CSE 316 Fall 2023: Individual Web Programming Assignment 1

Due: Sept 25, 2023 at 11:59 PM

Read This Right Away

This problem set is due at **11:59 pm on Sept 25, 2023 KST**.

Directions

- At the top of every file you submit, include the following information in a comment
    - Your first and last name
    - Your Stony Brook email address
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
- You will be solving this with HTML, CSS, and Javascript only. You may use a source of graphics like bootstrap icons from bootstrap for the buttons (hamburger and close menu buttons) or you can get the images from another source.
- I will provide the background image to match the mockups shown.
- I will provide very basic templates for your index.html and css stylesheet file.

Tasks

This will be the first in a series of assignments that will progressively develop a full featured course registration application. This initial pass will only use HTML, CSS, bootstrap and Javascript. Since there is no backend, it will not have persistence between runs and will not actually store courses for which the student finally registers.

The application will have an initial page that only contains a navigation bar across the top (or in a compressed hamburger menu for small screens.) We will be learning to develop 'responsive' apps that adjust to display well on a range of screen sizes.

There are 4 pages total:

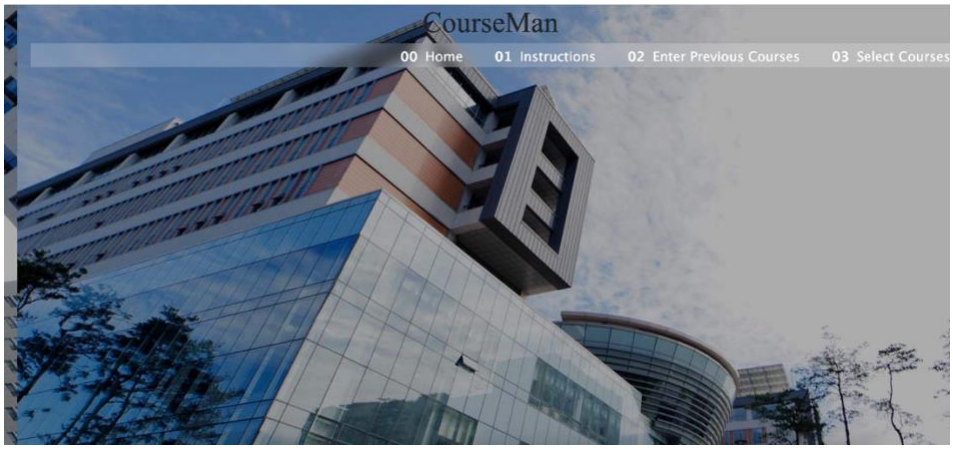
1. The home page with the navigation bar
2. An Instruction page that describes how to use the site to register for courses
3. A 'Previously Taken Courses' page that allows the user to enter what courses they have already passed (yes it is on the honor system for now!)
4. A course search/registration page

Mockups

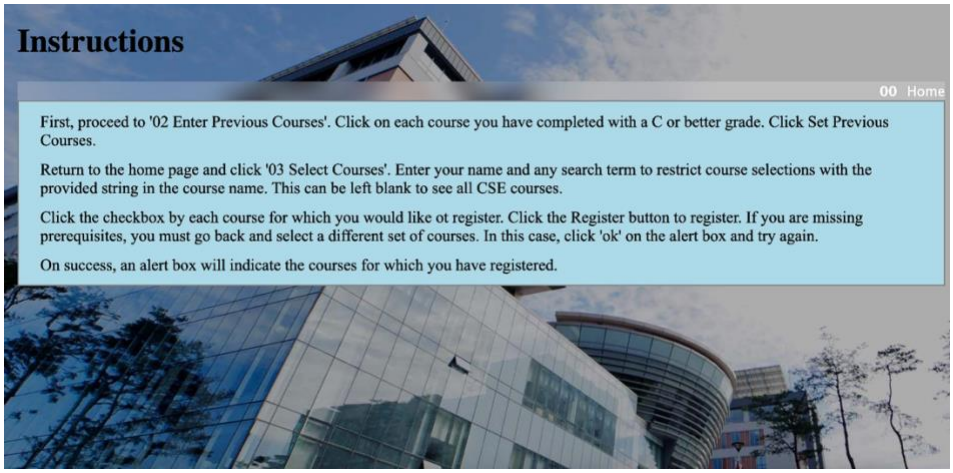
Following are mockups of the app's pages in large and small screen configurations. Implementation notes and details are given below after this Mockups section.

