SE 3XA3: Design Documentation Rev 1
Group 2

Bengezi, Mohamed - bengezim
400021279

Binu, Amit - binua
400023175

Samarasinghe, Sachin - samarya
001430998

December 3, 2017

Contents

1 Revision History **4**

2 Introduction **5**
2.1 Brief Overview 5
2.2 Design Overview 5
2.3 Document structure 5

3 Anticipated & Unlikely Changes **6**
3.1 Anticipated Changes 6
3.2 Unlikely Changes 6

4 Module Hierarchy **6**

5 Connection of Requirements and Design **7**

6 Module Decomposition **8**
6.1 Behaviour Hiding Module 8
6.1.1 Input Parameters Module 8
6.1.2 Scheduler Module 8
6.1.3 Output Module 8
6.2 Software Decision Module 8
6.2.1 Course Module 9

7 Traceability Matrix **10**
7.1 Requirements Traceability Matrix 10
7.2 Anticipated Changes Traceability Matrix 11

8 Uses Hierarchy **12**

9 MIS's of app.js **13**

10 MIS's of checkCourse.js **15**

11 MIS's of index.js **17**

12 MIS's of scheduler.js **19**

13 Schedule **29**

List of Tables

1 Revision History 4
2 Module Hierarchy 6
3 Module Labels 10
4 Requirements Traceability Matrix 10
5 Anticipated Changes Traceability Matrix 11

List of Figures

1 Uses Relation 12

1 Revision History

Table 1: Revision History

DATE	AUTHOR	DESCRIPTION	REVISION
08/NOV/17	Bengezi, Mohamed	Created Initial Document	0
08/NOV/17	Samarasinghe Sachin, Binu Amit	Initial Draft	0
28/NOV/17	Bengezi Mohamed	Fixed module decomposition and other minor errors	1
03/DEC/17	Bengezi Mohamed	Fixed req. list discussed in marking scheme	1

**Please note all changes from the previous version are written in blue

2 Introduction

2.1 Brief Overview

This document outlines the Module Guide (MG), Module Interface Specification (MIS), and a Gantt chart detailing the schedule of implementation and testing. This project is a timetable generator, designed for McMaster University students, which allows students to select desired courses, and generates a set of schedules based on the input.

2.2 Design Overview

This document succeeds the SRS and precedes the MIS document. The SRS defines the functional and non-functional requirements of the system. The MG serves as a guide to users, and introduces new concepts, as well as prepares them for the MIS. The MIS breaks down what the system consists of, in a readable form. This means that the modules in the system are defined, and broken down. The SRS document's requirements are continually looked at and checked while creating the MG & MIS, as well as used as a guideline when designing and implementing the modules. This document describes the Architectural Design and Detailed Design of the project. The goal of this document is to ensure the design and maintenance of the timetable generator by facilitating the successful interaction between modules in the program. The MG and MIS are created using the Separation of Concerns, Modularity, and Abstraction design principles. Separation of Concerns is when a problem is divided into smaller parts, or modules, and tackled individually. Modularity enables Separation of Concerns through Modular Decomposition, which is discussed in depth further in this document. Abstraction is used throughout in order to focus on the important aspects rather than minute details.

2.3 Document structure

This document is structured such that it contains an MG, which consists of likely/unlikely changes, module hierarchy, module decomposition trace-ability matrix between the SRS requirements and the modules. The document also contains MIS's for the various modules in the project, and lastly contains a schedule of the design, implementation, and testing tasks in the form of a Gantt chart. This structure is optimal because the Module Guide serves as an introduction, and equips the user with the knowledge necessary in order to fully understand the MIS. The MIS then allows users to see how the system is broken down into modules, and how those modules are broken down.

