John Hargenrader

May 1st, 2023

CMSCI 475

Documentation

Table Of Contents

Page 1: Table of Contents

Page 2: Abstract

Page 3: Problem Description

Page 4-5: Solution Overview

Page 6-39: User Manual

Page 40-50: Designers Manual

Page 51: Maintenance Issues

Page 52: How to Access

**Abstract**

My senior capstone project is called Mount Connect. It is a dynamic full stack website internal to students and administrators here at Mount St. Mary’s. My client is Associate Provost Dr. McCarthy who asked for this website to be created. He wanted a website that allowed Mount students to connect with one another to find off campus roommates and roommate groups. There is no current site here at the Mount that allows students to find off campus roommates or roommate groups. So, for the past year I’ve been developing a community style platform that solves this problem. The website only allows Mount students to register and access the website. Once a student registers and confirms their registration through their email, they will be able to log in and enjoy the website. Students have their own profile page, they can edit any of their information, and they can search for other students based on different criteria such as if a student is looking for roommates, class year, and gender. They can view other students’ profile pages as well. Student can create a roommate group and then invite other students to their group. Students can also search for groups and request to join groups if they are not currently in a group. Each group has their own group profile page as well that is editable by only the group master who is also in control of accepting students requests to join and removing students from their group. There is administrator functionality for my client that allows him to add other administrators to the site, as well as manage students and manage groups. My main future goal of the site is to add a page for students to be able to create posts to further emphasis the idea of a community style website.

**Problem Description**

Mount Connect tries to solve the problem of off campus living for students. The Mount currently has one platform for students to use for living needs which is called simplicity. This site is only for students looking to live on campus. There is no platform here that connects students together to find roommates for off campus living, find roommate groups, or create roommate groups. This problem makes it difficult for students in a handful of scenarios. Students who transfer to the Mount and might want to live off campus but have not met a group of friends yet to live with are forced to live on campus. Students who are looking to live off campus and want to see what roommate groups are available to possibly join or connect with can’t without a platform enabling them to. Students who want to find other students wanting to live off campus cannot come together to achieve living off campus without a platform like Mount Connect. These are just a few examples that cause problems with students here at the Mount attempting to live off campus. Most students who live off campus have succeeded by having mutual friends and friend groups since the beginning of their college years. Some students may not have big enough friend groups to easily find a place that fits their needs. Friendships and friend groups usually begin to occur in a student’s freshman year. Since some students might not be fortunate enough to experience that, they are forced to live on campus with either random roommates or a friend or two. There is no current solution that allows these less fortunate students to connect with other students and have that hope of living off campus. Most college students want to feel free or independent, especially towards their later years of college. Living on campus usually does not make a student feel free. My freshman year, living in the dorms was not amazing, but it was important. I met a ton of friends and gained close relationships. Only with that experience was I late able to live on campus my junior year with those same people I met in the dorms. Most of the people I talked with at the Mount wanted to live off campus after their freshman year. My friends and I struggled to find a place to live off campus, and even later in our Senior year, we struggled to find a 4th roommate to take an old roommate’s place.

**Solution Overview**

To solve this problem stated above, I used my computer science knowledge to build an online platform that enables students to connect with each other. That is how I came up with the name “Mount Connect”. Mount connect is a dynamic website allowing students to have their own profile pages, and their own roommate group pages. Students can connect via search capabilities. I built this idea with the following technology: ASP.NET, SQL Server, Entity Framework, and Azure. All these technologies are software built by Microsoft and allow you to create a dynamic full stack web application. ASP.NET is a web framework that uses the programming language C#. I utilized this by building out the front end of the website allowing me to create the user interface and templates. Templates essentially are boiler plate HTML elements created in .NET that allow you to put elements on a page such as labels and other data elements where you can write code that populates those elements with user data. For my application’s purposes, all these templates would be populated with students’ data like their full name, an about section, class year, gender, if they are looking for roommates or not…etc. All this data would be stored in my database which is SQL Server. I can easily connect my database through ASP.NET via the Entity Framework. This is an ORM (Object relational mapper) that turns my database tables into objects that I can easily manipulate and use in ASP.NET. This is how I effectively implemented the essence of a dynamic full stack web application. The application can update, and change based on what user is logged in, and what other users do on the site all in real time. This is how I was able to implement profile pages, and group pages which were the most important components of the website. Allowing students to have their own personal profiles, and personal roommate group pages made it possible to highlight the theme of connecting students to one another. To view and look at student’s profile pages or group pages, they could search for students or groups based on several different criteria and click on their profile page. The student could see the other student’s information, and even invite them to their roommate group if possible. Students can even request to join roommate groups that are not full. Another important implementation was allowing students to say if they were looking or not for roommates. Similarly, to how on LinkedIn you can say you’re open to work or not letting employers know you’re looking for a job, you can say your looking for roommates letting other students know that you are looking for roommates. I made it general like this because saying you’re looking for roommates can either mean you’re looking to join a roommate group, you’re looking to invite students to your roommate group, or you’re even just looking for roommates to form a new roommate group. This again highlights the whole idea of connecting with one another. One issue I encountered while designing the application was allowing a student to add a profile picture. I could not decide between storing the image in the database or storing the URL of the photo that is in a directory in my web application. Storing the URL and retrieving the photo in the directory would be more efficient in terms of data storage because storing raw photos in a database can eat a lot of memory. I decided to just store the photo in the database since it is only one photo per user. If I later wanted to allow students to be able to add more photos such as photos of their off-campus house, then I would need to transition storing all photos in a directory of my application. I am still working on implementing the profile photo logic which has been one of my most difficult tasks. I also plan on adding a page where students can create posts which further highlights the idea of connecting students with one another and making the application more of a community. I’d implement this the similarly to how I’ve implemented a lot of the other pages. Another problem I ran into was security testing. It was hard to do proper security testing locally since the main points of emphasis on security would be on the production server hosting this web application. I planned on hosting the application through Azure which is a cloud platform for Microsoft. It is easy to deploy .NET applications to Azure, and there are tons of security features I can easily enable in the Azure interface. My biggest issue was handling emails in my application. Registering on the website sends the email using a verification link, and once clicked by the user it allows them to log in. Similarly, I used email to allow students to send invites and request messages for the group functionality. I did all of this with a local smtp server, and my problem came when trying to deploy the site, having to change to a production level email server. This is a later problem I will finish when getting closer to deployment.