Large Screen

These mockups show screens larger than (or equal to) 40em in width

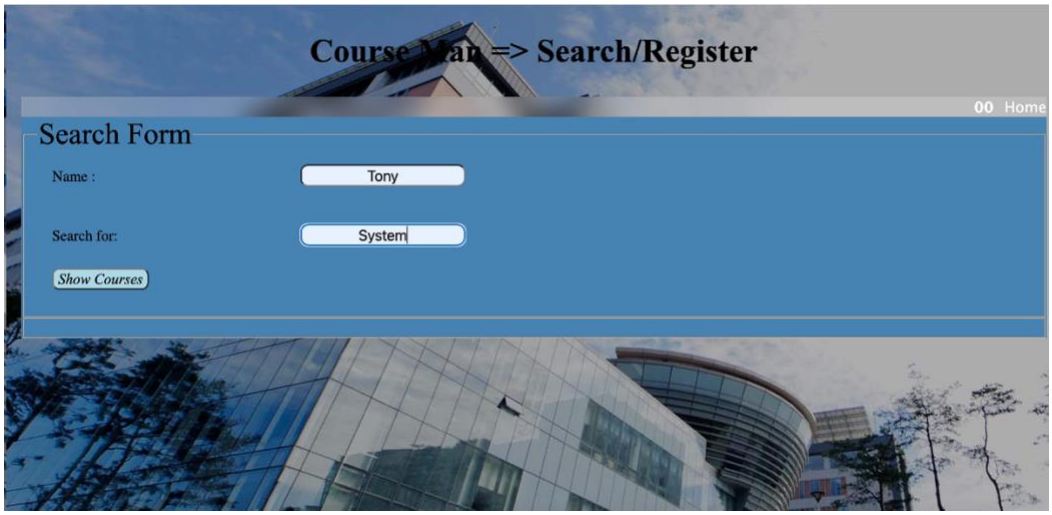


Large Home Screen

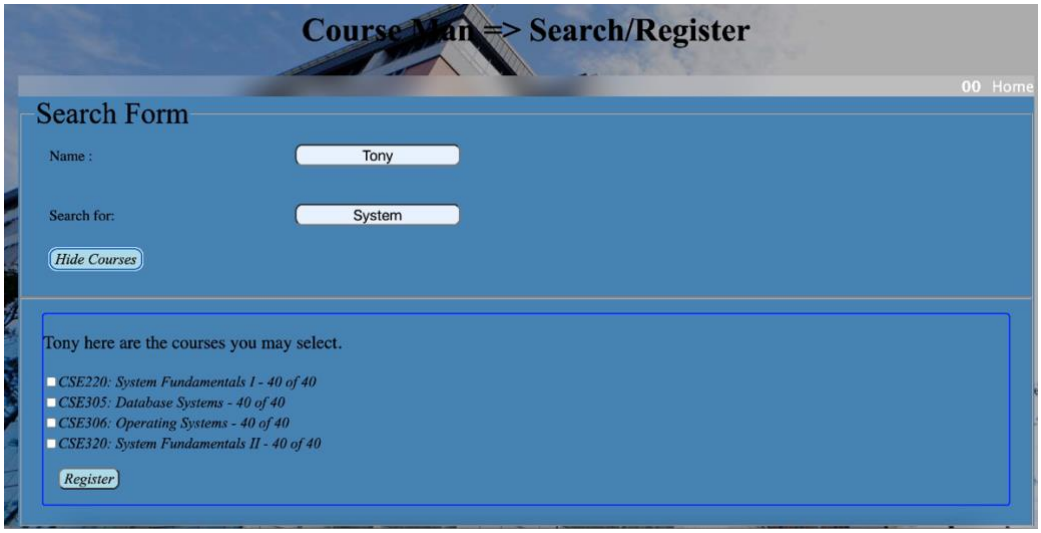


Large Instructions Screen

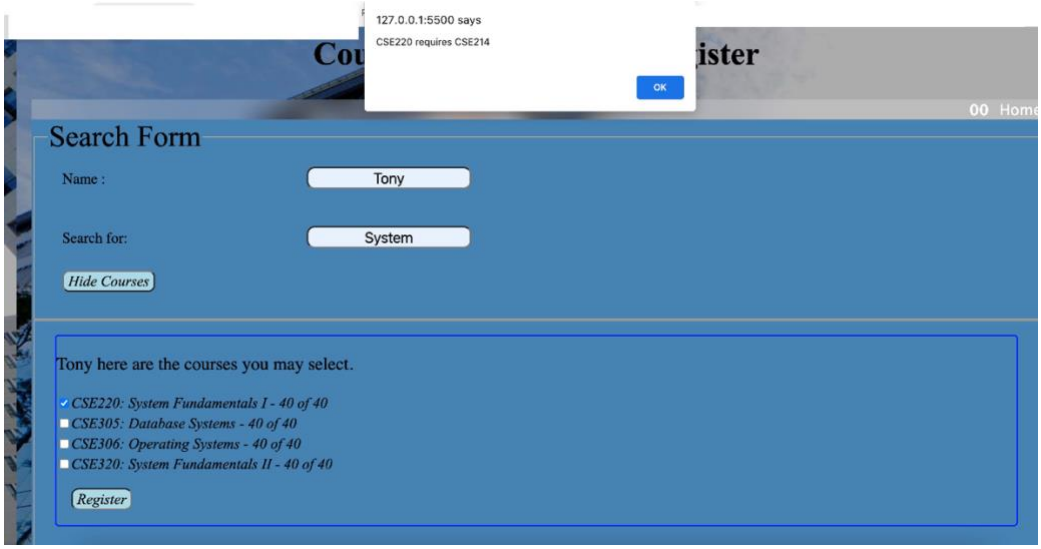
