Testing Plan

Western Michigan University Cohort Scheduler

CS 4900 - Dr. Kapenga

By Alex Markules, Jacob Kampf, and Grant Farnsworth

Table of Contents

References 3

Introduction 3

Definitions 4

Test Items 4

Features Not To Be Tested 5

Approach 6

Pass/Fail Criteria 8

Suspension Criteria 8

Test Deliverables 9

Testing Tasks 9

Environmental Needs 10

Responsibilities 10

Staff and Training Needs 10

Schedule 11

Risks and Contingencies 11

Approvals 12

1. References:

Rajkumar. “Test Plan Template with Detailed Explanation.” *Software Testing Material*, 9 Sept. 2018, [www.softwaretestingmaterial.com/test-plan-template/#How\_To\_Prepare\_Effective\_Test\_Plan](http://www.softwaretestingmaterial.com/test-plan-template/#How_To_Prepare_Effective_Test_Plan).

Hunt, Andrew, and David Thomas. *Pragmatic Unit Testing: in Java with JUnit*. Vol. 2, The Pragmatic Programmers, 2009.

Catchorg. “Catchorg/Catch2.” GitHub, 2017, github.com/catchorg/Catch2/blob/master/docs/Readme.md.

1. Introduction

The purpose of this project is to develop a web application to be used by the advising staff at Western Michigan University. The application creates schedules for “Cohorts” based on a set of predetermined requirements. Each cohort’s classes cannot conflict with each other, and no more students may be scheduled in a class than allowed by the class’s capacity. To do this, we will design and test a project to run on a web server. This server will host a web application where academic advisors can enter information about the requirements for each cohort, and upload excel files containing lists of classes. From the web application, advisors can trigger the start of another piece of software on the server, which will create all possible schedules, score them based upon a list of predefined criteria, and pass the best ones back to the web application to be displayed to the user.

1. Definitions

**Cohort** - a collection of 18 to 24 students defined by academic advisors. These students are typically incoming freshmen of the same major. The goal is to schedule all students in each cohort in the same classes so that they can learn and study together.

**Schedule** - the term schedule typically refers to one of 2 things depending on context:

1. A set of classes which do not conflict in time for a given cohort
2. Sets of classes for all cohorts which do not conflict in time for each cohort and capacity for each class

**Advisor** - advisor refers to academic advisors, the people who will be using this software. They currently create schedules for cohorts by hand, spending months coming up with the final product.

