it was noted that XML was only generated when proper input was submitted.

Conclusion

The RegisterView UI functions correctly, and reliably registers profiles to the system with proper persistence.

7 The Upload Course Data Use Case

Goal: To ensure that courses are created properly using the CourseView

Process

The process followed a very similar strategy to that of the system test described in section 6. Equivalence classes were chosen as test cases, and XML output as well as messages on the UI were verified.

Results

All tests were passed successfully. XML was created only when valid input was passed in, and all UI messages reflected the validity and errors of the input.

Conclusion

The CourseView UI functions correctly, and reliably uploads course data to the system with proper persistence.

8 The Publish Job Posting Use Case

Goal: To ensure that instructors may publish jobs for the courses that they teach, and to verify that only sane job postings are processed.

Process

This is the first system test that depends on the functioning of other use cases. Given the results of the previous system tests, there was confidence that the behavior of the current use case will not be hindered by its dependencies. It was first verified that the instructor may only choose to select courses that the profile is said to teach for the job posting. Then, it was verified that upon selecting a different professor, the list of courses to choose from updates (with a correct list of courses) immediately. Furthermore, aside from plain empty text fields, the PublishJobView has several input fields that can be erroneous due to incorrect input types, and even correct input types that represent non-sensical data. For example, the time budget fields must be integral types, and the start time of the job must occur before the end time. Therefore, configurations where the end time occurs before the start time were tested, and configurations where time budgets with non-integral time budgets were tested, along with the usual tests of empty fields and valid input. Furthermore, it was verified that XML was only generated upon *strictly valid* input, that is to say, input that has correctly-typed and non-empty fields *and* only logically-sane inputs. Finally, the application was closed and restarted. Then, the ApplicationView was opened to ensure that the job postings were present in the list, which would confirm that the persistence is functioning in such a way that its stored data may be retrieved Finally, the ApplicationView was opened to ensure that the job postings were present in the list, which would confirm that the persistence is functioning in such a way that its stored data may be retrieved.

Results

Unfortunately, not all test were passed immediately. It was noticed that it was possible to submit job postings with start times that occur after end times, which is undesirable. After many tests, the testers realized that by *nudging* the start time field, or simply changing its value and returning it to its initial value, the error vanished. Thus, the PublishJobView was modified to explicitly set the values of the start time and end time fields whenever the view refreshes. This ultimately solved the error. Otherwise, all other functionalities were successful, and XML was only being generated when the system interpreted that the input data was valid.

Conclusion

Although there was a slight failure early in testing, it appears to have been resolved, and currently the Publish Job Posting Use Case is considered to be functional. This use case has been tested extensively since its failures, as it is a dependency for several other use cases. Since the error was resolved, no other errors have been noticed.

9 The Apply to Job Posting Use Case

Goal: To ensure that students may apply to the job postings that were posted.

Process

This use case was very simple to test, since there are very few ways in which input can be invalid. In fact, the only inputs are the selections from two lists, which may be null. Therefore, there were four test cases, including one where the student list had no selection, one where the job list had no selection, one where neither list had a selection, and of course one where both lists had a selected value. It was ensured that proper error or confirmation messages were excreted upon each submission of input, and of course XML persistence was verified. In the process, the Publish Job Posting Use Case was also examined, as upon selecting jobs from the list the job's data appears in a text area, so job data was observed as a testament to the persistence in PublishJobView.

Results

All tests were passed. Appropriate error messages were output for all erroneous input configurations, and a correct confirmation message was output for the valid input class. Furthermore, persistence was only generated when valid input was submitted. Moreover, it was noticed that by observing the job data in the job description text area, all job data appears to be correct, further emphasizing the validity of the Publish Job Posting Use Case implementation.

Conclusion

The ApplicationView showed only correct behaviour throughout its vast amount of tests. Therefore it is believed to be functioning properly, and Students can reliably apply to Jobs.

10 The Hire Applicants Use Case

This use case was slightly difficult to test, because at this stage the XML persistence files were rather large, and in order to preserve efficiency, only a small amount of the XML is changed when hiring an applicant.

Goal: To ensure that instructors may choose to hire only students that have applied to the jobs that the particular instructor has posted. Furthermore, it was imperative to ensure that only one student may be hired for each job posting.

Process

Similarly to the Apply to Job Posting Use Case, the amount of erroneous input data is very small, since users are constrained to choose values from a list. In fact, due to the use of the `JComboBox` widget, there was only one list to choose from, making only two input classes: one where no student is selected, and one with valid data. Due to the intended functionality of the HireView, once a student is hired the list of students should be cleared immediately and disabled (so as to ensure that only one student is hired). Therefore, if a user is even capable of hiring two students for the same job, the test has already failed. The test process included attempting to hire an unselected student (and verifying that no persistence was made) and attempting to hire a student. When a student has successfully been hired, the persistence should show that the Job's `offerSent` property should be set to true, and that the Job should be included in the Student's jobs list.

