Basic Banking Corporation
Web Application – Report

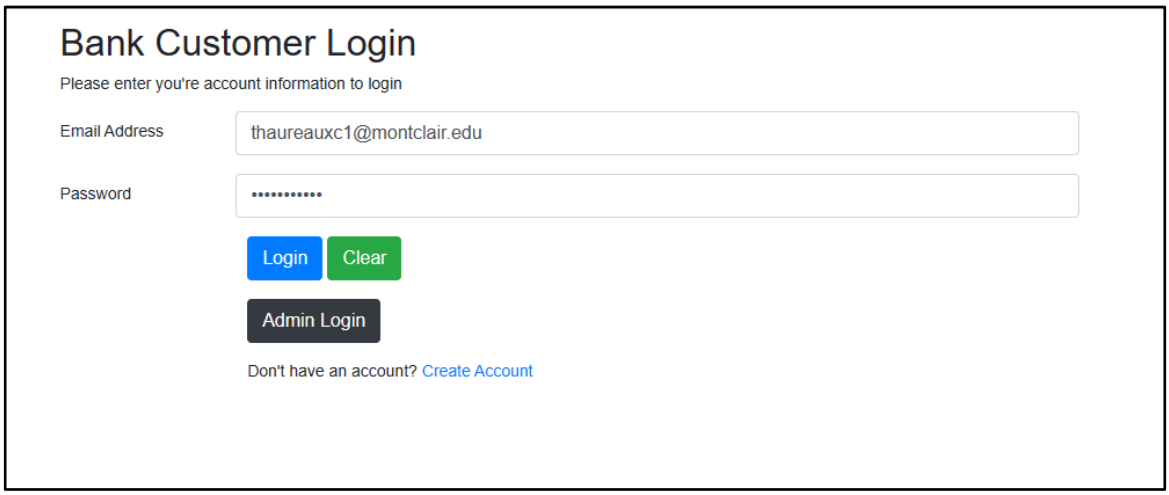
Catherine Thaureaux, Cassie Ihekwaba,
Jared Kruegel, Chris Rojas, Joe Chalet
CSIT415_03 Software Engineering II
Instructor: Dr. Kazi Zakia Sultana
Date Submission: 4/10/2023

Table of Contents

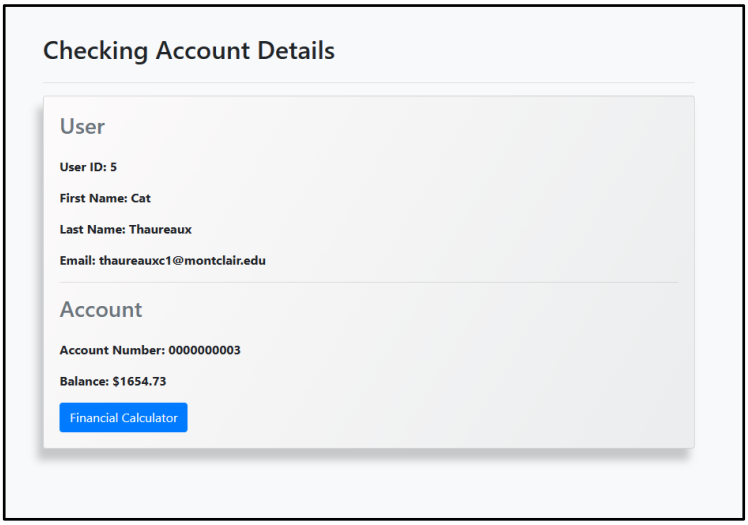
I. Functional Requirements:

1. User Side Functional Requirements:

Login Page



Post Form Submission (Correct Credentials):



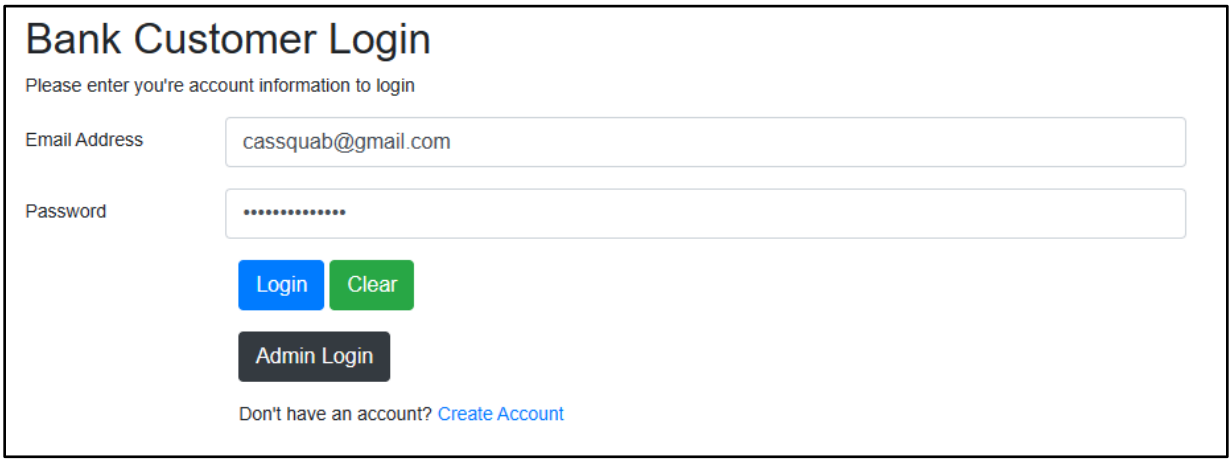
To implement this login function, we had to have a type of form validation that would take in both the username and password and compare it to what was already existing in the database. Upon form submission for login, the user email is first checked against what is in the database to find the row associated with it. This row will be the user

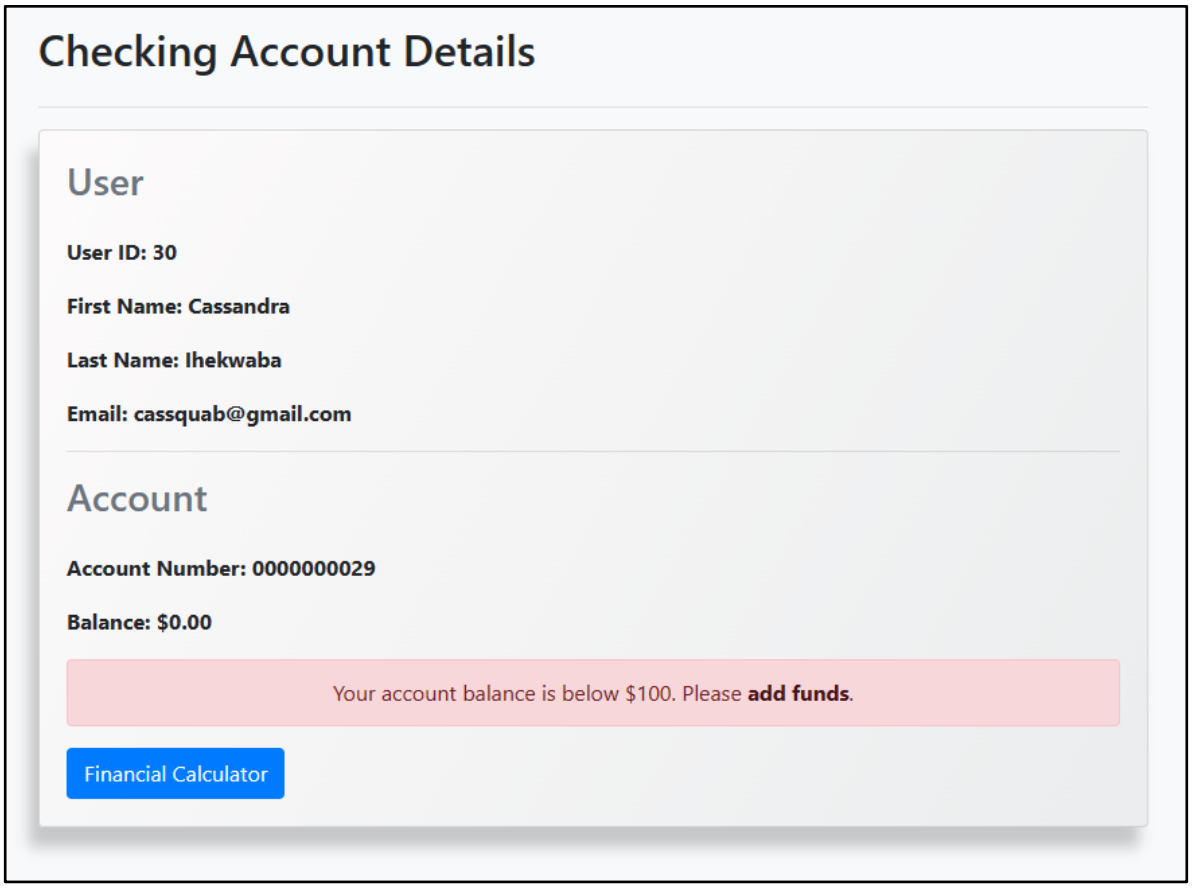
account that is checked. For security, anything that uses a variable that was entered by the user is prepared, then bound to a certain parameter to be sent to the database to be run.

Sign Up

Upon clicking user login, we look under the buttons to find a blue hyperlink that takes us to user registration:

Successful Sign Up (Login Redirection & User Page):





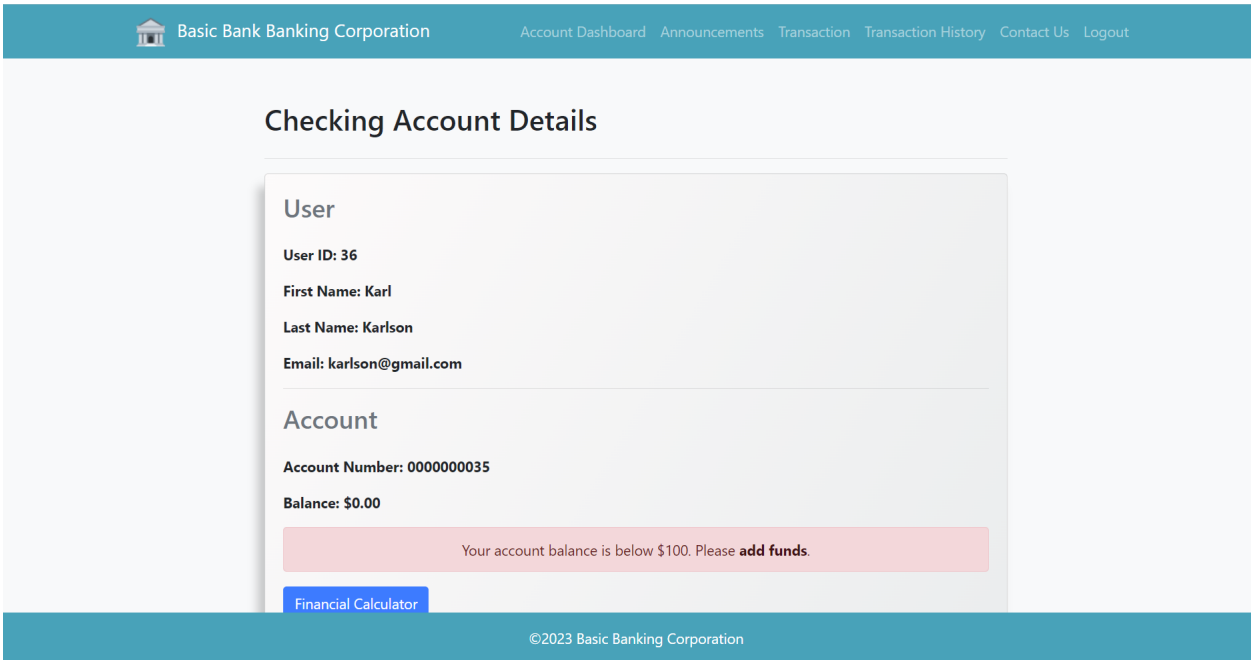
The implementation of this page is about the same as login, it just inserts these values into the database, rather than selecting them. Upon passing all the checks for user information, it will redirect you to the main login page, and you can login with your new credentials and start using the account, going through the same process of validation and session creation as was explained in logging in.

For form submission, minus not entering anything for the user email or password, there is a check in place for emails and passwords to ensure they are actually proper. The email field checks your email to ensure it has only letters, numbers, or underscores - any special characters are not permitted, and the password will only accept a password over 6 characters, and contain at least one number, letter, and special character:

This was a pretty basic modification - when the form is submitted and user entries are checked, there is a certain pattern that these entries are checked against in the **“preg_match()”** function which just checks to see if the user entry has any of the required/permitted characters. Both email and password have slightly different strings used to check this within **preg_match()**.

User Dashboard

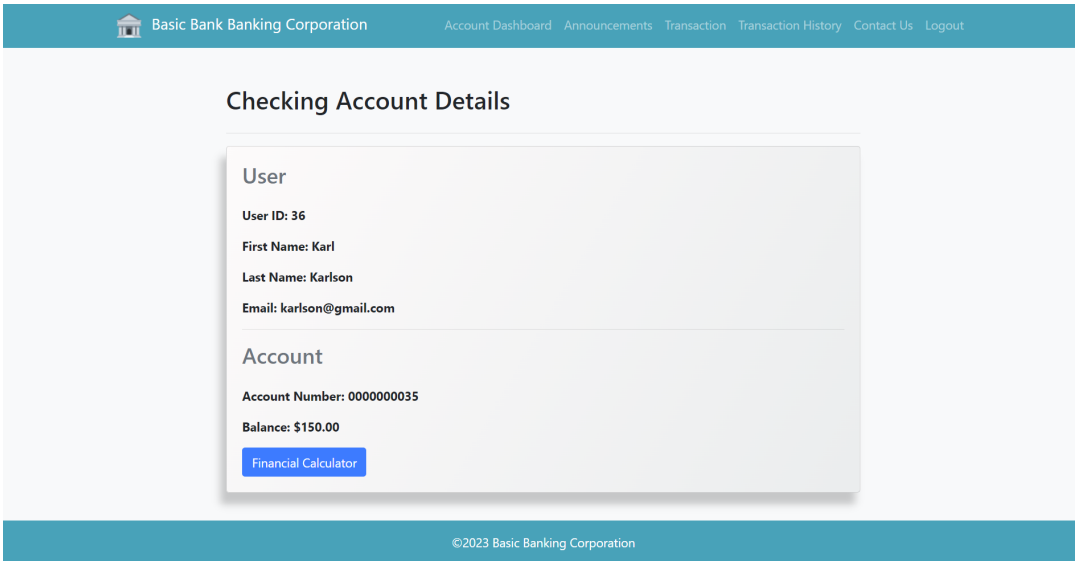
Once you have logged in to your user account, you will be directed to the user’s homepage, the “user dashboard”.