3 Anticipated & Unlikely Changes

3.1 Anticipated Changes

1. AC1: Addition of a 'Generate again' button
2. AC2: Drag and drop feature on output schedule
3. AC3: Input UI
4. AC4: Grammar/Syntax of JS/HTML/CSS

3.2 Unlikely Changes

1. Input format (Course codes)
2. Output (Will always be a schedule)
3. Schedule Generation Algorithms

4 Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding Module	N/A
Behaviour-Hiding Module	Input Parameters Module Scheduler Module Output Module
Software Decision Module	Course Module

Table 2: Module Hierarchy

5 Connection of Requirements and Design

In the SRS document, there are functional requirements, such as getting user input, computing a valid schedule, and outputting in schedule format, and there are non-functional requirements, like simple usability, decent performance, and maintainability. Design decisions were made in order to satisfy these requirements.
For gaining user input, as well as satisfy usability, a simple and elegant welcome page is designed. The page contains an input box that accepts course codes, and clear and concise instructions on the format of the course codes, as well as when to click the 'Add' and 'Generate' buttons. One the user has clicked the correct buttons, the input is then stored and used. In order to compute a valid schedule, a module dedicated to this is implemented, where it uses a graphing algorithm in order to create a schedule. A schedule output is achieved by creating an HTML table using an EJS module. Performance and Maintainability are ensured by following efficient programming procedures, as well as keeping clean and clear source code.

6 Module Decomposition

This section decomposes each module so that its secret, service, and implemented by information are known, and easy to understand. A secret is what the module is hiding, under the design principle of information hiding, by David Parnas. [1] The service is what the module does. The implemented by section just describes how it is implemented

6.1 Behaviour Hiding Module

Secret: Behaviours of System
Service: Implementing the various expected behaviours of the system
Implemented By: Not implemented by a specific module, rather made up of modules specified below

6.1.1 Input Parameters Module

Secret: How input is recieved
Service: Allowing users to enter desired information, in this case course codes
Implemented By: Main.js, checkCourse.js

6.1.2 Scheduler Module

Secret: Uses a Graphing algorithm
Service: Generates a schedule with selected input
Implemented By: scheduler.js

6.1.3 Output Module

Secret: Uses an HTML table
Service: Creates a visual table with time-slots of selected courses
Implemented By: schedule.ejs

6.2 Software Decision Module

Secret: Generic Modules and Data Structures
Service: House generic modules, and data structures that can be used by other applications
Implemented By: N/A

6.2.1 Course Module

Secret: Course Data Structure
Service: Provide a data structure for a course object
Implemented By: Course.js

7 Traceability Matrix

The requirements used in the following matrix are from the SRS Document. The labels, such as 3.1, refer to the section in the SRS document.

Module	Label
Hardware Hiding Module	M1
Input Parameters Module	M2
Scheduler Module	M3
Output Module	M4
Course Module	M5

Table 3: Module Labels

7.1 Requirements Traceability Matrix

Requirements	Modules
2.7.1	M2, M5
2.7.2	M3, M4, M5
3.1	M4
3.2	M4
3.3	M2, M3, M4, M5
3.5	M2, M3, M4, M5

Table 4: Requirements Traceability Matrix

**Please note, the requirements numbers refer to the section numbers of the corresponding requirements in the SRS Document. So 2.7 would be found in the SRS document at section 2.7

