**INSTRUCTIONS & GRADING:**

Description

You have been asked by Cuyahoga Community College to begin work on the front-end of a

new web site that will be used by Student Services to help students keep track of their

information. Another developer whom you do not know has already completed work on some

services on the back-end which you will be utilizing. The requirements for the new site

are still in a very early state, and so all Tri-C needs you to do at this point is get

the basics of a Login Screen and an Account Information Screen in place. Since end-state

requirements are not fully known, the immediate goal is to establish a foundation for the

system architecture, and get some good patterns and technical practices in place for the

coming work in the future. As such, the quality, readability and reusability of your code

is of the utmost importance; even more important than the details of the functionality.

Instructions

NOTE: All resources are in the "Requirements" directory of the FinalProject solution.

1.) Carefully read the Design Document which was provided for you by the Project Architect.

2.) Carefully read the Technical Specifications provided by the Back-End Developer.

3.) Carefully examine the Mockup Images provided by the UX Designer.

4.) Modify the "FinalProject" solution to develop the new Front-End according to requirements.

REMINDER: You only have to modify "Index.cshtml", "Home.js" and "Home.css".

Grading

Functionality of Interaction with Login Service [15 pts.]

Functionality of Interaction with Create Account Service [15 pts.]

Functionality of Interaction with Account Information Service [15 pts.]

Functionality of Interaction with Account Add/Update Service [15 pts.]

Matches Responsive UX Requirements for Login Screen [15 pts.]

Matches Responsive UX Requirements for Account Screen Desktop [15 pts.]

Followed High-Level Design Document Specification [10 pts.]

Total [100 pts.]

**HIGH LEVEL DESIGN:**

Student Services Website Front-End Design Document

Global Front-End Requirements

•Contains a Header Area.

•Contains a Content Area.

•Contains a Footer Area.

•Content area should be easily replaced with other content.

•All field validation will be done on the back-end.

•All error message display handling will be done on the front-end.

•Any server calls should be done using JQuery AJAX.

•Moving from "Login" to "AccountInfo" will not change the page/URL.

•The front-end will handle transitions between pages and workflow.

•There is no need for workflow beyond Login or Create Account.

•There is no need to handle back button functionality.

•All display changes due to form factor must be handled front-end.

•All responsive design should use media queries.

Login Screen Only

•Login section allows a user who already has an account to log in.

•Successful login will take you to the Account Information screen.

•Unsuccessful login will display error messages on the screen.

•Create Account section will allow a user to create a new account.

•They must provide a unique username that is at least 6 characters.

•They must provide a password that is at least 6 characters.

•They must provide an e-mail address that contains an "@" character.

•They must verify their e-mail address by typing it in again.

•Successful login will take you to the Account Information screen.

•Unsuccessful login will display error messages on the screen.

Account Info Screen Only

•Account Info Section should list every value in the user's account.

•Each item should be updatable except for the Username.

•Disable/Replace the text-box and hide the update button for Username.

•No field validation required at this time for updating values.

•Do not use an HTML table for the grid; use styled divs instead.

•Add Element Section should allow the user to add a new element.

•User will provide a name and a value for the new account element.

•"Username" will not be allowed as the Element Name.

•This is currently the only input validation we have implemented.

Global Architecture

•Keep the HTML content layer free from obtrusive markup or scripts.

•Keep the HTML content layer organized and precisely scoped.

•Keep the CSS layer organized and tightly scoped.

•Use fully qualified class names wherever possible.