**User Manual**

Login page: When first accessing the website, you will be directed to the student login page. There will be a form to submit your Mount email and password. You then will click the login button to access the core features of the site. If the email or password is wrong, you will get an error message in red below the student login header. If your login is successful, you will be redirected to the homepage. If you do not have an account yet. You will click the sign-up button in the nav bar or the link below the white box form that will redirect you to the sign-up page. There is also a forgotten password link below the password portion of the form that when clicked, redirects you to the forgotten password page.

A screenshot of a computer

Description automatically generated with medium confidence

Sign up page: The Sign-up page will be a form to fill out your first and last name, your mount email, your gender which can be male or female or prefer not to say, your class year, and looking for roommates or group which is yes or no. Yes, means you are looking for roommates or a roommate group and no, means you are not. You then will have to create a password and confirm that password. If every field is inputted correctly and your passwords match, you will click the submit button to confirm registration. If any input is missing, or your passwords do not match, you will be shown an error message below the student registration header. If everything is valid, you will be redirected to a page that tells you to confirm your registration via the mount email used in the sign-up process. You then will go to your mount email on outlook and see a link to confirm your registration. Once the link is clicked, it will redirect you to a verification page confirming your registration, and then you will click the here button or login button on the nav bar to log in. If this process goes wrong, you will be shown an error message and forced to register again.

A screenshot of a computer

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Homepage: The homepage contains an input box to search for a student by their first, last, or full name, or to search for a roommate group’s name. If searching for students, you must choose students in the first drop down list next to the find button or choose groups if searching for roommate groups. If groups or students is not specified, you will be shown an error. If a student’s name or group does not exist, you will be shown an error message asking you to try a more general search. You also can search for groups without adding a group name, just clicking the find button with groups selected will show all groups that are not full in alphabetical order. You can also search for students based on other search criteria. If the student list item is selected three more drop downs appear allowing you to choose students by class year, by gender, or by if a student is looking for roommates or not. The order of which the students will show is based on how many search criteria matched. For example, if you search for seniors who are males and who are looking, it will show the students that are male seniors looking on top while the rest of students shown will be match less criteria. It is a ranking system; you can also combine a student’s name with other filters to narrow your search even more. The list of students shown is in a grid form where it lists the students full name, email, class, gender, looking or not, and a button to click to view their profile page. When searching for groups, a similar grid will appear listing all available groups alphabetically if no group name was inputted, or if a group name was inputted it will search for that group. If it can’t find that group name, you will be shown an error. The group grid will show the groups name, contact info, about section, and roommate capacity to show how many students are in the group out of the maximum amount, and then the view group button which will take you to that groups profile page.

* What you first see

A screen shot of a computer

Description automatically generated with low confidence

* No students or groups specified

A screen shot of a computer

Description automatically generated with medium confidence



* Searching by name

A screenshot of a computer

Description automatically generated with medium confidence



* Searching by filters, you can see it shows the students that match the filters ranked.

A screenshot of a computer

Description automatically generated with medium confidence



* Search for group by name

A screenshot of a computer

Description automatically generated with medium confidence



* Search for available groups.

A screenshot of a computer

Description automatically generated with low confidence



* No group was found with that name.

A screen shot of a computer

Description automatically generated with low confidence



* No student found with that name.