7.2 Anticipated Changes Traceability Matrix

Anticipated Changes	Modules
AC1	M3, M4
AC2	M3, M4
AC3	M2
AC4	M2, M3, M4, M5

Table 5: Anticipated Changes Traceability Matrix

8 Uses Hierarchy

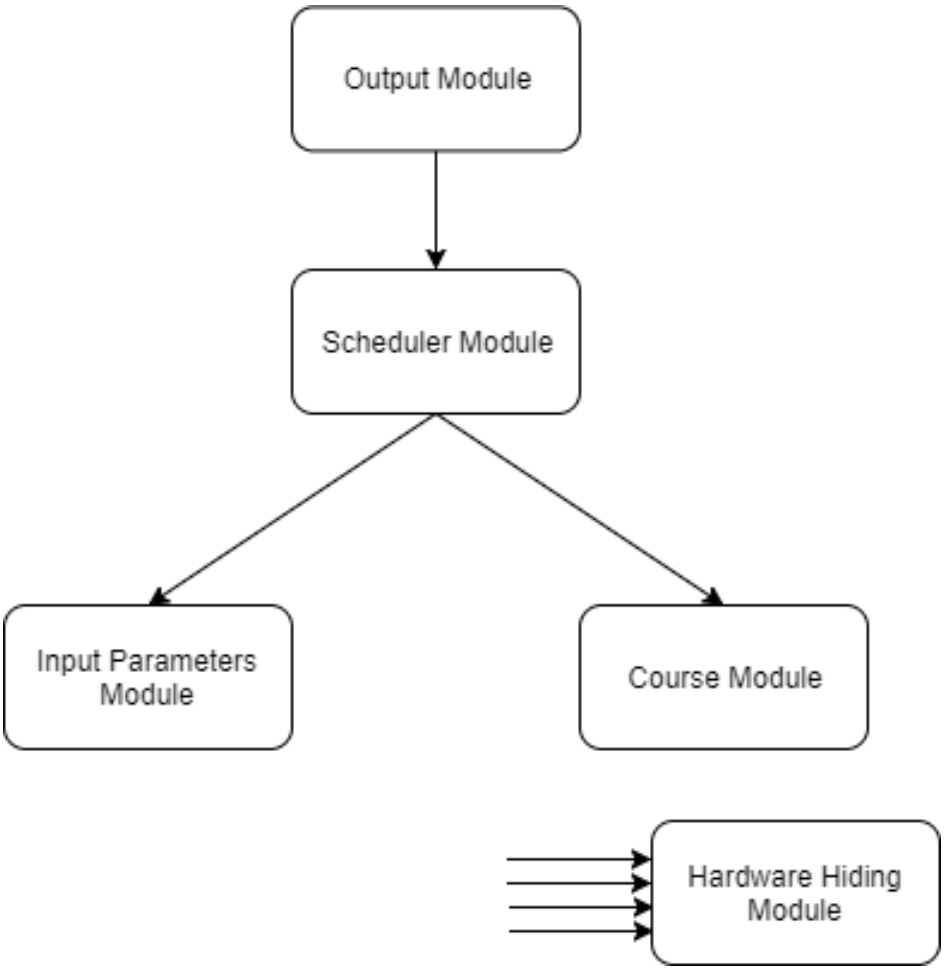


Figure 1: Uses Relation

9 MIS's of app.js

Template Module

course

Uses

N/A

Syntax

Exported Types

app = ?

Exported Access Programs

Routine name	In	Out	Exceptions
startProgram	-	-	UrlNotFoundException

Semantics

State Variables

$courseIds$:= Course ids of all the courses offered in McMaster University

$allCourses$:= Matrix of all the information about all the courses offered in McMaster University

Environment Variables

Screen: Display Output Device

State Invariant

none

Assumptions

It is assumed that the url used to fetch the data is always online.

Access Routine Semantics

startProgram():

- transition: Updates the $courseIds$ to have the course ids of all courses and $allCourses$ to have all the information (such as Prof name, room, courseId etc) of all the courses.
- output: *none*
- exception: none

10 MIS's of checkCourse.js

Template Module

course

Uses

app.js

Syntax

Exported Types

router = ?

Exported Access Programs

Routine name	In	Out	Exceptions
submit	Request,Response,Next	string	-
check	Array of strings	Boolean	-

Semantics

State Variables

None

Environment Variables

Screen: Display Output Device
Keyboard: Input Device

State Invariant

none

Assumptions

None

Access Routine Semantics

submit($req, res, next$):

- transition: none
- output: Outputs a string message to the user's browser if there were any problems with user's input, otherwise it will return null.
- exception: none

check($selectedCourses$):

- transition: none
- output: Outputs a Boolean value, true if there were no problems with user's input, and false if there was.
- exception: none

11 MIS's of index.js

Template Module

course

Uses

app.js checkCourse.js scheduler.js

Syntax

Exported Types

router = ?