Large Previous Course Completed Screen



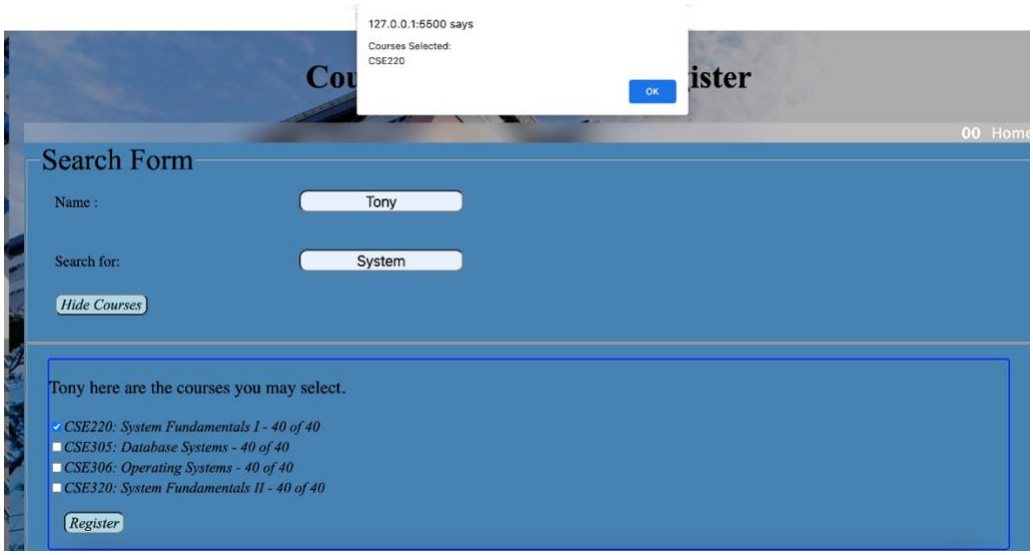
Large Search/Register Screen Filled



Large Search/Register After Search



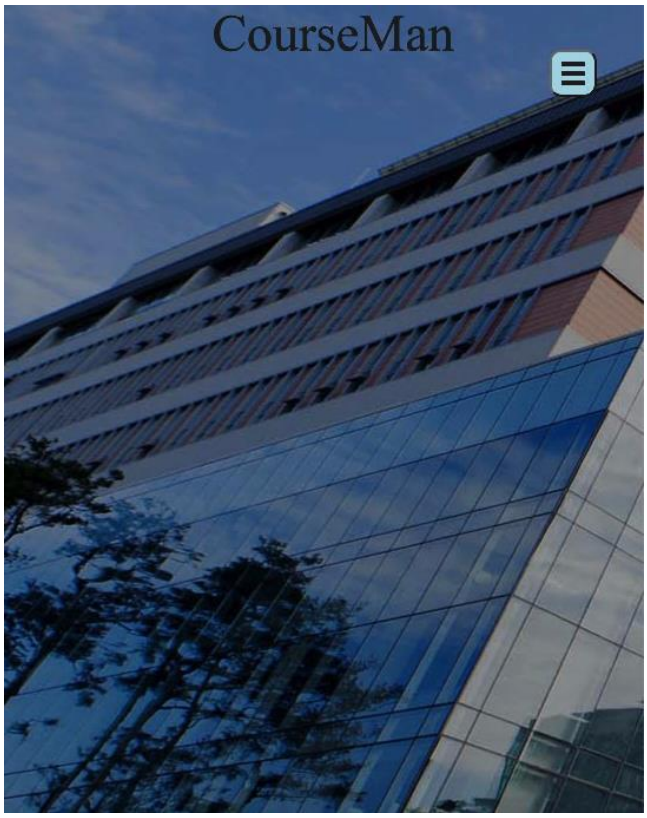
Large Search/Register Result 1 (error)



