Test Plan: Revision 1

Group 2 - Genzter

Binu, Amit - binua - 400023175
Bengezi, Mohamed - bengezim - 400021279
Samarasinghe, Sachin - samarya - 001430998
Professor: Dr. Bokhari
Lab: L01

December 6, 2017

Contents

List of Figures

List of Tables

1 Revision History

Date	Version	Description	Author
23/OCT/17	0.0	Created Test Plan	Mohamed Bengezi, Amit Binu, Sachin Samaras
27/OCT/17	0.0	Updated Test Plan	Mohamed Bengezi
21/NOV/17	1.0	Updated for Revision 1	Mohamed Bengezi
06/DEC/17	1.0	Changed some tests to automated	Mohamed Bengezi

Table 1: Revision History

**Please note that all additions and modificiations are in blue

2 Introduction

2.1 Purpose

The purpose of this document is to document the testing plans and procedures that the system will undergo. It will outline the general definition/purpose of each type of test, and how those tests will be implemented with regards to this project. The goal of this project is to create a viable set of schedules for the user based on the inputted courses. All corresponding core's, labs, and tutorials must be included in each timetable.
The automated testing shall include unit testing performed by the Mocha framework for javascript. Functional testing shall consist of input testing, along with conflict tests, and perfect input tests. Structural testing shall include performance tests, such as a large number of courses as input, as well as dataset traversal tests. Fault and Mutation testing will occur throughout development. Manual testing, such as static testing, shall be done primarily by group code walkthrough's & inspections, in order to check syntax and logic, and gain a better understanding of the program code as a whole. Dynamic testing shall include using a number of predetermined schedules from various programs, and using the courses on each schedule as input, and determining that each necessary core, lab, and tutorial are listed in the generated timetable.

2.2 Definitions

- Static Testing: Testing that does not involve program execution. This includes syntax checking, document and code walkthroughs/inspections, correctness proofs, etc.
- Dynamic Testing: Requires the program to be executed. Test vases are run and results are checked against the expected behaviour.
- Unit Testing: Finds differences between the design model of a unit and its corresponding implementation
- Functional (black-box) testing: Testing a system based solely on the requirements, without knowing or acknowledging the implementation.
- Structural (white-box) testing: Testing a system based on its implementation. This tests the non-functional requirements are structure of the internals of the system.
- Manual testing: Conducted by people, includes by-hand test cases, structured walkthroughs, and code inspections
- Automated testing: Using software tools such as Mocha, or JUnit, to automate the testing. They run the specified test cases.

2.3 Objectives

This test plan will allow the team to head into the testing phase in an organized and informed manner, and with a clear goal in mind. These tests ensure that the product will be working as it should be when released for users. The plan will also ensure that testing is conducted properly, and any interested parties can learn about the testing of the product.

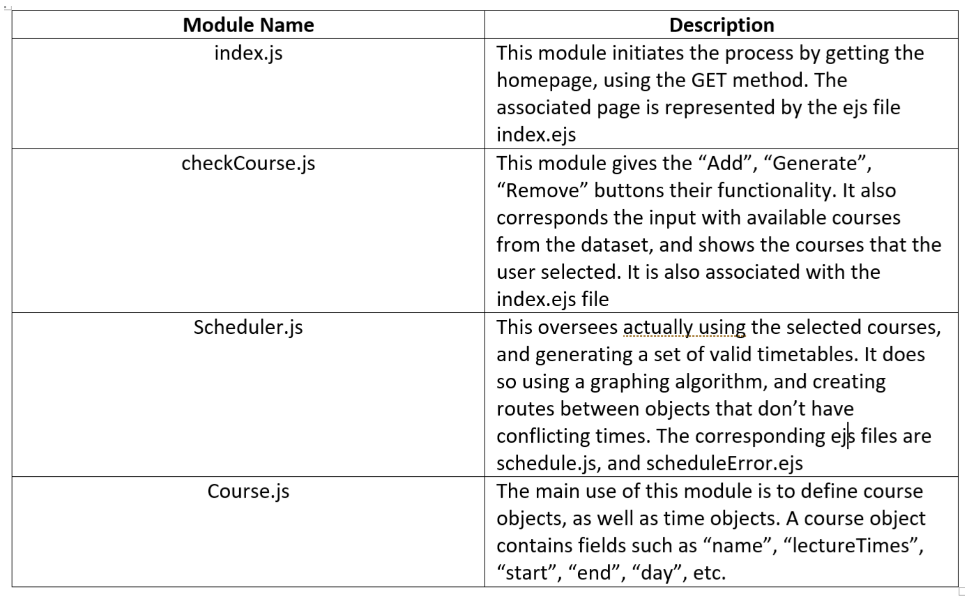
2.4 References

Project inspiration: `http://timetablegenerator.io`
Open sourced similar project: `https://github.com/ash47/TimetableGenerator`

3 Plan

3.1 Software Description

The program uses Node js for the backend, and HTML/CSS for the front end. The main modules in the program are the index.js module, which contains the code for the main page, the checkCourse.js module, which checks and ensures all user inputs are valid and viable, the scheduler.js module, which is the main algorithm for creating the timetables, the Course.js module, which defines the course objects used throughout the program, and all the corresponding ejs files, which contain the HTML code for the front end of each page.