**A screenshot of a computer

Description automatically generated with medium confidence**



Public Profile Page: The public profile page of the student is only accessible when clicking on the view button on the home page when searching for a student. You will see the students full name at the top, a default profile photo to the right of the full name, an about section under the full name, looking for roommates or not under the about section, and a contact section under the looking section. You will see an invite button on their profile page if that student is currently not in a roommate group and you are currently in a roommate group that is not full. When clicking the invite button, a success message will under their full name telling you to wait for that student to accept the invite and that button will disappear. If there was an error, an error message will be shown in the same place.

* Public profile page where you’re in a group and it isn’t full and that student is also not in a group

A screenshot of a test

Description automatically generated with medium confidence

* Success message after clicking the invite button.

A screenshot of a computer

Description automatically generated with medium confidence

* Email Sent to the person being invited to the group.

A screenshot of a computer

Description automatically generated with medium confidence

* Student already in a group (no invite button).

A screenshot of a computer

Description automatically generated with medium confidence

* The group I am in is full, so you cannot invite the student who is not in a group (No invite button).

A screenshot of a computer

Description automatically generated with medium confidence

Public Group Profile Page: Similar structure as the student public profile page. Contains the group name, number of members out of maximum members, about section, contact section, the group master’s information, and a list of students in the group. You can also click on the student’s name that would redirect you to their public profile page. A button will show on the bottom of the page that says request to join if the group is not full and you are not already in a group. When clicking request to join, a success message will appear indicating the group master received an email and must accept.

* Group is full, no request button:

A screenshot of a computer

Description automatically generated with medium confidence

* I’m already in a group, no request to join button.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

* I am not in a group and the group is not full.

A screenshot of a group of members

Description automatically generated with medium confidence

* Success message after requesting to join button clicked.

A screenshot of a computer

Description automatically generated with medium confidence

* Email sent to the group master after request is sent.

A screenshot of a computer

Description automatically generated with medium confidence

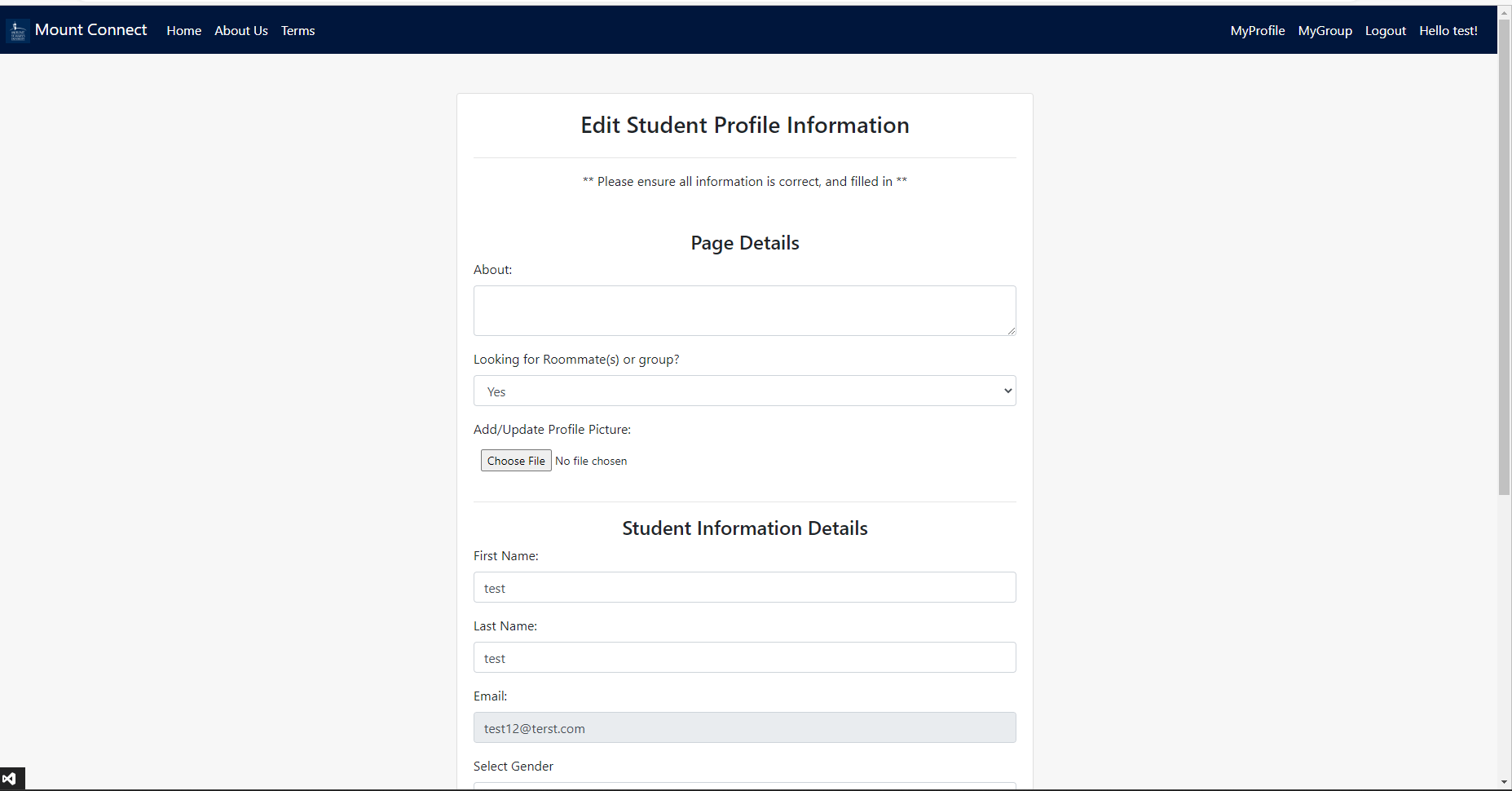
Private student profile page: Same look as the public student profile page, only difference is that there is an edit button at the bottom when clicked, it redirects you to the edit page which allows you to edit your information. This page is unique to each student and cannot be accessed by any other student. You can access this page only by clicking the MyProfile button at the top of the nav-bar.