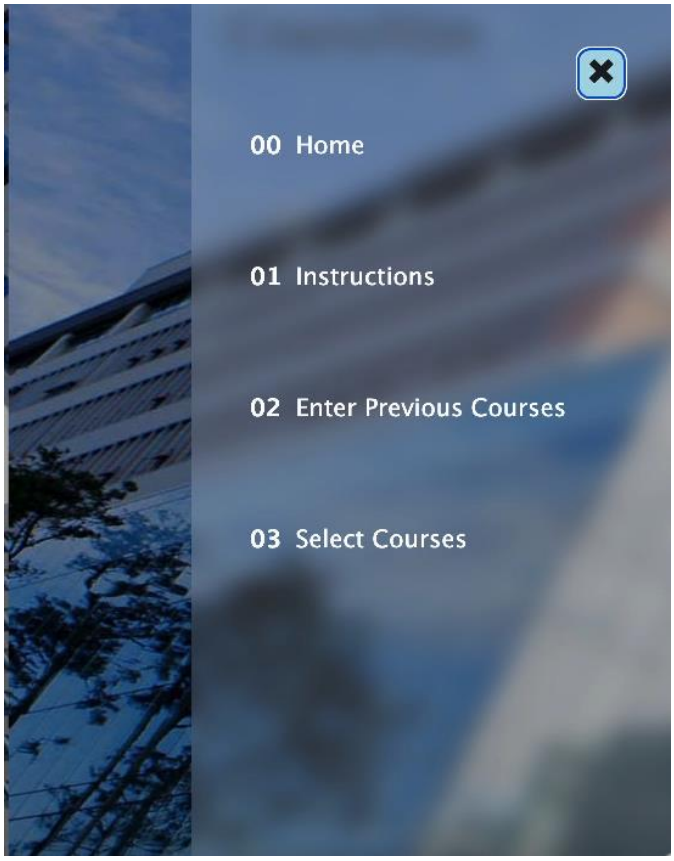
Large Search/Register Result 2 (success)

Small Screen [< 40em width]

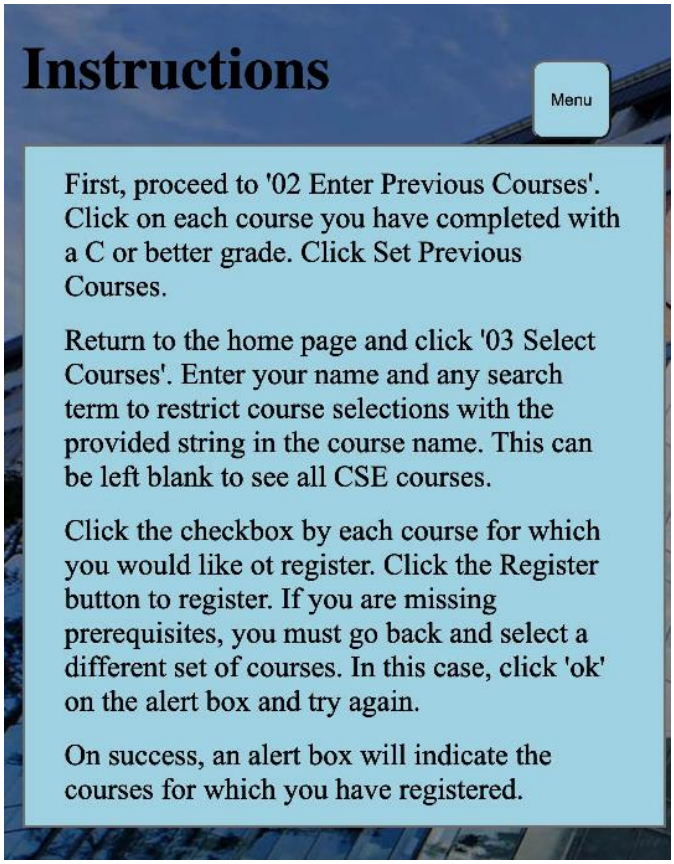
These mockups show screens less than 40em in width.



