High-Level Requirements Document

Brute Force (Group 14)

Members:

Yiyang Hou	(yiyangh@usc.edu)
Sean Syed	(seansyed@usc.edu)
Eric Duguay	(eduguay@usc.edu)
Xing Gao	(gaoxing@usc.edu)
Sangjun Lee	(sangjun@usc.edu)
CP: Aya Shimizu	(ashimizu@usc.edu)

Objective

The main goal of our project is to create an web application which is able to parse the USC Course Catalog and allow the user to select preferred classes for the semester. A user should be able to specify the times within a given day that they would either like to attend class or have a break. Users will be able to register based on a predetermined priority scale, and will be given a schedule based on courses in which there is space available for them at the time of their respective registration.

Motivation

Organizing class schedules with the USC's current scheduling system is broken. It is strenuous, time-consuming, and inefficient. The system doesn't automatically generate or filter classes by students' preferred time frames and has an abysmal search function. By only showing classes that meets the students' expectations, students will spend less time on browsing through irrelevant classes or trying to mix-and-match their schedules.

Core Functionality

Adding Core Classes:
The bulk of the application will deal with the user's class preferences. The user will be able to choose their major and year and have classes required for that major fill in automatically. From this point, the user will be able to choose other classes he or she wishes to take and times during which they wish to be in class. From this data, the web application will then generate suggested schedules for the user.

Class Priority:
When choosing classes, the user may wish to take multiple classes, but have a preference of one class over another. The user will be able to assign a priority to the classes and then use these priority values to provide an optimally generated schedule.

Suggesting Classes that Fit:
Some users may have all of their core major-related classes chosen, but may need to fill some other requirement like a GE or Writing course. If the user is not picky about what this course is, or cannot be picky due to the current order of their schedule, then the application will allow them to choose a category (e.g. History GE) and offer suggestions to the user about possible courses which would fit into that space.

Challenges

Getting Class Data:
As of right now, we are unsure if we will be able to get a database of classes directly from the school or if this information would need to be scraped from the website. If the latter is true, the initial time investment for this goes up dramatically as one or more members must be tasked with writing a script in order to get this information from the Internet.

Authentication with an Actual API:
Before accessing all user components of our system, the student is required to login into our system by providing an existing username and matched password. If the username is non-existing in our database, or the password does not match with the username, the student will be asked to provide his correct validation until he inputs a pair of valid username and password. Since most universities prefer to use Shibboleth as their authentication solution, we would like to build our own Shibboleth login in system first to support future extensions. However, most frameworks work better with OAuth 2 solutions better than SAML 2 which is used for Shibboleth. Our second choice will be building an OAuth 2 authentication by using the framework skeleton.

Efficient Schedule Algorithm:
The main draw of the application would be the allure of pain free scheduling, and thus, a significant portion of time will be spent on perfecting the scheduling algorithm in order to ensure efficient and accurate results for the user.

Responsive Design:
Displaying a large amount of data on a mobile device would be challenging, especially when the data is presented in a specific format: a schedule with fixed number of rows and columns. Therefore, meticulous testings are essential to preventing design conflicts in mobile devices.

Stretch Goals

Course Rating System:
After a given grading term, students can rate the course that they took, as well as the professor that teaches it. This would allow prospective students to gauge whether or not they would enjoy taking the course. An issue with this sort of functionality is the fact that it may create excessive bias towards particular courses. A proposed solution would be asymmetric viewing permissions between students and professors in which professors would not be able to view the ratings. However, student submissions would not be anonymous in order to keep students accountable for what they post to the site.

Create a Forum for People to Switch Course Sections:
Since a number of students will inevitably end up with a class schedule that they do not particularly prefer, they can access a forum for people to discuss course section trading so that

such disputes may be resolved internally. The functionality of this forum would also allow students to swap courses straight from the forum, itself, since it is part of our internal code base.

Push Notifications for Full Classes:
Once subscribed to a push notification, students will get notified for all or selected activities occurred on particular classes. Thereby, the students won't have to check frequently whether their desired classes have remaining seats or not.