Exported Access Programs

Routine name	In	Out	Exceptions
welcomePage	Request,Response,Next	EJS file	-
giveLink	Request,Response,Next	EJS file	-

Semantics

State Variables

None

Environment Variables

Screen: Display Output Device
Keyboard: Input Device

State Invariant

none

Assumptions

None

Access Routine Semantics

welcomePage($req, res, next$):

- transition: none
- output: Sends an ejs template to the user's browser
- exception: none

giveLink($req, res, next$):

- transition: none
- output: Sends an ejs template to the user's browser
- exception: none

12 MIS's of scheduler.js

Template Module

course

Uses

Course.js
checkCourse.js
app.js

Syntax

Exported Types

router = ?
reset = ?

Exported Access Programs

Routine name	In	Out	Exceptions
start	req, res,next	EJS file	-
getSchedule	req, res, next	-	-
algorithm	-	Boolean	-
permutation	int	int	-
updateSemester	List	List	-
dobothSemester	List, List, List	List	-
MadeBefore	List, List	Boolean	-
Conflicts	int, int, int	Boolean	-
prioritize	List	List	-
doSemester	List	Boolean	-
makeGraph	-	Boolean	-
getConnectedComponent	Graph	List	-
reserveTime	int, int, int	-	-
avoidConflicts	int, int, int, int	-	-
hasConflict	int, int, int, int	Boolean	-
hasPathConflict	int, int, int, int	Boolean	-
BreadthFirstPaths	Graph, int	-	-
bfs	Graph , int	-	-
hasPathTo	int	Boolean	-
putInaDay	int, String	-	-

Semantics

State Variables

graph : Graph - a graph where each node represents a course and an edge represents a conflict-less relationship between courses

dataset : List of strings - array that represents the data-set

allCourses : List of strings - List of course codes all courses available in McMaster University

finalCourses :List of strings - List of course codes the user added

bothSemesters : List of strings - List of course codes of courses that are offered in both Semesters

*semester*1 : List of strings - List of course codes of the courses that are available only in semester 1

*semester*2 : List of strings - List of course codes of the courses that are available only in semester 2

fixedCores : List of strings - List of course codes of courses with one core(lab,lecture,tutorial). e.g) "SFWRENG 2XA3 C01"

flexCores : List of strings - List of course codes of courses with more than one core(lab,lecture,tutorial). e.g) "SFWRENG 2XA3 L01 , SFWRENG 2XA3 L02"

*finalSemester*1 : List of timeObjects - List of timeObjects that for final semester 1 schedule

*finalSemester*2 : List of timeObjects - List of timeObjects that for final semester 2 schedule

*day*1 : List Course Objects - List of course objects of courses that are on Monday of the final schedule.

*day*2 : List of Course Objects - List of course objects of courses that are on Tuesday of the final schedule.

*day*3 : List of Course Objects - List of course objects of courses that are on Wednesday of the final schedule.

*day*4 : List of Course Objects - List of course objects of courses that are on Thursday of the final schedule.

*day*5 : List of Course Objects - List of course objects of courses that are on Friday of the final schedule.

*day*6 : List of Course Objects - List of course objects of courses that are on Saturday of the final schedule.

times : 2D Matrix of double - The values in this array represents the times reserved by courses. This is used to check for conflicts between courses.

success : Boolean - The value of this variable represents the success of the program. True means a schedule was generated, false means there was an error.

schedule : List of strings - List of course objects of fixed cores that are in the final semester 1 schedule.

*schedule*2 : List of strings - List of course objects of fixed cores that are in the final semester 2 schedule.

madeArrays : 2D Matrix of integers - This array keeps track of different versions of arrays that were made by shuffling its elements.

$conflictCourses$: List of strings - List of course codes of courses that conflicts with each other.