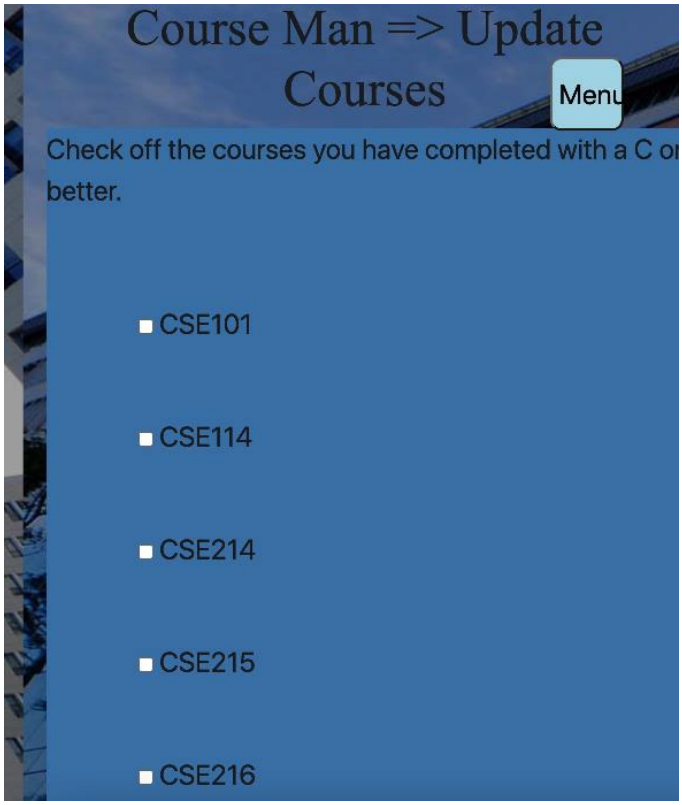
Small Home Screen



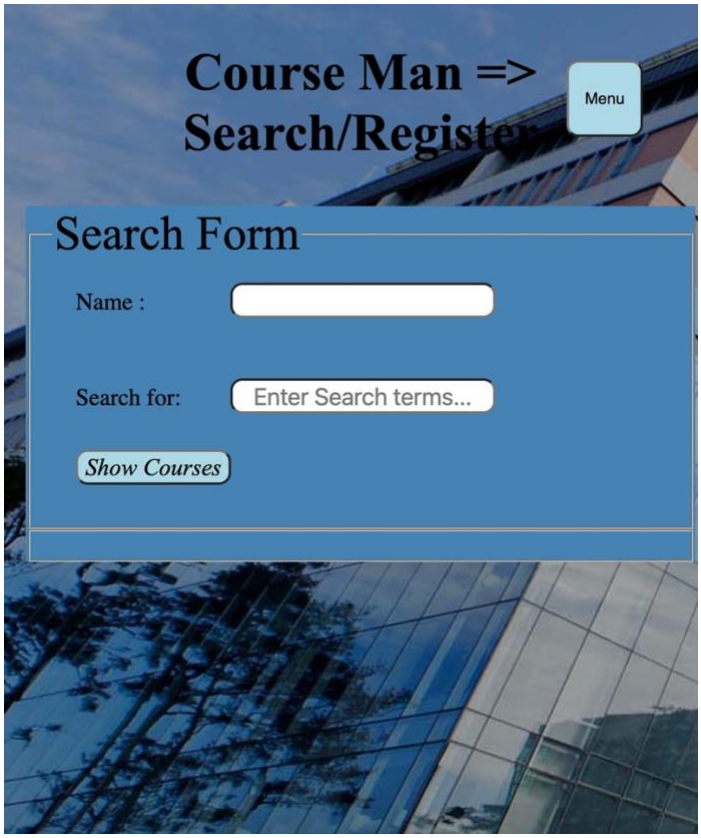
Small Home Screen [Menu Expanded]