Google Maps Tracking Walking Time:
Students who are concerned about the time that it takes for them to travel from one class to another will be able to see this directly from the application so that they may be able to take this information into consideration before finalizing their schedule. Since the

Add Clubs to the Program:
Clubs can register with the application in order to incorporate their schedule of room reservations so that members can add these times to their course schedule. Club leaders can send a roster of members authorized to schedule themselves in the section.

Suggest Future Course Based on Previously Taken:
This would require that the user have a class history stored somewhere in the system. This class history would be analyzed against other users' history in order to suggest a course which they may like to take.

Technical Specifications Document

Brute Force (Group 14)

Members:

Yiyang Hou	(yiyangh@usc.edu)
Sean Syed	(seansyed@usc.edu)
Eric Duguay	(eduguay@usc.edu)
Xing Gao	(gaoxing@usc.edu)
Sangjun Lee	(sangjun@usc.edu)
CP: Aya Shimizu	(ashimizu@usc.edu)

Web Interface (8 hours)

- The initial page should include a login form where the user will put a username and a password.
- Upon authentication from the server, a new page will load and display a table with two columns —columns for class code and time frame—where the user can type in classes and time frame by preference.
- After the user fills out the table, the user may click on a button which will call a function that will pass the user input to the server.
- The server will respond with an object that contains optimized schedule if any.
- The page reloads and displays the result: a new schedule received from the server or a message that notifies the user that the input could not generate any schedule.

Server (16 hours)

- The server must handle data from the client side that contain class codes and time frames.
- The server must contain "Schedule Optimization Algorithm" to handle the data.
- The server must connect to a database that contains schedules of classes.
- After the server receives data from the client side, it must check if the data contains proper information: class codes and time frames.
- Once the data are verified, call a method that contains "Schedule Optimization Algorithm."

Schedule Optimization Algorithm (8 hours)

- Upon receiving refined data from the server's call, the algorithm method must access the database and retrieve data from the table which is organized according to the user's preferences.
- Some may be done server side, but there may be some speed optimization that could happen on the front-end through the caching of possible choices.

Database (8 hours)

- The initial schema for the database must be created, optimizing for quick access by the optimization algorithm.
- We must write a script to pull the information publicly available on the course catalog website.
- We must write a script in order to populate the database given the information taken from the website.

Detailed Design Specification Document
Brute Force (Group 14)

Members:

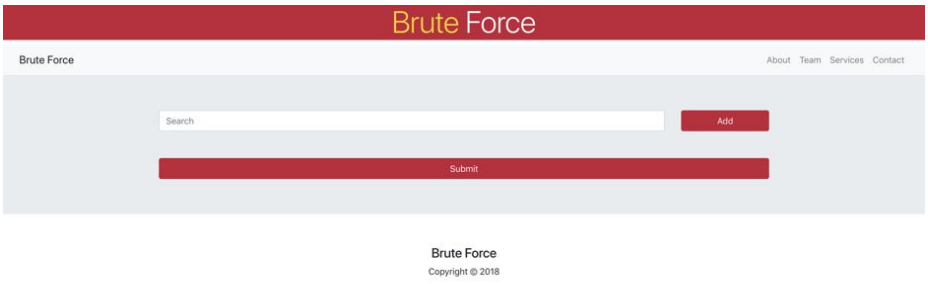
Yiyang Hou	(yiyangh@usc.edu)
Sean Syed	(seansyed@usc.edu)
Eric Duguay	(eduguay@usc.edu)
Xing Gao	(gaoxing@usc.edu)
Sangjun Lee	(sangjun@usc.edu)
CP: Aya Shimizu	(ashimizu@usc.edu)

1. Detailed specifications
    a. Difficult algorithms - Pick the Best Choice
        i. Retrieve data from Database
        ii. If nothing has been returned, then display “Your selection is not available. Please modify your search.”
        iii. Else compare the time interval between the earliest start time and the latest end time for each day of each combination. Store the largest time interval among the five days.
        iv. Compare all the largest time intervals and pick the minimum one.
        v. Display the result with the minimum time interval as the “Recommendation”.
        vi. Display other results below the “Recommendation”.

    b. GUI
        i. ***Image: login page***

        ii. ***Login page: Login***
            1. Once the user types in email and password and clicks the login button, the user will send the account information to the server for authentication. The server will respond with login status whether it was successful or not.
            2. Upon authentication, the page will be linked to the home page.

        iii. ***Image: home page***



        iv. ***Home page: Searching***
            1. As the user types in class code (e.g. CSCI, PHYS, and etc.), javascript will listen to any text input in the search box.