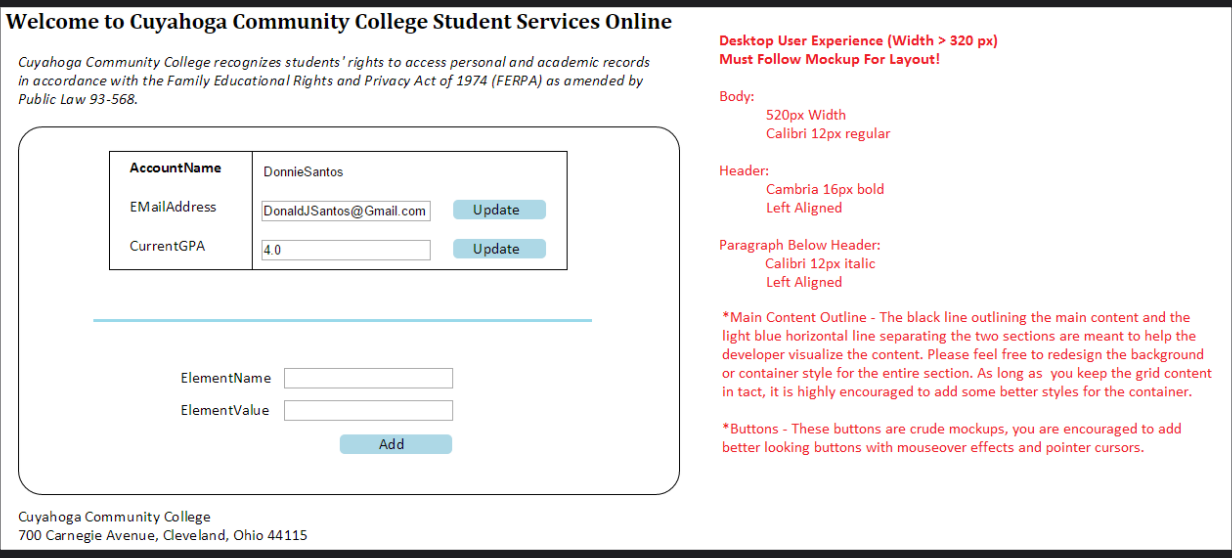
•Use direct-child selectors wherever possible.

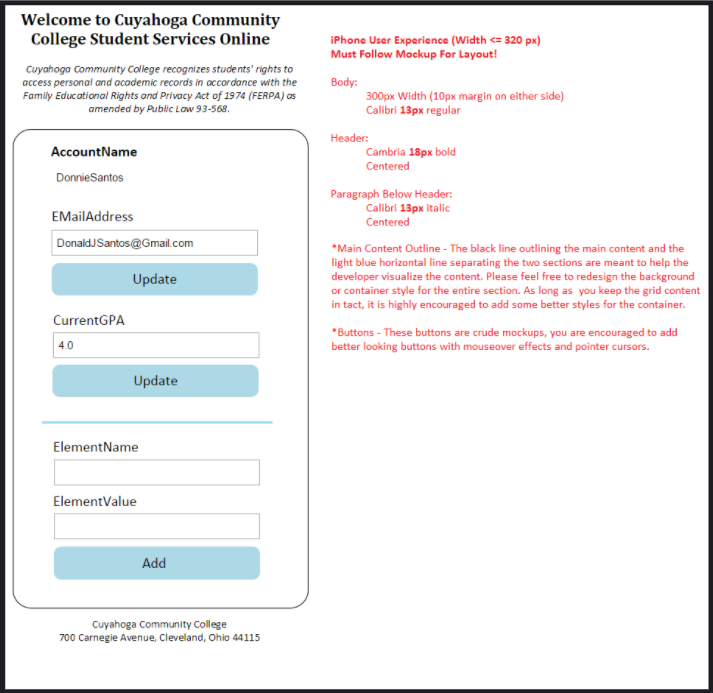
•Apply Object-Oriented JavaScript principles to organize behavior.

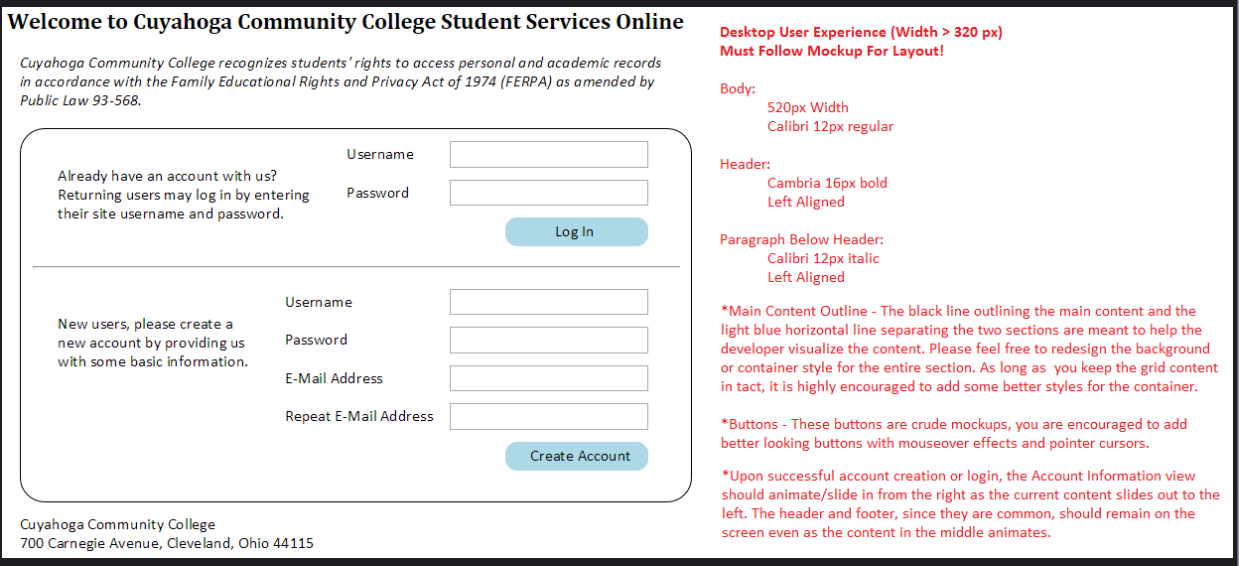
•Fully qualify and scope all methods to specific objects.

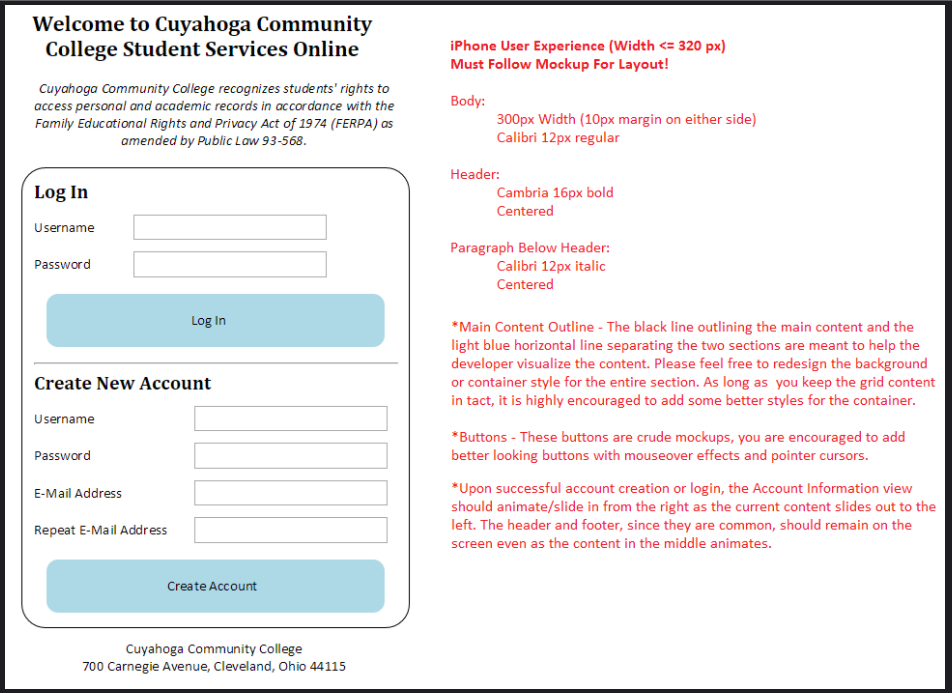
•Encapsulate reusable code within object methods.

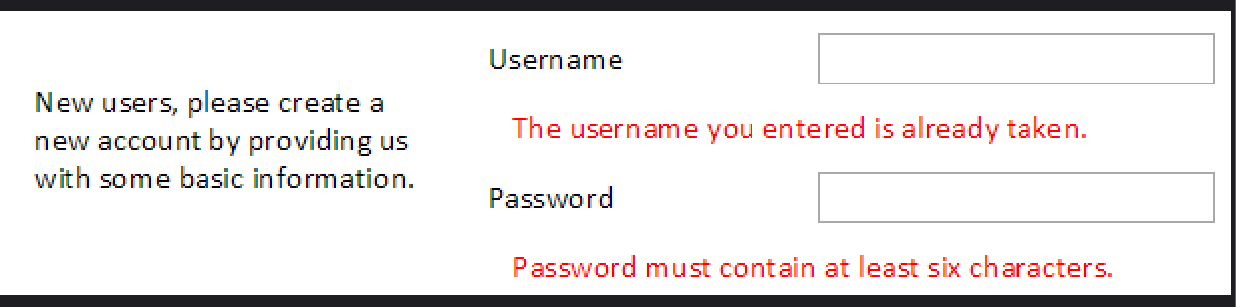
•Separate concerns; Keep objects and methods small and focused.











**TECH SPECS:**