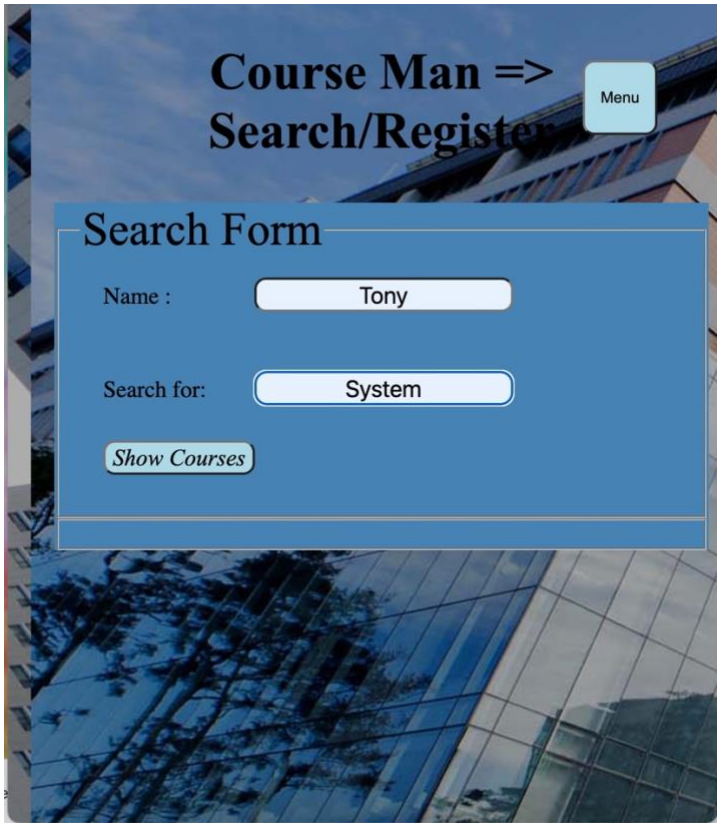
Small Instructions Screen



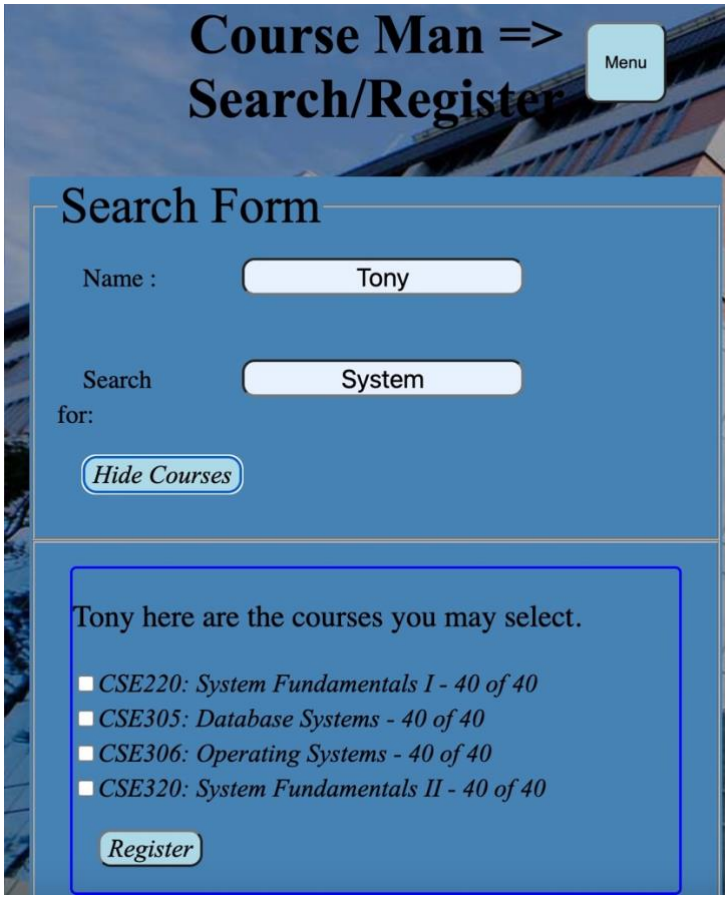
Small Previous Course Completed Screen



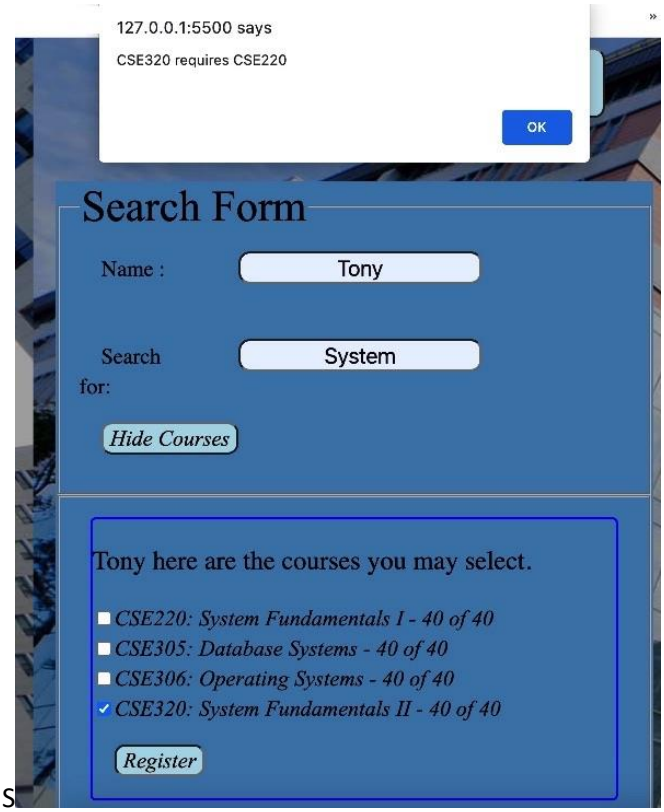
Small Search/Register Screen



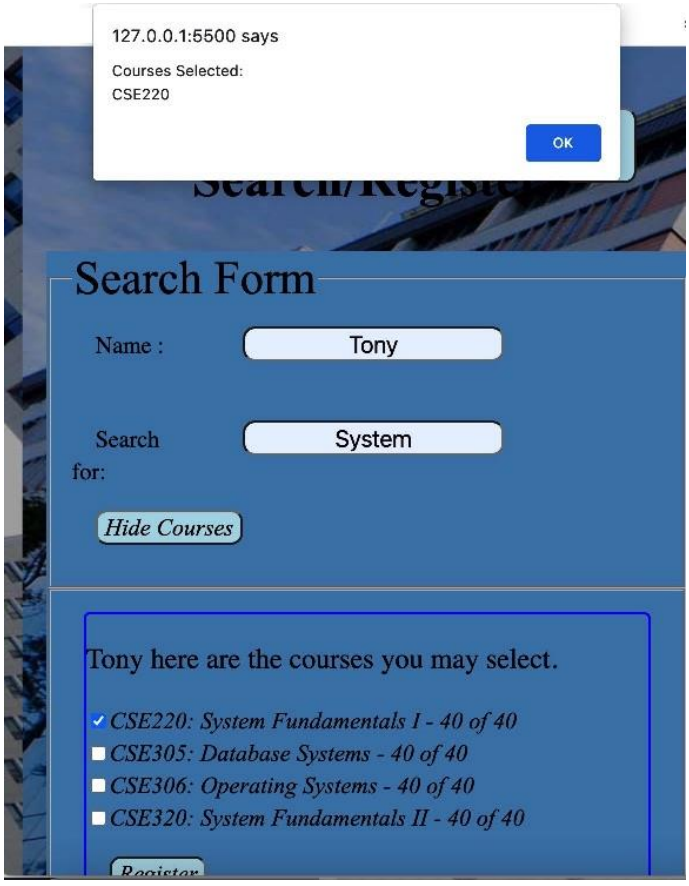
Small Search/Register Screen Filled



Small Search/Register After Search



Small Search/Register Result 1 (error)



Small Search/Register Result 2 (success)

Implementation Details

I will provide the background image seen in the app. You should attempt to copy as closely as possible the layout as well as most of the style (you can make minor style variations if you wish to be 'creative'). The mockups will help you do this but I also have some hints regarding color usage and fonts.

I recommend you set up your development tree with several folders to organize source and content by file type. So you may want to create a *courseman* folder and below that create folders for javascript (*js*), cascading style sheets (*css*), image files (*images*), and so forth. You can also choose to keep related css code in the same folder has the html files. You can pick how you want to organize your source tree but be consistent! That is the most important aspect when developing large systems especially if other developers are working with you. Later, when we work with React, there are a few tools to create an initial directory tree for a project. These are the create-react-app command and Vite. The latter is new but is gaining popularity in the fullstack development community.

Overall Design

I recommend you approach your development effort incrementally. By that, I mean build only a small piece of code at a time and test that the functionality for that module or page works before moving on. Since there are 4 pages to the app, you probably want to develop and test each one as you go. As such, you may also want to develop simpler pages first (i.e. home and instructions).

The application should implement the above mentioned nav bar with the 4 items listed. As a recommendation, I think you should consider building the NavBar first. Then proceed to each page.

The nav bar should be across the top of the screen for large screens and should be right-justified on the screen. For small screens, it should switch to a hidden column menu opened with a hamburger button (or button that says Menu, but see if you can get a hamburger icon in place). The panel should slide out when the menu/hb button is clicked. The button should switch to an 'X' to indicate it closes the nav bar (sliding it to the right and offscreen.)