Module Name	Description
index.js	This module initiates the process by getting the homepage, using the GET method. The associated page is represented by the ejs file index.ejs
checkCourse.js	This module gives the "Add", "Generate", "Remove" buttons their functionality. It also corresponds the input with available courses from the dataset, and shows the courses that the user selected. It is also associated with the index.ejs file
Scheduler.js	This oversees actually using the selected courses, and generating a set of valid timetables. It does so using a graphing algorithm, and creating routes between objects that don't have conflicting times. The corresponding ejs files are schedule.js, and scheduleError.ejs
Course.js	The main use of this module is to define course objects, as well as time objects. A course object contains fields such as "name", "lectureTimes", "start", "end", "day", etc.

Figure 1: Software Description

3.2 Test Team

The entire Genzter team will be in tasked with testing. Testing shall be evenly divided between members.

3.3 Automated Testing Approach

As stated above, automated testing will be done using the javascript testing framework called Mocha. Accompanying Mocha will be an assertion library called Chai. Unit tests will be set up, and the framework will be used to automatically run through all the unit tests, and reveal any faults that may reside in the program. The unit tests will include normal input, boundary cases, as well as extreme cases and conflicting cases. This will ensure that the program functions expectedly, and that no special cases were missed. In other words, this automated testing approach will make sure that in any case, the program will have a path to follow, and its behaviour can be predicted.

3.4 Testing Tools

For unit testing we will be using the testing framework Mocha. System testing will be conducted on internet browsers. Code analysis will be done by the Genzter team.

3.5 Testing Schedule

The testing schedule is included in the Gantt chart, located in the Project Schedule folder in the main directory.

4 System Test Description

4.1 Tests for Functional Requirements

4.1.1 Test Area 1 Conflict Detection Test

1. Type: Dynamic
   Initial State: Homepage
   Input: SFWRENG 3RA3, EARTHSC 2EI3
   Expected Output: Conflict error page
   The courses in the input field will be selected and added in the UI manually. The expected result, an error page, will be checked against the actual output. The courses selected must have conflicting time slots.

2. Type: Dynamic
   Initial State: Homepage
   Input: RELIGST-2TA3, SFWRENG 3BB4
   Expected Output: Conflict error page
   The courses in the input field will be selected and added in the UI manually. The expected result, an error page, will be checked against the actual output. The courses selected must have conflicting time slots.

3. Type: Dynamic
   Initial State: Homepage
   Input: BIOLOGY 2F03, SFWRENG 3MX3
   Expected Output: Conflict error page
   The courses in the input field will be selected and added in the UI manually. The expected result, an error page, will be checked against the actual output. The courses selected must have conflicting time slots.

4. Type: Dynamic
   Initial State: Homepage
   Input: MATLS 2B03, EARTHSC 2EI3
   Expected Output: Conflict error page
   The courses in the input field will be selected and added in the UI manually. The expected result, an error page, will be checked against the actual output. The courses selected must have conflicting time slots.

4.1.2 Test Area 2: Output Timetable

1. Type: Automated
   Initial State: Homepage
   Input: SFWRENG 3XA3, SFWRENG 3BB4, SFWRENG 3DB3, SFWRENG 3MX3, COMMERCE 1AA3
   Expected Output: Valid Schedule in a timetable format.
   The courses in the input field will be selected and added in the UI manually. The expected result, a color coded schedule, will be checked against the actual output. Generated timetable must have every core, tutorial, lab etc. required for each course. It will be manually checked against the dataset and our own timetables

2. Type: Automated
   Initial State: Homepage
   Input: ENGINEER-1C03, MATH-1ZC3, MATH-1ZB3, ECON-1BB3, PHYSICS-1E03, MATLS-1M03
   Expected Output: Valid Schedule in a timetable format.
   The courses in the input field will be selected and added in the UI manually. The expected result, a color coded schedule, will be checked against the actual output. Generated timetable must have every core, tutorial, lab etc. required for each course. It will be manually checked against the dataset and our own timetables

3. Type: Manual
   Initial State: Homepage
   Input: RELIGST-2TA3, RELIGST-2QQ3, POLSCI-2I03, POLSCI-4CA3
   Expected Output: Valid Schedule in a timetable format.
   The courses in the input field will be selected and added in the UI manually. The expected result, a color coded schedule, will be checked against the actual Expected Output. Generated timetable must have every core, tutorial, lab etc. required for each course. It will be manually checked against the dataset and our own timetables

4. Type: Manual
   Initial State: Homepage
   Input:EARTHSC-2EI3, EARTHSC-2C03, GEOG-2UI3, GEOG-2GI3, BIOLOGY-2F03
   Expected Output: Valid Schedule in a timetable format.
   The courses in the input field will be selected and added in the UI manually. The expected result, a color coded schedule, will be checked against the actual output. Generated timetable must have every core, tutorial, lab etc. required for each course.