2. The text input listener will send requests to the server with the keyword, whenever it detects any changes in text input.
3. The server will access the database and query for class codes that have the keyword as their prefixes.
4. The server will transform the query result into JSON and send it back to the client.
5. On the client side, the search box will display suggestions if there was any.

v. ***Image: result page***

vi. ***Result page: Result***

1. Once the user successfully added the classes upon suggestions, the list-box will be created with the name of the class.
2. After adding all the classes, the user can submit the form.

vii. ***Image: submit page***

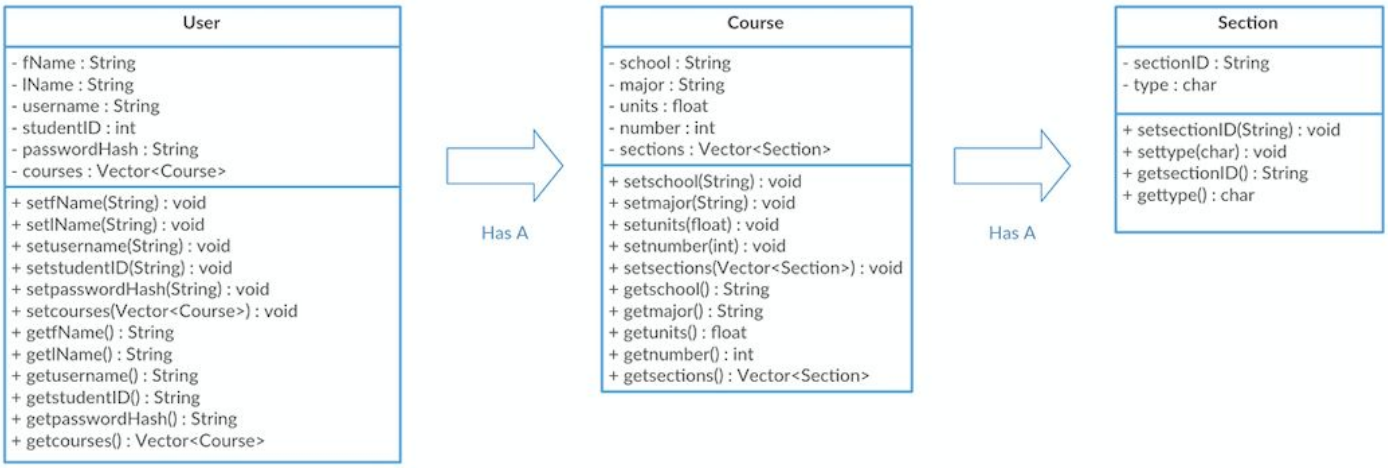
viii. ***Submit page: Submission***

1. Upon submission, the server will add the classes to the user's database.

c. Database schema (ER diagram)

d. Hardware/Software requirements
   i. Hardware Requirements
      1. Ethernet/Wireless LAN Adapter
   ii. Software Requirements
      1. Browser support HTML5
      2. Enable Javascript in your Browser
      3. Enable Flash in your Browser

e. Class diagram



Testing Document
Brute Force (Group 14)

Members:

Yiyang Hou	(yiyangh@usc.edu)
Sean Syed	(seansyed@usc.edu)
Eric Duguay	(eduguay@usc.edu)
Xing Gao	(gaoxing@usc.edu)
Sangjun Lee	(sangjun@usc.edu)
CP: Aya Shimizu	(ashimizu@usc.edu)