1. Test Items
   1. Scheduler
      1. Ranking Functions
      2. Valid Schedule generation
      3. Data Input handling
   2. C++ Rest API
      1. Server that receives job from node server
      2. Part that send data back to Node server
      3. Part that passes input into Scheduler
   3. Node server
      1. RESTful API for creating cohort requirement, and assignment records
      2. Login, account registration, and session handling functions
      3. Full CRUD operation on all tables in the NoSQL database
   4. Angular Front end
      1. Typescript functions
      2. Services
2. Features Not To Be Tested
   1. Testing frameworks
      1. Mocha and Chai for Node.JS
      2. Catch for C++
      3. CUnit for C
   2. Widely used public libraries
      1. Node Modules
      2. C++ Boost library
      3. Standard C libraries
3. Approach
   1. Unit Tests
      1. Our code will be written in a functional style where possible to make unit testing easier
      2. Mock objects will be used where necessary to ensure unit tests are modular
      3. Valid input, invalid input, and boundary conditions will be tested for each function
   2. Automated Tests
      1. We will use makefiles in each source directory to automate both testing and running our project
      2. All unit tests will be run automatically in this way before commiting to the master branch
   3. Functional Tests
      1. Our functional test will consist of a weekly run of the scheduling algorithm via the web application running on a cloud server (Amazon EC2)
      2. The results of this test will be reviewed by the team, factoring in the time taken, the validity of the schedules generated, and that no errors occured in the process of initiating, scheduling, and returning schedules
   4. Usability and Acceptance Testing (UAT)
      1. Throughout the Spring semester (January 7 - April 26, 2019) we schedule bi-weekly meetings with the CAS and CEAS academic advisors
      2. At these meetings, we will show them the User Interface we have created, as well as the schedules we generate
      3. We will ask the advisors for feedback and change the project as necessary based on their input
   5. Security Tests
      1. Once our application is deployed in its final location (a server owned by WMU) we will implement security tests to meet the following criteria
         1. Only academic advisors can create accounts
         2. Only those with a proper username and password may log in
         3. Only those who are logged in may view the Cohorts, Schedules, and Assignments pages
         4. Only those who are logged in may initiate scheduling
         5. Only those who are logged in may access RESTful service calls in the application
      2. We will validate these parameters ourselves using OWASP, and enlist members of the CS 4910 class to try and break in to our application in a white box testing style
4. Pass/Fail Criteria
   1. Unit Tests Cover all scenarios for all functions and all pass
   2. All tests are set up to run with a single command, and are verified to pass before committing to master
   3. Functional testing results in a valid set of schedules which are ideal for the advisors’ needs
   4. The application is easy to use and covers all use cases. This will be confirmed by the academic advisors in the CEAS and CAS
   5. When possible, recommendations from OWASP are followed and flaws corrected. Additionally, all vulnerabilities found by pen testers shall be patched.
5. Suspension Criteria
   1. Functional testing should suspend in the case that any major functionality is not functional
   2. When usability testing is done, it should not be repeated until appropriate changes have been made
   3. When flaws are found in security, all testing should stop until those flaws are patched
   4. If functional testing fails, usability and security testing will be paused until functional tests pass.
6. Test Deliverables
   1. Unit test files containing all test cases
   2. Report detailing any failed tests. This will be kept in the testing directory of the project and updated periodically, when automated tests are run
   3. Functional Test Results: The top 20 Schedules generated by the program
   4. Feedback Report: after every meeting with the academic advisors, we will compile a document listing the feedback they gave us on the project
   5. Security testing report: a report detailing all flaws found by OWASP, and all flaws found by penetration testers, as well as what actions we took for each
7. Testing Tasks
   1. Create testing environments for all code
   2. Create unit tests and run for each module
   3. Document test results
   4. Run tests before merging new code in to the program
   5. Frequently test usability with advising staff (At least once every two weeks)

1. Environmental Needs
   1. Amazon EC2 server running Ubuntu 16.04 LTS
   2. WMU server account running Ubuntu 16.04 LTS

1. Responsibilities
   1. We are all responsible for writing unit tests for code we write, including as many possible cases as we can think of
   2. All tests should be set up to run with the command “make test” from the root directory of the project
   3. We are responsible for writing Failed Tests, Functional Tests, Feedback, and Security reports
2. Staffing and Training Needs
   1. Testing staff will consist of the developers and the advising staff at WMU.
   2. Advising staff will be utilized for Usability Testing, as they are the target population for the application.
   3. In regards to Usability Testing, a basic explanation of the application may be necessary before use.
3. Schedule
   1. Unit tests will be completed by the end of version 1 development of the final product. This will be completed by the end of January
   2. Automated tests will also be completed by the end of January, and run periodically as changes are made
   3. Exact requirements for our functional testing will be determined at our meeting with the academic advisors in the week of January 14. After that, this test will be done before every change to the master branch of the project
   4. The first feedback report will be based on our Jan 14 meeting with the academic advisors. There will be a similar report every 2 weeks after that, with the last being the official acceptance test

1. Risks and Contingencies
   1. Missed web flaws: If any security flaws in our web application are missed by OWASP and our testers, there could be potential vulnerabilities. This risk is small since it is unlikely that anyone other than the intended users will find any information held here valuable
   2. Node modules: if any modules used by our project are found to have vulnerabilities, the project will need to be updated as soon as possible. In order to prevent this, we will check the modules we use periodically for security updates, and look into notifications for module updates
   3. Input parsing errors: file uploads, and user input both represent areas which need to be heavily validated by our program. In order to prevent errors in these errors, we will devote substantial time to ensuring all input is well validated.
2. Approvals
   1. Dr. Kapenga will give the final approval of the testing plan.