A screenshot of a computer

Description automatically generated with medium confidence



Edit student information page: This page has the following inputs, the about section which is initially empty when creating your profile. The default about section is put in as you can see in the above picture when first registering. Then, there is a drop-down list saying if you are looking for roommates or not, add/update a profile picture button, first name, last name, mount email, drop down list for gender and class. There is a save button underneath the inputs and deleted button under the editing information. When the save button is clicked, all the information in the inputs will be updated, and if the delete button is clicked and confirmed it will delete your account and redirect you to the log in page. You can only access this page via clicking on the edit button on your private profile page. Also, all the inputs are populated with your current information to begin with.



A screenshot of a computer

Description automatically generated with medium confidence

Private group profile page (not in a group): You can access this page by clicking on the MyGroup tab on the top of the nav-bar. If you are not in a group, you will see a form to create a group. If you are in a group, you will see your private group page. The form to create a group consist of a group name to fill out which must be unique, it shouldn’t be the same as any group name in the system, group about section, group contact information section, and a maximum number of roommates which must be larger than 2. If you do not put valid inputs in, an error message will appear. The about section and contact section must be 300 characters max and 150 characters max respectively or an error message will appear. Once you’ve created a valid group it will redirect you to your group profile page.

* Form to create a group when you first click on the my group tab.

A screenshot of a computer

Description automatically generated with medium confidence

Group name is taken error.

A screenshot of a computer

Description automatically generated with medium confidence

* Invalid input

A screenshot of a computer

Description automatically generated with medium confidence

Private group profile page (in a group): The private group profile page is accessible when clicking on the myGroup tab in the nav-bar. If you are not in a group, you will see the form to create one. The page looks the same as the public group profile page except for a couple of things. If you are the group master which can only happen if you created the group or the current group master makes you the group master, you will see an edit button that allows you to edit the group which when clicked takes you to a page to handle that functionality. There is also a leave button that when clicked makes you leave the group. This is shown on this page for all members of the group. If the group master clicks leave, a new group master in the group is randomly assigned, if you are the only one in the group which makes you the group master and you leave, the group is automatically deleted from the system.

* Group master’s view

A screenshot of a computer

Description automatically generated with medium confidence



* Ordinary member’s view

A screenshot of a computer

Description automatically generated with medium confidence

Edit group information page: This page is only accessible to the group master and can be accessed via the private group profile page by clicking the edit button. The features you can do are update the group name, group about, group contact information, max number of roommates, assign a group member as the role of group master, remove a roommate, and delete the group. Clicking the save button saves the group information, clicking the assign button next to the group members name assigns them as group master, and clicking the remove button next to the group members name removes them from the group. Clicking the delete button will delete the group entirely from the system.

A screenshot of a computer

Description automatically generated

Forgot password page: You can access this page from the sign in page link that says forgot password?   
You’re redirected to a page that prompts you to put your email in. If the email is not in the system, an error occurs. If it is a valid email, the verification code is sent to the email used. You must go to your mount email to get the code and come back to the site to put it in the verification code text boxes. Each letter goes into each box in the correct order it was in the email. Once the code matches, it shows you a new password and confirms password field. Both passwords must match, if they do your password is reset and a message is shown to click to be redirected to the log in page. If any of this causes errors such as invalid code input or non-matching passwords, an error message is shown.

* Valid email inputted, code is sent to that email.

A screenshot of a computer

Description automatically generated

* Email containing

A screenshot of a computer

Description automatically generated with medium confidence

* Code was successfully inputted.

A screenshot of a login

Description automatically generated with medium confidence

* Passwords successfully matched, success message shown.