$conflictAvoider$: List of double - Each element in this array represents the times be-

tween and including the starting and ending time of a lecture/lab/tutorial.

allTimeObjects : List of timeObjects - This array contains the time objects of all cores(lab,lecture, tutorial) of all the courses the user selected.

marked : List of Boolean - This array is used to get the BreadFirstPath in a graph

edgeTo : List of real - This array is used to get the BreadFirstPath in a graph

distTo : List of real - This array is used to get the BreadFirstPath in a graph

q :List of real - This array is used to get the BreadFirstPath in a graph

Environment Variables

Screen: Display Output Device

State Invariant

none

Assumptions

None

Access Routine Semantics

start($req, res, next$):

- This routine starts the calculations for generating a schedule by calling other routines and initializing some global variables.
- transition: $schedule, times, day1, day2, day3, day4, day5, day6,$ $semester1, semester2, bothSemesters, fixedCores, flexCores, finalSemester1, finalSemester2$:= [ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ],[ ]
- output: $EJSfile$
- exception: none

getschedule(req,res,next)

- This routine sends the generated schedule to the user's browser.
- transition: none
- output: $EJSfile$
- exception: none

algorithm()

- This routine finds the courses that are offered only in semester1, only in semester 2 and in both semesters, and then they are put in proper arrays.
- transition: $semester1, semester2, bothSemesters, sucess :=$ {string, string ...},{string, string ...},{string, string ...}, boolean
- output: $EJSfile$
- exception: none

permutation(number)

- This routine finds the total permutation by calculating the factorial of a given number.
- transition: none
- output: *int* - Outputs an integer that is the factorial of the input *number*
- exception: none

updateSemester(semester)

- This routine creates a duplicate of an array and outputs it.
- transition: none
- output: - Outputs a duplicate of the given array (*semester*)in the input.
- exception: none

dobothSemester(bothSemesters,semester1,semester2)

- This routine re-orders the courses in *bothSemester* and tries to put them in $semester1$ and $semester2$.Then, it calls the routine $doSemester(semester)$ twice, one time with $semester1$ and the other time with $semester2$. If a working schedule is found, it will return an array of bothSemesters that are arranged in a specific way which will generate a schedule, when they are put in semester1 and semester2.

- transition: none
- output: - Outputs an empty array if there is no way to make a schedule. If it is possible to make a schedule, it will output an array with courses in bothSemester arranged in a specific way.
- exception: none

MadeBefore(madeArrays,arr)

- This routine checks if *madeArrays* have the array *arr*.
- transition: none
- output: - Outputs a boolean value, true if the *arr* is in *madeArrays* and false if it is not.
- exception: none

Conflicts(startTime,endTime,day)

- This routine checks if the array , in the state variable *times* , at index d have any values between and including, startTime & endTime. If it does, it return true and if it doesn't then it returns false.
- transition: none
- output: - Outputs a Boolean value, true if any values between and including starting & endTime are in the array in *times* at index *day*.
- exception: none

prioritize(semester)

- This routine reorders the array *semester* based upon how many lectures,tutorials and labs a course has.If a course in *semester* has the most number of lectures,tutorials and labs, then it will be placed in the beginning of the array.
- transition: none
- output: - Outputs a new array that has the courses in *semester* but in a different order.
- exception: none

doSemester(semester)

- This routine updates the state variables $fixedCores$ and $flexCores$ by putting the courses that offer only one lecture or one tutorial or one lab in $fixedCores$. It puts the cores($Example\ of\ core\ :\ PHYSICS\ 1e03\ c01,\ chem\ 2e03\ T01$) that offer more than one lecture or more than one tutorial or more than one lab in $flexCores$. It also tries to find if there is any conflict between courses in $fixedCores$

- transition: $fixedCores, flexCores$:= {string,string...},{string,string...}

- output: - Outputs a Boolean, true if there were no conflicts between cores in $fixedCores$ and false if there is.

