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 Test Plan

1 Description

In this section we will present an extensive suite of test cases aimed to cover in the most efficient manner possible the Fantasy Basketball system we are to deliver. Following the paradigm of Test-Driven Development, we shall also implement the unit test cases. This will help keep the development of the system on track with the requirements imposed to the team.

2 Test Cases

All the classes to test will have the following pattern:

CLASS TO TEST/NOT TO TEST

- Reason to test/not to test
- attributes and related tests

2.1 Classes To Test

- **Course**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Its attributes TA/Grader time budget define how many students can apply to the position. Less relative to the application, but still needed for the correct behaviour of the process is the fact that the student must have experience in the given course in order to apply for a position as TA/Grader for it.
  - **Test Cases:**
    - **className**: test for null, empty, only numerical and only spaces inputs.
    - **CDN**: test for non-unique, null, empty, not alphabetic and only spaces inputs.
    - **className**: test for null, empty, not numerical and only spaces inputs.
    - **graderTimeBudget/taTimeBudget**: test for null, empty, not numerical and negative inputs.
    - **Jobs**: test for retrieval of job list
- **Job**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. It encapsulates half of the persistence aspect of the app which is to publish and view job postings.
  - **Test Cases:**
    - **CDN**: test for non-existent, null, empty, only numerical and only spaces inputs.
    - **requirements**: test for null, empty, empty and only spaces inputs.
    - **position**: test for null inputs.
    - **salary**: test for null, empty, not numerical and negative inputs.
    - **day**: test for null, empty and weekend inputs.

    * **startTime/endTime**: test for null, empty and outside of working hours inputs.

- **Tutorial**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Job postings are for either TA/Grader. In the first case, the availabilities of the TA must be compatible with those of the tutorial.
  - **Test Cases:**
    * test the same attributes as Course to ensure inheritance was correctly implemented by the UML

- **Laboratory**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Job postings are for either TA/Grader. In the first case, the availabilities of the TA must be compatible with those of the laboratory.
  - **Test Cases:**
    * test the same attributes as Course to ensure inheritance was correctly implemented by the UML

- **Profile**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. The system requires profile identifiers to discern which methods and attributes each instance has access to.
  - **Test Cases:**
    * **id**:test for non-unique, null, empty, not alphabetic and only spaces inputs.
    * **username**: test for non-unique, null, empty, not alphanumeric and only spaces inputs.
    * **password**: test for invalid difficulty, null and empty inputs.
    * **firstName/lastName**: test for null, empty, not alphabetical and only spaces inputs.

- **Student**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. The student is the only instance of profile allowed to apply to a job posting.
  - **Test Cases:**
    * test the same attributes as Profile to ensure inheritance was correctly implemented by the UML
    * **experience**: test for null, empty and only spaces inputs.
    * **degree**: test for null inputs.
    * **Jobs**: test for retrieval of job list

- **Instructor**

- **Reason to test:** This class is intrinsic to the job posting and application transaction. The instructor is the only instance of profile allowed to post jobs.
- **Test Cases:**
  - test the same attributes as Profile to ensure inheritance was correctly implemented by the UML
  - **experience**: test for null, empty and only spaces inputs.
  - **degree**: test for null inputs.
  - **Jobs**: test for retrieval and modification of job list
  - **Application**: test for retrieval of application list
  - **course**: test for retrieval of course list

- **Application**
  - **Reason to test:** This class is necessary for the Student to apply for a job. Each application is associated to a student and a job posting; hence one needs to test for different Student attributes for different Jobs.
  - **Test Cases:**
    - **Test Apply for a Job:** test for error if hours are not compatible, test for error if job is null, and test for error if student is null.

- **Admin**
  - **Reason to test:** This class is intrinsic to the creation of a course, the creation of a Student and Instructor, and application transaction. Its attributes inherit from profile. They define information about the person that is registering an admin profile; hence it is needed to test for different attribute inputs and for successful creation of classes.
  - **Test Cases:**
    - test the same attributes as Profile to ensure inheritance was correctly implemented by the UML
    - **Application Transaction:** test for successful application transaction between Student and Job Posting
    - **Successfully Create Classes:** test to successfully create Instructor, and Student.

- **Persistence**
  - **Reason to test:** This class is intrinsic to the job posting and application transaction. Without persistence capability the controllers have no data from which to derive the desired outcome
  - **Test Cases:**
    - creation, modification and deletion of Course, Job, Application and Profile instances.

2.2 Classes to Not Test

- **Application Manager**
  - **Reason to not test:** This class acts as a container for Application and Job. It does not perform any action; hence, there are no actions for it to test.
- **Profile Manager**
  - **Reason to not test:** This class acts as a container for Student, Instructor, and Admin. It does not perform any action; hence, there are no actions for it to test.
- **Course Manager**
  - **Reason to not test:** This class acts as a container for Course. It does not perform any action; hence, there are no actions for it to test.

3 Techniques and Tools

The system is to be designed in a complete Test-Driven Development paradigm. Hence all test cases will be written before the actual controllers are implemented. Furthermore, the order in which the tests will be passed follows the same order as the requirements derived in the Requirements document of deliverable #1. This will aid the classification and prioritization of each bi-weekly runs' objectives. In parallel to this method, the team will rely heavily on code revisions by the senior members of the team to ensure the code written is up to visual, performance and logical standards. In terms of frequency, individual nightly tests will be ran on all platforms of the system and bi-weekly builds and test will be done every Saturday to ensure the deliverable validates and verifies the given requirements. To facilitate the testing process, the team will use the following tools:

- **Unit Test Framework:** Automated, well documented and supported system to allow a standardized set of tests to be done regardless of the platform. JUnit and PHPUnit will be used.
- **Test Coverage Tool:** Ensure that the measure to which the tests and actions cover the source code is optimal by function, statement, branch and condition standards. Its outputs will be used during the code revision sessions. EclEmma and PHPUnit will be used for this purposes.

4 Coverage Statistics

TODOOOOOOO To the extent possible, include a quantification of your testing goal (e.g., N% statement coverage or M% branch coverage).

Furthermore, discuss how often your tests are executed, i.e., what triggers your tests to be run? Finally, describe any differences between your desktop/laptop app, mobile app, and web app when it comes to unit testing.