The navbar should 'blur' the image behind it (either the band across the top or the column on small devices when the menu is visible). Note that creating the 'responsive' navbar may involve researching bits of css and javascript *that I have not covered in detail*. A useful video to help with that part of the assignment is:

Responsive navbar tutorial using HTML CSS & JS

Page Design

[Instructions]

You can use the text shown in the instructions page directly. It is formatted as 4 paragraphs with a lightBlue background and 2 pixel solid grey border (again, the actual style may vary slightly if you are feeling creative). Font size should be about 24 pixels. I used Times Roman but you can pick a different font, if you want. It is said that non-serif fonts (Helvetica, Arial, etc) are better for 'accessibility' (easing use for visually impaired persons) and more readable. Use a reasonable amount of padding and margin (it can differ a bit from what I show as long as it is reasonable and looks nice/professional.

I recommend you build this page first (starting with the index.html I provide.) Since it is just for navigation, you can create bare html files for the targets of the navbar menu items but do not implement those pages till later.

[Previous Courses]

The ***Previous Courses Completed*** screen should list the course shown as checkboxes inside of a div with the bootstrap classes *d-flex* and *flex-wrap*. This will allow the list to be laid out according to the amount of available space and make the checkbox elements wrap around in a responsive manner. They should read left to right and wrap around as shown. The button to select the checked courses should accumulate the course numbers in a string separated by spaces. If no courses are checked, the string should read '<None>'. This string is to be saved in 'localStorage'. Look up information on how a browser's localStorage works and can be used from Javascript. This is the only 'persistence' that must be implemented for this assignment. It is so the following Search/Register page can look up the taken courses list and verify the student has the required prerequisites.

This is a good choice for a second page to develop. Have it lay out the checkboxes for each course and gather the course numbers when the 'Set Previous Courses' button is clicked. Write the code to create a variable (with a name of your choice) in local storage with the course number list.

[Search/Register]

The Search/Register page will need to read the variable from localStorage to find what the user indicated as previously completed course work. This list will be used to check:

1. That the requested courses are allowed since the student has the necessary pre-requisites.
2. The student did not request a class they have already completed.

Either of the above conditions should cause an error dialog (as shown in the mockups). If the requested courses satisfy the conditions, an alert dialog should indicate the successfully selected courses (as shown in the mockups.)

The Search/Register page allows entry of a name and a search string. If no search string is entered, all CS courses should come up in the list. If a string is entered, only courses with the text in the description (with a case-insensitive compare), should come up in the selection list. The selection list is invisible before the search. It should simply be a div like:

```
<div id="courseList"></div>
```

You will use javascript to build a list of checkboxes and labels for each course found in the search. The checkbox label will have the Course number, title, and the number of seats left out of the capacity for the class. Again, see the mockups above. For now, we are not worried about checking if seats are left in the class since there is no persistence.

After the html is built (using javascript calls like '*createElement()*', '*setAttribute()'*, etc.), the resulting element can be set as the content (innerHTML) of the *courseLIst* div (use *getElementById()* to retrieve the div element). It will also build a submit button that will call javascript code to do the checks for prerequisites and re-taking of a class and post either a success alert or an error alert (if there were errors).

Verify that you can generate both types of errors (even both types of errors in a single dialog). Also, verify at least one success case.

Submission

When the code is working properly, you should do the following before submitting:

1. Clean up any dead code or debug code (remove it)
2. Write some meaningful comments to make the code easy to follow (mainly in your javascript but it doesn't hurt to make some notes in the HTML as well.)
3. Move your development folder to a top level folder called CSE316_HW1_<name>_<idnumber> where *<name>* is replaced by your name and *<idnumber>* is replaced with your student id. If your name is Joseph Kim with a student number of 12345678, then your folder name for submission is CSE316_HW1_JosephKim_12345678.
4. Compress your top level folder containing entire development tree (producing a zip file). Upload the zip file to the assignment page in Brightspace.
5. Navigate to the course Brightspace site. Click **Assignments** in the top navbar menu. Look under the category 'Assignments'. Click **Assignment1**.
    a. Scroll down and under **Submit Assignment**, click the **Add a File** button**.**
    b. Click **My Computer** (first item in list).
    c. Find the zip file and drag it to the 'Upload' area of the presented dialog box.
    d. Click the **Add** button in the dialog.
    e. You may write comments in the comment box at the bottom.
    f. Click **Submit**. ← ***Be sure to do this so I can retrieve the submission!***

Grading

I will be testing your submissions on Google Chrome under Mac OSX.

Grading will be based on the styling and functionality of the submission. The points are assigned as follows:

- **Static design**: Layout matches the mockups and style is similar to the same mockups. Content is correctly displayed: 25 points
- **Interactivity**: Checkbox, mouse click and menu functionality all work correctly: 15 points
- **Responsiveness**: Good style when resizing when resizing page as descried above. 10 points.