- exception: none

makeGraph()

- This routine makes a graph where each node is a course and each edge represents a conflict-less relationship between two courses.

- transition: none

- output: - Outputs a Boolean, true if it is possible to make a graph with cores from all courses.

- exception: none

getConnectedComponent(graph)

- This routine finds all the nodes that are connected to at least one other node. The cores that are not connected are ignored.

- transition: none

- output: - Outputs an array with all the cores that are connected to at least one other core.

- exception: none

reserveTime(startTime,endTime,day)

- This routine puts all the values between(and including) $startTime$ and $endTime$ in an array that is at the index day of the state variable $times$

- transition: $times$:= {{double,double,double},{double,double,double},{double,double,double}, {double,double,double},{double,double,double},{double,double,double},{double,double,double}}

- output: - none

- exception: none

avoidConflicts(startTime,endTime,day,courseIndex)

- transition: This routine puts all the times between (and including) $startTime$ and $endTime$ at $conflictAvoider[courseIndex][day]$.
- output: none
- exception: none

hasConflict(startTime,endTime,day,courseId)

- transition: none
- output: Ouputs a Boolean, true if the state variable $conflictAvoider$ has any values between (and including) $startTime$ and $endTime$ at $conflictAvoider[courseIndex][day], andfalseifitdoe$
- exception: none

hasPathConflict(coreId,day, start,end)

- transition: none
- output: Outputs a Boolean, true if there is a conflict between any courses that are in path between node 0 and the node at coreId and false if it is not.
- exception: none

BreadthFirstPaths(graph, start)

- transition: Updates the values of state variables $marked$, $edgeTo$, $distTO$ based upon the graph algorithm. This routine is a part of the graph algorithm.
- output: none
- exception: none

bfs(G, s)

- transition: Updates the values of state variables $marked$, $edgeTo$, $distTO$ based upon the graph algorithm. This routine is a part of the graph algorithm.
- output: none
- exception: none

hasPathTo(v)

- transition: none
- output: Outputs a boolean, true if there is a path to specified node from node 0 and false if there is no path. This routine is a part of the graph algorithm.
- exception: none

putInaDay(courseDay, course)