Use Case #01: Login System				
Test Run Information: Tester Name: Xing Gao Planned Date(s) of Test: 11/05/2018				
Objective: The user submits his/her existing username with a matchable password to visit the course selection page.				
Input Data: Username; Password; Submit				
Output Data: Next page				
Pre-Conditions: The application must build a connection with database; AND database must contain a user table storing username and password.				
Post-Conditions: None				
Test Case	**Create Dates**	**Testing Strategies**	**Expected Result(s)**	**Actual Result(s)**
01-01: Input an existing username in the username input field; input a password that matches the username in the password input field; submit the username and password input.	11/01/2018	White Box Test	The user will be taken to the course selection page.	The user will be taken to the course selection page.
01-02: Input an existing username in the username input field; input a password that does not match the username in the password input field; submit the username and password input.	11/01/2018	White Box Test	The user will be taken to the login page with a message “Invalid Login”.	The user will be taken to the login page with a message “Invalid Login”.
01-03: Input a non-existing username in the username input field; input a password in the password input field; submit the username and password input.	11/01/2018	White Box Test	The user will be taken to the login page with a message “Invalid Login”.	The user will be taken to the login page with a message “Invalid Login”.

Use Case #02: User Search System

Test Run Information:
Tester Name: Sangjun Lee
Planned Date(s) of Test: 11/05/2018

Objective:
As the user types in a search keyword, the server should respond with suggestions of course codes.

Input Data:
Search keywords

Output Data:
Suggestions with course codes (e.g. CSCI-201, PHYS-151, and etc.)

Pre-Conditions:
The application must build a connection with database; AND database must contain classes with class codes.

Post-Conditions:
None

Test Case	Create Dates	Testing Strategies	Expected Result(s)	Actual Result(s)
02-01: Input keywords to search for class codes	11/02/2018	White Box Test	The suggestion box should only contain courses with keywords as their prefixes	The suggestion box should only contain courses with keywords as their prefixes
02-02: Input keywords that is not a prefix of any class codes	11/02/2018	White Box Test	The suggestion box should contain a message “No results”	The suggestion box should contain a message “No results”
02-03: Click one of suggestions from the suggestion box	11/02/2018	White Box Test	A new list-item of the corresponding class code must be added to user’s class submission table.	A new list-item of the corresponding class code must be added to user’s class submission table.

Use Case #03: Database Reading and Writing 1/2

Test Run Information:
Tester Name: Sean Syed
Planned Date(s) of Test: 11/05/2018

Objective:
The server should be able to read and write user data and update class information within the database.

Input Data:

Front-end update information or get requests regarding user schedules or course information
Output Data: Either confirmation that the database has been updated or the appropriate information that has been requested in the form of a JSON object
Pre-Conditions: The application must build a connection with database; AND database must have infrastructure to store course and user information.
Post-Conditions: None

Test Case	Create Dates	Testing Strategies	Expected Result(s)	Actual Result(s)
03-01: Overwrite a user's schedule to the database	11/04/2018	White Box Test	A subsequent pull of that user's information should reflect the updated class schedule.	A subsequent pull of that user's information should reflect the updated class schedule.
03-02: Retrieve a user's information from the database	11/04/2018	White Box Test	A JSON file titled as the user's unique identifier should be written with all necessary member variables.	A JSON file titled as the user's unique identifier should be written with all necessary member variables.
03-03: Update the number of seats remaining in a section	11/04/2018	White Box Test	A subsequent pull of that section's information should reflect the new change in capacity.	A subsequent pull of that section's information should reflect the new change in capacity.

Use Case #04: Web Crawler Database Input
Test Run Information: Tester Name: Eric Duguay Planned Date(s) of Test: 11/10/2018
Objective: The web crawler should be able to scrape the course website and input into the database the list of classes
Input Data: Course information found at: https://classes.usc.edu/term-20191/
Output Data: A correctly filled and formatted database
Pre-Conditions:

The database infrastructure must be correctly formatted. The website must be up and up to date				
Post-Conditions: None				
Test Case	**Create Dates**	**Testing Strategies**	**Expected Result(s)**	**Actual Result(s)**
04-01: Add an overall course to the database	11/06/2018	White Box Test	The course section is added to the overall database	The course section is added to the overall database
04-02: Add a specific course to the database	11/07/2018	White Box Test	The specific instance of the course is added to the database	The specific instance of the course is added to the database
04-03: Adds a linked course to the database	11/08/2018	White Box Test	Links the lecture course to the other specific course instances of the discussion, lab and quiz sections	Links the lecture course to the other specific course instances of the discussion, lab and quiz sections

Deployment Document
Brute Force (Group 14)

Members:

Yiyang Hou	(yiyangh@usc.edu)
Sean Syed	(seansyed@usc.edu)
Eric Duguay	(eduguay@usc.edu)
Xing Gao	(gaoxing@usc.edu)
Sangjun Lee	(sangjun@usc.edu)
CP: Aya Shimizu	(ashimizu@usc.edu)

Environment Setup

1. Eclipse: download Eclipse from: https://www.eclipse.org/downloads/
2. MySQL: download MySQLfrom: https://www.mysql.com/downloads/
3. MySQL Workbench: download MySQL Workbench from: https://www.mysql.com/products/workbench/

Deployment

1. Launch MySQL Workbench and open and run the following script files respectively:
    a. SampleBuilding.sql
    b. SampleCourse.sql
    c. SampleLecture.sql
    d. SampleDiscussion.sql
    e. SampleLab.sql
    f. SampleQuiz.sql
2. Ensure all of the scripts were executed without any errors.
3. Launch Eclipse and open the project “BruteForce” in “csci201finalproject” folder.
4. Go to WebContent and open “login.jsp”
5. Run “login.jsp” on server at localhost on port 8080.
6. If there is any errors, make sure no other programs are using that port.
7. Upon successfully opening “login.jsp”, try to test registration functionality by clicking on “register” button.
8. Perform the following tests to ensure that all the functionalities are working:
    a. Register a new user
        i. Check if the database is updated
    b. Register an existing user
        i. Check if the system declines the registration of an existing user
    c. Login
        i. Check if the user can successfully login with a valid username and password

        ii. Check if the system declines logging in with an invalid username and password

9. Upon successfully logging into the system, check if “index.jsp” opens.
10. Perform the following tests to ensure that all the functionalities are working:
    a. Search course names on the search box
        i. Check if a suggestion box returns course names that contain keyword as prefixes
    b. Click an “add” button and see if a selected course is added to the course list.
    c. Check schedule
        i. Click a “check” button with a valid schedule and check if the system returns a success message
        ii. Click a “check” button with an invalid schedule and check if the system returns an error message
    d. Submit schedule
        i. Click a “submit” button and check if the database is updated.
11. Upon successfully submitting the schedule, click on “My Schedule” button.
12. Perform the following tests to ensure that all the functionalities are working:
    a. Check if “integrated_schedule.jsp” opens in a new window.
    b. Schedule box layout
        i. Check if the schedules are distributed into boxes properly by their start and end times.
    c. No schedule
        i. Check if the schedule doesn’t display anything if the user doesn’t have a schedule
13. Upon successfully submitting the schedule, click on “Download Schedule” button.
    a. Download
        i. Check if the button links to a download of an ics file.
        ii. Check if the file is successfully downloaded at a proper location.
    b. Email

i. Check if the receiver has received an email.

ii. Check if the email has been well formatted.

Review

After having gone through all the deployment setups and tests, check if the database is updated accordingly. If all the tests have fully passed without any errors, notify the users with the url: http://localhost:8080/BruteForce/login.jsp

Version track table

Version	Summary	Components Modified	Comment
1.0	Document created.	Added detailed description of: document description, table of contents; difficult algorithms, GUI, database schema, hardware/software requirements, class diagram; table to track versions and changes of this document	First draft of this document. Created on Oct 24th.
1.1	Modified Section	Added class hierarchy diagram	Oct 25th
1.11	Test Case 1 tested	Completed testing and added the results	Nov 1st
1.12	Test Case 2 tested		Nov 2nd
1.13	Test Case 3 tested		Nov 4th
1.14	Test Case 4 tested		Nov 6th
1.15			Nov 7th
1.2			Nov 8th