A screenshot of a login box

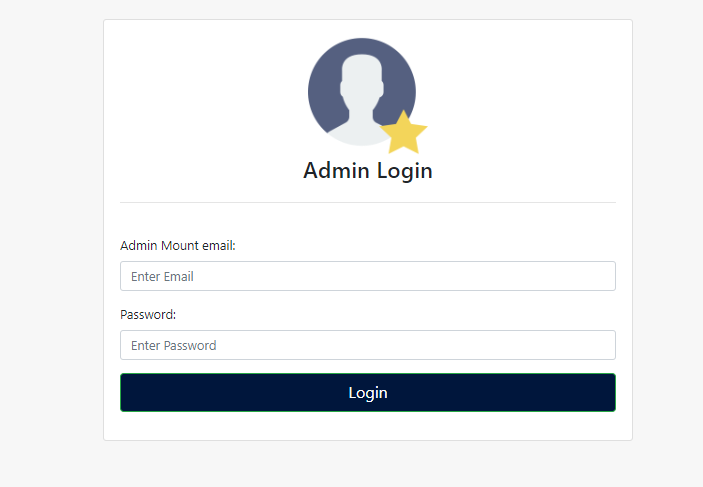
Description automatically generated with medium confidence

Invalid code entered:

A screenshot of a login form

Description automatically generated with low confidence

Admin login page: The link to access the admin login page is at the bottom footer of the web site. Clicking the link will redirect you to the admin login page that makes you put in your admin mount email and password. Not entering a valid email associated with that password results in an error. A successful login redirects you to the student management page by default.



Student Management Page: This page is only accessible to administrators by clicking on the Students tab of the nav-bar. The page shows the left side for an admin to search for students by name or by searching for all the students in the system alphabetically ordered. It is the same output as the homepage for the grid result, but the manage button instead of the view button allows an admin to have the right side of the page populate with the student’s information to edit or delete the student. If an admin enters a student name that does not exist in the system, an error will appear letting them know that. If an admin edits any of the students information that isn’t valid an error message will also appear.

* Searching for all students

A screenshot of a computer

Description automatically generated with medium confidence

* Manages a student (manage clicked on)

A screenshot of a computer

Description automatically generated



* Can delete the student from the system.A screenshot of a computer

  Description automatically generated with medium confidence

Manage Roommate Group Page: This page is also only accessible to administrators and can be accessed via the Groups tab on top of the nav-bar. It is the same logic as the manage students page but you can search for groups by name or all groups alphabetically ordered, manage the group by clicking on the manage button associated with that group in the grid. The right side of the page populates with that groups information and also allows an admin to assign a new group master, remove a group member, or deleted the group.

* Searches for all groups.

A screenshot of a computer

Description automatically generated with medium confidence

* Manages a group clicking on the manage button.

A screenshot of a computer

Description automatically generated with medium confidence



Manage Admin Page: This page is only accessible to the admin master who is me right now but will soon be my client. He can access this via the admin tab on the nav-bar. This page shows all administrators currently in the system via a grid. The admin master can delete any admins except themselves from the system. There is a form underneath the grid that allows the admin master to add an admin to the system. He must input their first and last name and email. Then click add. An email will be sent to that administrator that has a link for them to click on in the email to confirm being added. If the admin has already been sent a request, an error message will occur if the admin master attempts to add the same admin master. If the form is valid, a success message appears.

* What the page first looks like depending on what admins are in the system.

A screenshot of a computer

Description automatically generated with medium confidence

* Admin already in the system error shown.

A screenshot of a computer

Description automatically generated with medium confidence

* Deleted the “aaa aaaa” admin and then added a new one with the success message showing.

A screenshot of a computer

Description automatically generated with medium confidence

* Email sent to the admin added.

A screenshot of a computer

Description automatically generated with medium confidence

Admin Registration form: This page is only accessible via the link generated in the email sent to the admin who has been added. It is impossible to access if the admin master did not add you to the system. The page contains your email prefilled in, and inputs to add your first, last name, and password and password confirm. Clicking register will bring a success message and allow you to click the here button that takes you to the admin login page.

* Registration form by clicking the link in the email.

A screenshot of a computer

Description automatically generated

* Success message after filling out the form.

A screenshot of a computer

Description automatically generated with medium confidence

**Designers Manual**

Student Login Page:

When the login button is clicked, it has an onclick function that gets the text input of the email and password. It then makes a request to my database via a stored procedure to get the matching email and password of that student’s record. If there is a match it returns all their information to that row of the student table. The password uses a hashing class to hash the input password to see if it matches the same password in the database that is also hashed using bcrypt and a random salt for that record. If all of this is true, I set the session id of the user equal to their id in the database. I also set a session cookie equal to their group id if they are in a group. Once getting all this information I can determine whether to show certain pages or not to the user based on their information. The user is then redirected to the homepage of the application.

Sign Up Page:

The user inputs their first and last name, a mount email, gender, class year, if they’re looking for roommates or not, a password, and that password confirmed. Once the submit button is clicked, the onclick function is fired validating all the input, it then generates a GUID, which is a unique identifier code, it binds this to an anchor tag that is put inside the email template. The email template has replacement variables that the first, last name, and email go into, the email is sent to that user who registered. Then, the user’s current session variables are set to all of their information they entered, once the user clicks the link the email sent them, they are redirected to a verification page that grabs the url parameters for the unique code and email associated with the registration, confirms through the session that everything matches, and shows a success message if it all matches. They are then added to the database via a stored procedure, and then can log in to the application through the login page.