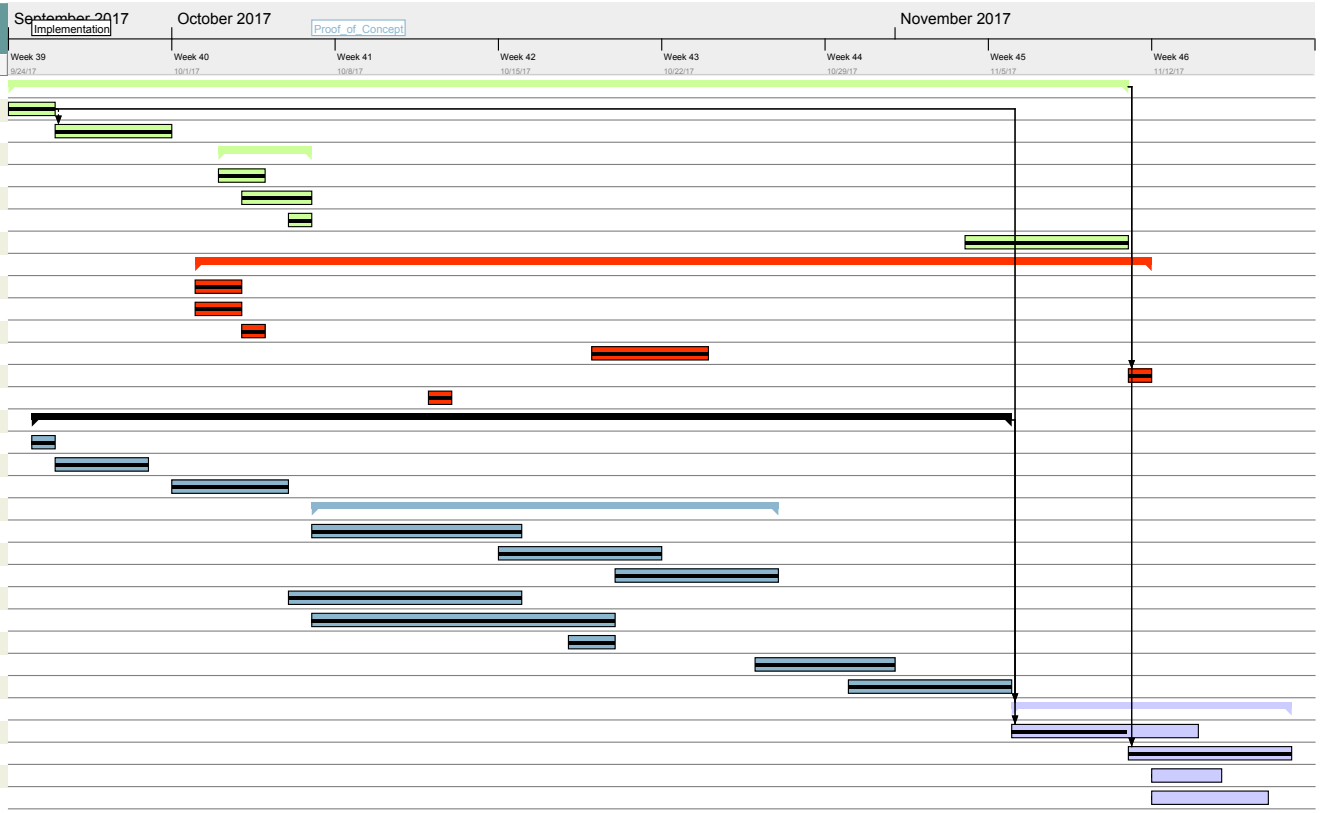
- transition: Depending on the value of *courseDay*, it will update either the state variable day1 or day2 or day3 or day4 or day5 or day6 , by pushing the value of course into one of the arrays that was mentioned before.
- output: none
- exception: none

13 Schedule

Genzter Project Schedule

Gantt Chart

Name	Begin date	End date
Documentation	9/24/17	11/10/17
Problem Statement	9/24/17	9/25/17
Development Plan	9/26/17	9/30/17
SRS	10/3/17	10/6/17
Functional Requirements	10/3/17	10/4/17
Non-Functional requirements	10/4/17	10/6/17
Other	10/6/17	10/6/17
Design Document	11/4/17	11/10/17
Issues	10/2/17	11/11/17
Sessions are not isolated	10/2/17	10/3/17
Cannot remove added courses	10/2/17	10/3/17
Error when attempting to remove course	10/4/17	10/4/17
Course conlicts not detected	10/19/17	10/23/17
Error messages do not fit the screen	11/11/17	11/11/17
Error message color sheme mismatch	10/12/17	10/12/17
Implementation	9/25/17	11/5/17
Node Project Initialization	9/25/17	9/25/17
Basic Web Server	9/26/17	9/29/17
Node Session Management	10/1/17	10/5/17
Scheduling Algorithm	10/7/17	10/26/17
Semester 1 Functionality	10/7/17	10/15/17
Semester 2 Functionality	10/15/17	10/21/17
Full Year Functionality	10/20/17	10/26/17
HTML Templates	10/6/17	10/15/17
Proof_of_Concept	10/7/17	10/19/17
PoC Testing	10/18/17	10/19/17
Drag and Drop feature	10/26/17	10/31/17
Error Messages	10/30/17	11/5/17
Testing	11/6/17	11/17/17
Structural Testing	11/6/17	11/13/17
Create Test Plan	11/11/17	11/17/17
Functional Testing	11/12/17	11/14/17
Parallel Testing	11/12/17	11/16/17



References

[1] On the Criteria To Be Used in Decomposing Systems into Modules *D.L. Parnas* https://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf