Project Description

(*Use additional pages as needed for this report.*)

PART I.

1. General Information

Project Title: Student Self-Counseling
Submitted by (insert names of team members): Tomasz Andres, Liangliang Xu
Submitted to (supervisor name): Imran Ahmad
Date submitted: 03/18/2020

2. Project Overview (*Describe the project & its purpose*)

This project will be focused around giving students the option to have a robust and productive self-counseling experience.

The project shall allow for students to access a web interface to view which courses they need to complete to finish their degrees.

Currently, the replacement for the Student Information System has not been very accurate, and this system will resolve that issue.

3. Deliverables (*Describe all products to be produced*)

There is one deliverable, and it is the web interface that shall be able to do the following things:

a) Allow students to select their program name from a list of options.
b) Show a list of required courses for a specific program; students may click on each course to show that it was *taken*
c) Courses that are a part of a selected program shall be highlighted.
d) A list of *remaining* courses shall be displayed; a suggested order will be given to the student, based on pre-requisite status and course availability
e) The student will have the option to load or save his/her course list in order to make it easy to access in the future.

Please note that items D and E will likely be implemented but are subject to discussion with the project instructor.

4. Requirements (*Describe the required resources e.g. hw/sw, technical knowledge and skills etc*)

We will need to learn about the following (technical knowledge):

1. Back-end web design, likely with PHP or NodeJS
2. Front-end web design with JS, HTML, and CSS
3. MySQL or NoSQL database design
4. Using MySQL or NoSQL with the back-end language
5. Interfacing the back-end with the front-end via AJAX
6. How to store course data in the database

7. What course data is available and what is the fastest way to parse it

Our skills will need to include:

8. Good teamwork
9. Timely submission
10. Ability to test the project to ensure that there are no bugs
11. Good communication
12. Ability to code at a good pace
13. Ability to share code in an effective manner to make progress
14. Ability to design a proper interface that is visually stunning

Hardware required will be:

15. A server to host the website; likely the UWindsor servers
16. Our computers for web development and testing

Software required will be:

17. Nginx for back-end hosting
18. PHP or NodeJS for back-end language
19. Github for file sharing and progress tracking
20. Visual Studio Code or Sublime Text for text editing

5. Constraints

PART II.

1. Risk analysis *(There is no fixed format, as it will vary depending on the nature of the project. For some general guidelines please see the file under the* ***Resources→Risks*** *link on BB)*

Project risks include:

1. Project files will become corrupted when stored on GitHub or on our computers. The cost of this could be catastrophic, since it makes it so all our files are *gone, reduced to atoms*; monetarily, it would be a big hit for the company. The likelihood of this happening is very low since files don't tend to corrupt very often, especially on the largest file hosting service in the world, GitHub.
2. We will set unrealistic schedules in order to get things going way too quickly without doing appropriate planning. The cost of this could involve developer frustration or lack of enthusiasm. The likelihood of this is low because we know that we should produce a proper schedule to get things going at a correct pace.
3. We will set unrealistic expectations which would potentially hinder our ability to proceed if a system fails to work. The cost of this could, again, involve developer frustration or lack of enthusiasm. The likelihood of this is medium because we don't have too much experience with this type of project and we're not aware of how long specific features may take to implement.
4. The database will not be able to store the required course data. The cost of this will be a major setback, and it would cost time, and we know that *time is money*. The likelihood of this taking place is very low since databases are very robust.

5. The interface design will look poor because we did not spend enough time on the graphical aspect. The cost of this will be that nobody will want to use the product and therefore the whole project would be a waste. The likelihood of this taking place is high because we don't have too much experience with interface design.
6. The system will have bugs, such as long loading times, bad course data being displayed, or 404 pages. The cost of this will be user frustration, which will cause people to not want to use the product. The likelihood of this happening is low because we will be performing adequate testing.

Risk mitigation strategies for the above include:

1. Storing backups in case of data corruption or in case GitHub goes down. An alternative to GitHub is either self-hosted git or simple file-sharing back-and-forth. GitHub is a better solution because it is easier to use and quicker to get set up; however, a self-hosted solution may provide the ability for lower latency.
2. We will gauge schedules based on our past experience and ensure that we are realistic with our timeframes for each milestone. An alternative to this is to get assistance with schedule design from an expert in the field. If we ask for an expert's assistance, the downside is that it may cost money, while the upside of doing it ourself is that we learn more through research.
3. Our expectations will be set based on what we see is possible with currently web-based technology. If we are unable to meet an expectation in a certain way, we will find other ways to meet the expectation; otherwise, we will tweak the expectation and provide an explanation for why we were unable to reach the goal using the technology that we have. An alternative to this is that we ask for an expert for assistance; and the upside is the same as before: we get an expert's help but the downside is potential cost; therefore it is better to do it ourselves.
4. We will experiment with JSON files, as well as databases, in cases databases are not the right choice for this project. An alternative to JSON is storing data in comma-separated values (CSV) files. The upside of CSV is that it is easy to read via a program such as Excel but it is more bulky to store than JSON or SQL.
5. We will look at other websites and take great inspiration on how designers do their work. We will likely ask our colleagues to review our interface in order to ensure that it is clean and easy-to-use. An alternative to taking inspiration is to read a book on how to design. The downside is that we would take a long time to read the book but the upside is that we would learn a lot about interface design. Searching Google is a lot better solution, though, because it is faster than reading a book.
6. We will squash bugs by performing stringent testing on the product and we will ensure that all links go to the proper place. An alternative to finding bugs, ourselves, is that we ask our peers test the software. The upside is that our peers may find bugs that we would've otherwise not found; however, the downside is that we have to find someone who has the time to do it. It's a great idea to ask our peers to be beta testers, though.

PART III.

1. Work Breakdown Structure (WBS) *(There is no fixed format, as it will vary depending on the nature of the project. For some general guidelines please see the file under the* ***Resources → Work Breakdown Structure (WBS)*** *link on BB)*

Goal: Develop a robust web interface and a back-end for students to be able to check on the progress of their degree and see which courses they need to graduate.

Deliverables:

1. Front-end web interface design.
2. Web interface, itself.
3. Back-end server-side.
4. Database system.
5. Testing and final review.

Modules:

1. Front-end web interface design
   Estimate: 2 weeks
   a. Nice-looking theme via CSS.
2. Web interface, itself.
   Estimate: 4 weeks
   a. Main page that has a brief description of what the site does and has course selection interface
   b. JS functionality when buttons are clicked in order to send an AJAX request to the server for information
3. Back-end server-side.
   Estimate: 8 weeks
   a. Getting NGINX set up for hosting.
   b. Setting up PHP or NodeJS.
   c. Setting up an API for the front-end AJAX requests.
4. Testing and final review.
   Estimate: 2 weeks
   a. Test product for bugs and squash bugs
   b. Ask for peer-review

Tasks:

5. Nice-looking theme via CSS
   a. Looking up how to code CSS via Google and YouTube
   b. Looking up proper design concepts and user interface cleanliness ideology
6. Main page that has a brief description of what the site does and has course selection interface
   a. Ability to click on a course to *select* it
   b. Ability to view courses that are required for graduation
   c. Ability to choose a program
   d. Ability to load existing courses data
   e. Ability to save courses data
7. JS functionality when buttons are clicked in order to send an AJAX request to the server for information
   a. AJAX handler for sending courses list.
   b. AJAX handler for getting courses list back.
   c. AJAX handler for saving.
   d. AJAX handler for loading.
8. Getting NGINX set up for hosting.
   a. Looking into how to set up NGINX on Google and YouTube

    b. Setting up NGINX on the hosting machine, likely a UWindsor server
    c. Configuration nginx.conf file (or whatever config is available) to direct traffic to the appropriate folder
    d. Setting up the website's work folder and the *public_html* or *public* folder
9. Setting up PHP or NodeJS
    a. Downloading PHP/NodeJS
    b. Installing PHP/NodeJS
    c. Hooking up PHP/NodeJS to NGINX in the configuration file
    d. Configuration NGINX/NodeJS if any configuration is necessary
10. Setting up an API for the front-end AJAX requests.
    a. Function to give course information
    b. Function to get courses list from given courses that are provided
    c. Function that saves courses list for a particular student via storage system like SQL
    d. Function that loads courses list for a particular student via storage system like SQL
11. Test product for bugs and squash bugs
    a. Test each feature for bugs.
    b. Visit all pages and ensure no 404 pages are present.
    c. See if there are any script errors in the front-end console.
    d. See if there are any script errors in the back-end console/log.
    e. Resolve any issues that were found in this stage.
12. Ask for peer-review
    a. Ask a student friend to test out the product and get their opinion about it and ask them if they found any issues/bugs

For all tasks, we will handle them in order of priority to the immediate feature.
For all activities, we will handle them in numerical order from 1-9.

Work will be separated on a 50/50 basis with each project partner in order to work efficiently and effectively.
We will check in with each-other and work together strongly in order to ensure that we can get the project completed ahead of schedule, if possible.