Home Page:

The home page is where students can search for other students or groups. Based on if a user puts input into the text box or not, I will make requests to my database using stored procedures to get the necessary information. If a student or group is not selected from the drop-down list when the find button is clicked, an error message appears. I have two functions that show either students or groups based on which drop down item is selected when the find button is clicked. For the group function, if there is not input in the text box for group names, I get all groups alphabetically from the database that are from a stored procedure and bind that result to the group grid view and show it. If a group name is inputted, then I get the associated group with the inputted group name. If no student name or group name matches, I show an error message. For the student search, there are drop down lists that will appear based on which ones are selected. I have a stored procedure that takes the three inputs (gender, looking, class year) and it queries my student table and ranks the results for each student that matches the inputted filter. I then get that result and bind it in the student grid view and show it. The view buttons are also binded to each student or group row based on there ID in the database. I generate a URL to the public profile page of group or student, and then add the ID of the student or group to the URL parameter and bind it to an href attribute of the button that is clicked. This allows a student to be redirected to that profile page.

Student Public Profile Page:

The student profile public page is a generic page with the full name label, about label, looking for roommate’s label, and contact label. These labels are defaulted to null values. When a user is redirected to this page from the homepage, the ID of the user is taken from the URL parameter, I then query the database from a stored procedure getting the user’s information and bind the null values to the appropriate values of the student. This allows me to populate the same page for all students with their information making it dynamic. The invite button shows only if that student isn’t a group which is information, I also get from the student ID, and if the current user isn’t in a group that is full which I get from their session ID because they are the ones currently logged in. I that is all true I show the button. If the button is clicked, I grab the email template for inviting students to a group and bind it with all the necessary information for that user so the email can be sent to the student being invited. Similarly, to registration functionality, I bind a unique URL to an anchor tag in the email so that when the student clicks it, it takes them to a verification page where I pull the code and information from the URL and session variables to ensure I’m adding the student to the correct group. On the verification page, I also check to see if the group is full because a student could accept the invite after the group became full or deleted, I check for this and show an error if need be.

Group Public Profile Page:

This is the same exact functionality as the student page, but I show the information based on the groupID from the url parameter. I show the request to join button if the current user is not already in a group, and if that group is not full. This time, the email is sent to the admin master, the url is generated in a link for them to click in the email template, and if the admin master clicks it, they are redirected to a verification page that grabs the session and url information and makes a request to my database to insert the student in the group via a stored procedure checking for the necessary validation that the group is not empty and that student isn’t already in a group.

Private Profile Page:

This the same functionality has the public profile but populates the null labels with the current logged in students’ information by accessing the sessionid. When a student clicks on the myprofile page they are redirected to this page and the labels populate by getting the information from a stored procedure that takes the student’s id as a parameter and gets that student’s information. I show the edit button, and if the student clicks the edit button it redirects them to the edit page.

Private Group Page:

This page first grabs the session cookie group id. If the group id null, the user is not in a group, so I redirect them to the create group page. If it’s not null, I use a stored procedure that takes in the group id and returns all the information about the group. It then populates the null labels for the page to correct values of the group and list of students in that group. It also shows the leave button for all group members. If the currently logged in user is the group master which also is stored as a session cookie when they first log in, I will show the edit button as well. If the edit button is clicked by the group master, it takes them to the edit group page. If the leave button is clicked, the user is removed from the group, I then check if that was the group master leaving and if there are more members in the group. If there are more members in the group, I assign a new group master randomly, if the group master was the only member, the group is deleted from the system. Once the student leaves, they are redirected to the homepage and their appropriate session cookies are updated based on them not being in a group anymore. The application uses stored procedures to remove the group member, check for the master, check for more members, and delete the group or reassigned a new group master if necessary.

Edit Profile Page:

The edit profile page shows textboxes for all the students information. The textboxes autofill with the student’s information based on their session id. When the page is loading, my code grabs the session id, get’s the user’s information, and populates the textboxes with their information. The user then can edit any of those textboxes except the email and click save. Once they click the save button underneath their information, I get all the values from the textboxes and drop-down lists, then call a stored procedure to update that record of the student based on their session id. I also validate their inputs to ensure they are valid. Another thing is that their about section can be empty. If it is empty, the public and private profile page of that student will show a boiler plate about section that is a string that I just bind a few of their information records to. After saving, I redirect the user to their private profile page to see their page dynamically update based on their changes. There is also a delete button at the bottom of the page that when clicked, deletes the users from the system, clears their session variables and redirects them to log in page.