It will be manually checked against the dataset and our own timetables

4.1.3 Regression Testing

Regression testing will continually happen to ensure all features continue to function ideally, no matter the changes occurring to the program.

4.1.4 Parallel Testing

Please refer to this section

4.2 Tests for Non-functional Requirements

4.2.1 Test Area 3: Speed Performance

1. Type: Manual
   Initial State: Homepage
   Input: SFWRENG 3XA3, SFWRENG 3BB4, SFWRENG 3DB3, SFWRENG 3MX3, COMMERCE 1AA3
   Expected Output: Output should be displayed within 3 seconds
   The input courses shall be selected, and the the generator is timed from when the "Generate" button is pressed.

2. Type: Manual
   Initial State: Homepage
   Input: ENGINEER-1C03, MATH-1ZC3, MATH-1ZB3, ECON-1BB3, PHYSICS-1E03, MATLS-1M03
   Expected Output: Output should be displayed within 3 seconds
   The input courses shall be selected, and the the generator is timed from when the "Generate" button is pressed.

4.2.2 Test Area 4: UI

1. Type: Manual
   Initial State: Homepage
   Input: Any McMaster Course
   Expected Output: Add course to list of courses
   Any McMaster course is selected, and the "Add" button is pressed. This is to test the ability of the UI to retrieve input from the user.

2. Type: Manual
   Initial State: Homepage with list of courses chosen
   Input: Pressing the "Remove" button
   Expected Output: Removal of the selected course
   Starting off with a list of courses, the "Remove" button will be pressed to test the ability of the UI to modify the course list at the request of the user.

3. Type: Manual
   Initial State: Homepage with list of courses chosen
   Input: Pressing the "Generate" button
   Expected Output: Timetable
   Starting off with a list of courses, the "Generate" button will be pressed to test the ability of the UI to display output at the request of the user.

4.2.3 Test Area 5: Maintainability and Robustness

1. Type: Static
   Initial State: Not running
   Input: Code walkthrough and inspection
   Expected Output: N/A
   This test is a static test for confirming the syntax and structural integrity of the backend of the program.

2. Type: Dynamic
   Initial State: Homepage
   Input: ENGINEER-1C03, MATH-1ZC3, MATH-1ZB3, ECON-1BB3, PHYSICS-1E03, MATLS-1M03, ECON-1B03, MATH-1ZA3, ENGINEER-1D04, CHEM-1E03
   Expected Output: Valid Schedule in a timetable format
   Actual output: The result was a valid, working schedule with all necessary items

1. Type: Static, Automated
   Initial State: Not running
   Input: Code inspected automatically by JSHint
   Expected Output: N/A
   This test is an automatic test for confirming the syntax and structural integrity of the backend of the program.

4.2.4 Test Area 6: Browser Compatability

1. Type: Static
   Initial State: Not running
   Input: Visit website using different browers
   Expected Output: N/A
   This test is a static test for confirming that the website works for various browsers.

4.2.5 Traceability

Requirement	TA1	TA2	TA3	TA4	TA5
2.7.1	X				
2.7.2		X	X	X	
3.2				X	
3.3			X		
3.5					X

Table 2: Traceability Matrix

5 PoC Test

5.1 Conflict Detection

1. Type: Dynamic
   Initial State: Homepage
   Input: BIOLOGY 2F03, SFWRENG 3MX3
   Expected Output: Conflict error page
   The courses in the input field will be selected and added in the UI manually. The expected result, an error page, will be checked against the actual output. The courses selected must have conflicting time slots.

5.2 Schedule Output

1. Type: Manual
   Initial State: Homepage
   Input: SFWRENG 3XA3, SFWRENG 3BB4
   Expected Output: Valid Schedule in a timetable format.
   The courses in the input field will be selected and added in the UI manually. The expected result, a color coded schedule, will be checked against the actual output. Generated timetable must have every core, tutorial, lab etc. required for each course. It will be manually checked against the dataset and our own timetables

6 Comparison to Existing Implementation

The project will be benchmarked against two existing implementations, `http://timetablegenerator.io` and `https://github.com/ash47/TimetableGenerator`. Mainly, we will be comparing our project with `http://timetablegenerator.io` because it uses the McMaster database as well, so we can fully compare the generated timetables. The outputs are fairly similar, so comparison should not be very difficult. The comparison will be done manually by the testing team.

7 Unit Testing Plan

Unit tests will be defined by specific modules and their functionality. So for example, from the software description figure, index will have a set of unit tests, and checkCourse will have its own, and so forth. Each module will have its own type of tests. We may need to implement stubs, to imitate connection to the database, to ensure the right functions are being called, etc. As for code coverage, equivalence will definitely be used throughout the majority of the test cases. Boundary cases will be used. An example of such a boundary case would be using a full year, six unit course as input, and testing the behaviour of the program. This is a boundary case because the program has had some trouble with full year courses. Lastly, control-flow testing will be implemented in order to extensively test as much lines of code as possible, and ensure that every path and edge is covered. As for specific details, that is still unclear at the moment, but will be updated and posted as soon as a defined and detailed plan is created.