Results

Unfortunately, these tests did not pass on the first shot, though this was due entirely to a logical error in the HireView class. Thankfully, persisting faulty data never occurred, rather the problem was that the data that was persisted did not correspond to the data seen on the view. This was actually due to a simple array-indexing flaw, that was fixed immediately. Upon fixing the logical error, all tests passed - that is to say, only valid inputs were accepted (and corresponding error/-confirmation messages appeared), and the persistence worked exactly as expected. Furthermore, upon clicking the Hire button, the list of students for the current Job clears, and that state persists.

Conclusion

After fixing a slight logical error, the HireView has only shown flawless functionality, and it is believed to be valid.

11 The Simplified Accept or Reject Job Offers Use Case

This use case was strategically implemented before the Admin's Modify Job Placement Use Case, so as to make system and integration testing easier.

Goal: To ensure that Students may view all pending Job offers, and reliably accept or reject them.

Process

Once again, there is only one source of potential error in the OfferView, so only two input classes exist. As usual, both input classes were tested, and it was verified that persistence was only modified in the case of valid input. In the Accept/Reject Job Offers behaviour, the only changes to persistence are that the Application that the student has an accepted from will have its `offerAccepted` field set to true, and once the student accepts or rejects a job offer, the corresponding Job should be removed from its jobs list (and thus from the OfferView's offers list).

Results

All tests were passed. It was noted that after clicking accept or reject, the selected job was removed from the list on the OfferView. This *does not* imply that the change was persisted, so this was examined by verifying the XML and by restarting the application and verifying that the offers list remained as it was before restarting. Furthermore, it was ensured that the `offerAccepted` field in the corresponding Applications were set when a job offer was accepted. All XML proved to be correct.

Conclusion

Based on the functionalities that were tested, due to the lack of any failures after numerous tests (and given such a small range of invalid input), it is believed that the Accept or Reject Job Offers use case has been implemented correctly.
However, it is important to note that not all behaviors of this use case were tested, as they have not been implemented yet. For example, when a Student chooses to reject a job offer, that Job's `offerSent` should be set back to false to allow for its professor to hire someone else. Therefore, for the sake of clarity, the Apply/Reject Job Offers Use Case has not been fully implemented, but the parts that have been implemented are functioning correctly. Remaining behaviours of this use case will be tested in further system tests later on in development, as those behaviours are implemented.

12 The Simplified Modify Job Placement Use Case

Goal: to ensure that the Admin user can remove job offers, save changes to job placements, and send job offers. To ensure that Students may only see job offers after the Admin has sent them.

Process

The View associated to this use case, ApproveOffersView, once again only allowed one class of invalid inputs, since there was one list to choose job offers from. However, there were three buttons that implemented interesting behavior. Fortunately, only one of these buttons acted based on the input, and the behaviour of that button does not invoke any persistence! This made testing slightly easier, because it was guaranteed that no invalid input could be persisted assuming the aforementioned button was functioning. This button is the Remove button, which removes a job offer. As a consequence, the job of the offer should be removed from the student's jobs list, and its `offerSent` field should be set back to false to allow the instructor to hire someone else. However, *these changes should not persist*, at least not yet. Next, the Save button was tested, which simply persists the current states of the ApplicationManager and ProfileManager, thus effectively revoking the job offers that were made. This was tested by verifying the persistence of the value mentioned above. Finally, the Approve button was tested. This button modifies an integer value in persistence, and allows the OffersView to be opened by a student. This was tested by first making changes to the job offers and saving them, followed by attempting to open the OffersView, which should fail and output an error message on the MenuView. Then, the ApproveOffersView was opened yet again, and Approve was selected. Following this, the OffersView was opened, followed by restarting the application and attempting to open it yet again. Of course, upon opening the OffersView, it was verified that the modifications to the job offers had taken effect.

Results

Surprisingly, all results passed without any trouble. It was seen that the OffersView can be opened only after the Admin has approved the job placements, which implies that students may only accept job offers after the Admins have approved of them. Furthermore, the job placement modifications were shown to be consistent, which implies that both the Remove and Save buttons are functioning as intended.

Conclusion

Considering the intended behaviour of this use case, it is believed that the implementation is functioning properly, as no tests have been shown to fail. However, it is important to know that it is planned to improve the robustness of this use case in the near future. Of particular note, it is intended to give more flexibility to the modification of job placements, rather than just removing them. Thus, it cannot be said that the Modify Job Placement Use Case is completely functional, as it has not been implemented in its entirety yet. However, the simplified (and still useful) Modify Job Placement Use Case has been shown to behave only successfully, and it is considered to be functional in its current state. As more features are added to this use case, further system tests will be carried out.

The system tests decribed above account for all system tests that have been done before Saturday, April 1, 2017. Furthermore, these system tests account for all use cases that have been implemented in the Desktop application by that date. As mentioned previously, further system tests will be created as the remainder of the use cases of the application are implemented on the various platforms.