Edit Group Profile Page:

The edit group page is like the student profile page functionality for updating the group information (the about section, contact section, group name…etc.). I get the session cookie group id value and populate the page’s textboxes. I also show a list of the students in the group with a remove and assign button next to each student. When the page is being loaded, I get all the students in the group based off the group id using a stored procedure and bind each student to a data list where each row in the data list behind the scenes stores their student id. This makes it easy to work with the two buttons associated with each student. I do not show the group master because they should not be able to assign themselves as group master or remove themselves, they can only remove themselves from the leave button under the private group profile page. When the assign button is clicked, I update the group table’s record for that group column that stores the foreign key of the student id to the student table to that student’s id being assigned as the new group master. I then redirect the previous group master who has assigned a new student to the private group profile page and then the page will dynamically change to not show the edit button because they are no longer the group master. The remove button similarly calls a stored procedure to update the student’s record group id column to null. This removes them from the group. There is also a delete button at the bottom of the group edit page that when clicked removes all students from the group first, then deletes the group from the database due to the foreign key constraint.

Forgot Password Page:

The forgot password page first takes your mount email as an input. It queries the student table in the database to find a record associated with that mount email input. If it does not find a record, it shows an error message. If it does, I generate a unique 6-character code and store it as a session cookie which get’s encrypted. I generate an email using the password recovery template, I add the code to the template, and send the email to that user. A success message is shown telling the user to go to their email to get the code to be inputted into the textboxes. The 6 block sized textboxes are then shown for the code to be input into. I added JavaScript that moves the user to the next text block after a single character is put into it. They click the verify code button. I then concatenate the text blocks characters into one string, and then get the correct code from the session cookie generated and compare the two codes. If they match, I hide the verification code portion, show a success message, and show the text boxes to add a new password and confirm that password. The student then clicks confirm, if the passwords both matches, I used a stored procedure to get the record associated with that email and update the user’s password which is hashed the same way as the sign-up page before being inserted into the database. Otherwise, I show an error message. I then disable the confirm button and hide the password textboxes, show a success message that has a link for the student to click that redirects them to the log in page.

Admin Login Page:

The admin login page works the same as the student page but only looks up the correct email and password matching in the admin table.

Manage Student Page:

The manage student page is only shown to admins who are logged in by getting the session variable that is equal to the user type. This would only be admin or admin master user types. When the find button is clicked, I check if the textbox for student’s name has input, if it does, I make a request to my database via a stored procedure to get all the students that have the name input. It can be a full name, first name, or last name as input the same as the homepage logic. I then bind the grid view to show the list of the students. If no student matches the input name, an error is shown. If no input for name as submitted, I bind the grid view with an alphabetical ordering of all students in the system by last name. I have a manage button binded to each row of the grid view. The button contains the student’s id. Once the button is clicked, I populate the right side of the page with the student’s information. I always have the right side disabled so when a student is clicked, you cannot manage an “empty” student. I validate that student’s information that is being edited is valid, and when the save button is clicked, I update the student’s record in the database and update the page dynamically to show the updated information of the student. There is also a delete button under the student’s information which allows the admin to delete the student from the system. If it is clicked, I call the same stored procedure that deletes a student from the system.

Manage Group Page:

The same logic as the manage student page but gets groups instead of students. When the manage group button is clicked, I also populate the students in the group list with all the students in that group. I add two buttons next to each student’s row that is assign or remove. I make the group master green to show that they are the group master compared to the other students in the group. I also disable the assign button for the group master for the same reason as the edit group page functionality. When the assign button is clicked, I update the student the same way as the edit group page, and if the remove button is clicked, I remove the student the same way as the edit group page as well. If there is one student in the group, it automatically deletes the group if they are removed. There is a delete group button which removes every student from the group then deletes the group.

Manage Admin Page:

This page is only accessible to the admin master. Once the admin master clicks on this page, the page on load function populates a grid view with all admins in the admin table. It also binds a delete button to each admin allowing the admin master to remove admins from the system. If it is clicked, it removes the admin from the admin table via a stored procedure. It does not allow you to delete yourself since you are the admin master. There is a form at the bottom of the page that allows the admin master to add a new admin. He fills out the new admins first and last name and email. It checks to make sure the email in the pending admin table is not already in there, and to make sure this admin is not already in the system. If any of that is true, it shows an error message. If it is not true, it shows a success message, inserts the admin into the pending admin table that contains their email, and the GUID code that I generate. I then use the email template for inviting admins to send an email to the added admin. There is a link that binds the GUID code and email to the URL of the verify admin page inside the email. The link when clicked by the added email redirects them to the verification page. I then query the URL parameters, look up in my pending admin table to make sure the email and code both match a record in the table, if they both match, I show the registration form for the admin to fill out. If they don’t match, I assume it is a security attempt and I show an error message saying unauthorized access. The registration form takes the admin’s first and last name as well as a password and confirm password field. If the information is all valid, I insert them into the admin table and show a success message that allows them to click a link redirecting them to the admin login page.

Master Page:

The master page in .NET is a page that every page in the application inherits. What this means is, the master page is the default template or style of the application. The nav bar and footer are contained in the master page so that every single page also has it. The logic behind the master page handles the cookies and session ids. It also handles what nav bar links to show based on what user is logged in. This way, I can adapt to the different type of user who is logged in which is either a student or an admin at the base level. I can show the nav-bar links for students to see when they’re logged in and vice versa for admin.

Security:

User types are student, group member, group master, admin, and admin master. These are the roles throughout the application and on each page when the page is first being loaded, I used these session variables to see what to show and what not to show. The session id is the most important security vulnerability in this application. Since it is the biggest vulnerability, I’ve made it the most secure. .NET automatically encrypts any session id or session cookies being created. I use the student’s id in the database as a session id but the encryption is very strong, and the session id is fairly unique in the database to make it almost impossible to crack. Every single page I check to see if the user’s id contains a valid id in the database. This ensures that only logged in users can see certain pages because their session id on the server is created once they log in. I also always check the student’s type meaning if they are group member or group master to ensure they can only see certain pages as well. Group members can’t see their group edit page due to this, only group masters can. The same security is implemented with the administrator logic. Admins have session id’s they cannot access the student portion of the application; they can only access the management portion. Also, the admin master can only handle adding admins, and removing admins, a normal admin could never access the admin management page due to their role in the system. I added cross site scripting checks for every single input ensuring that it isn’t possible. Thanks to .NET they detect any form of cross site scripting input, I raise that exception on the global page error function for every page and redirect the user to error page. I use stored procedures for every single piece of data or database interaction which is the most secure way to do any database transactions via a web application. This is because the input passed into the stored procedure is parameterized and treats any input as an actual value. If someone tried to put sql into the input, it would just cause an error because no record would ever be found with the given input because it is always going to be passed into a stored procedure. I never query my database directly from the application, the entity framework turns my stored procedures into functions that I can use in .NET. I hash my passwords with bcrypt and a random salt which is the most secure web practices for passwords. They are stored in the database. My GUID’s are all almost 16 characters in length except for the password recovery portion, and a GUID is the most secure form of generating a unique code. I use these throughout the application when dealing with invites, request to joins, signing up, pretty much any activity that could be potentially vulnerable to any kind of attack. The admin logic for adding admin is handled in the most secure way I can implement. This especially being true when adding a new admin into the system. My web config file contains encrypted api keys for my smtp server, and other sensitive information which is always the best practice because you do not want these values in plain text in the code for the website. When implementing Azure, I set up a key vault to store these secrets and passwords in my web config file that are encrypted and accessed through the key vault. Once I would deploy, I would configure the key vault with the web config file. I also use HTTPs which is the most secure network protocol for web requests. Another thing to note too, is that I use try’s and catch blocks through every single piece of code to always catch any exceptions or errors that could possibly be unintended. I sent this exception message to my email as the system administrator to always have a log on what errors may occur.

Database:

My dataset is SQL Server. There are 4 tables, the admin table, student table, pending admin table, and group table. The schema is called studentInfoDB and it has all the proper constraints added to every single column in each table to ensure good database integrity. There are stored procedures that are used for .NET to access. Passwords are hashed inside any table that contains passwords, and the relationship between students and group are properly defined where it is a 1 or many students to 1 or 0 groups. This means one or many students can be in only 1 or 0 groups ensuring that a student can either not be in a group, or only be in one group. I’ve added users in the database as well, I am the root user who would be considered the DBA (database administrator), and I added a few tests users that would have the role as developers only being able to access views I’ve created and use the stored procedures I created so that they could work on the website.

**Maintenance issues:**

* One maintenance issue that someone may wish to adapt the code I’ve created for this application is to further separate some of the logic. Each page load function does similar things dealing with the session id’s and validating the user can access this page. You could further separate this by creating a class called Permissions, and just creating a permissions object and calling it in each page load function to get the user necessary permissions. This would help the maintainability when adding more and more pages to the application so that you don’t have to continue using the boiler plate same code I use for every page.
* Another maintenance issue would be for whoever has access to the database. They should be creating a backup of the database which I have not done, they should also modularize the stored procedures and organize them since they’re all in the same folder. There’s about 20-30 of them and they’re all in the same folder. They could separate the stored procedures into a student folder, group folder, and admin folder. Also, they could fix some code throughout that calls multiple stored procedures repeatedly throughout the application. It usually is better to not call multiple stored procedures on an event in .NET because it can make things slower. Therefore, I’d go back to each page, see where I call multiple stored procedures and try to combine them into one procedure and get all the data necessary from a single procedure. I know there are some cases where I get the students group information from one procedure, then also use another to get the students from that group. This could be combined to get all students from that group and the group information.

**How to access**

Right now, the website is not up and running on a production server. I planned on hosting through Azure, and if I get to it, I will name the site msmaryconnect.com. This will be domain for the site and that is how you will be able to access it in the future.