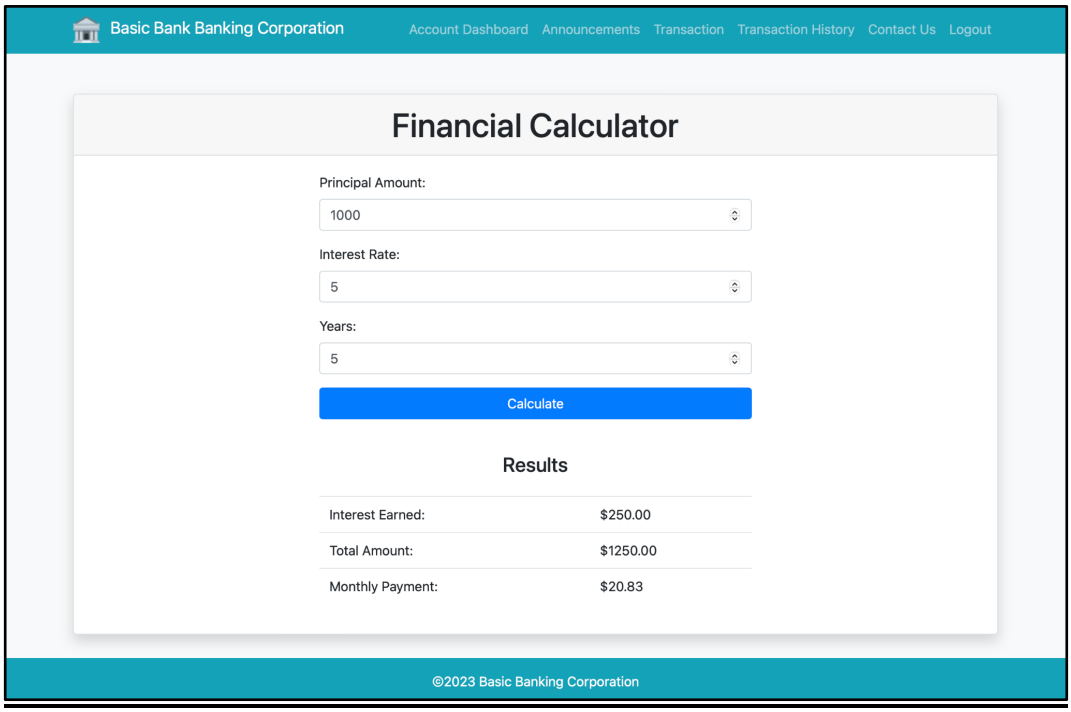
This page presents you with all of the information that you will need. Under the user section it presents your user ID, first name, last name, and email address. Below that in the account section it presents your account number and your account balance.

Since this is a new account, it starts with no money in the account balance. Note that a red warning message shows up at the bottom when the account balance is less than $100. Transactions will be further covered in the transaction section, so the

following image simply shows you what happens once you've deposited 100 or more dollars into your account.



Financial Calculator



On the homepage, there is an option to open a financial calculator. Once opened, it takes you to a new page where the user is required to enter the amount,

interest rate, and time (in years). The calculation for this calculator were done with the following equation:

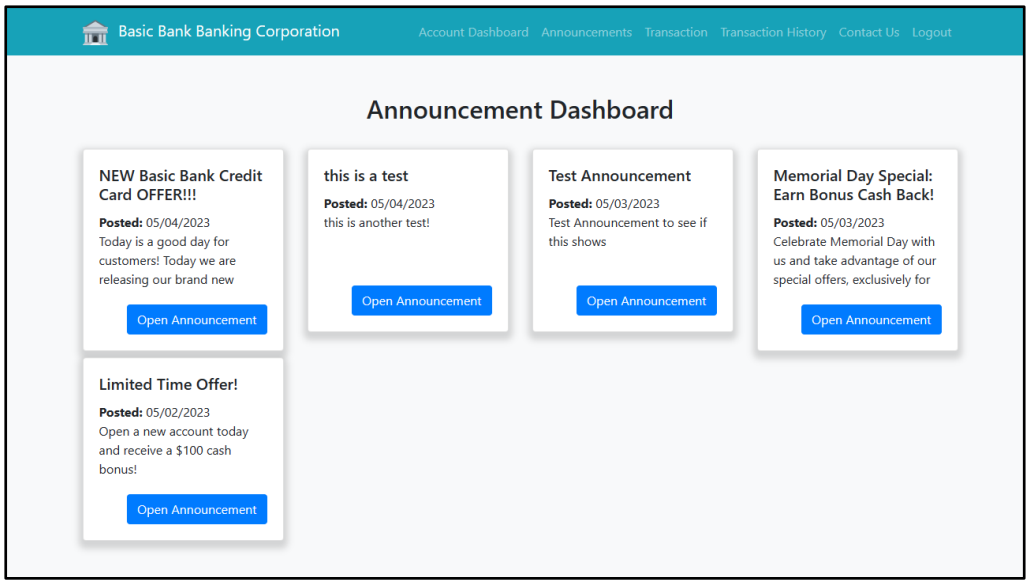
I = PRT

- I = interest
- P = principal amount
- T = number of years held in investment

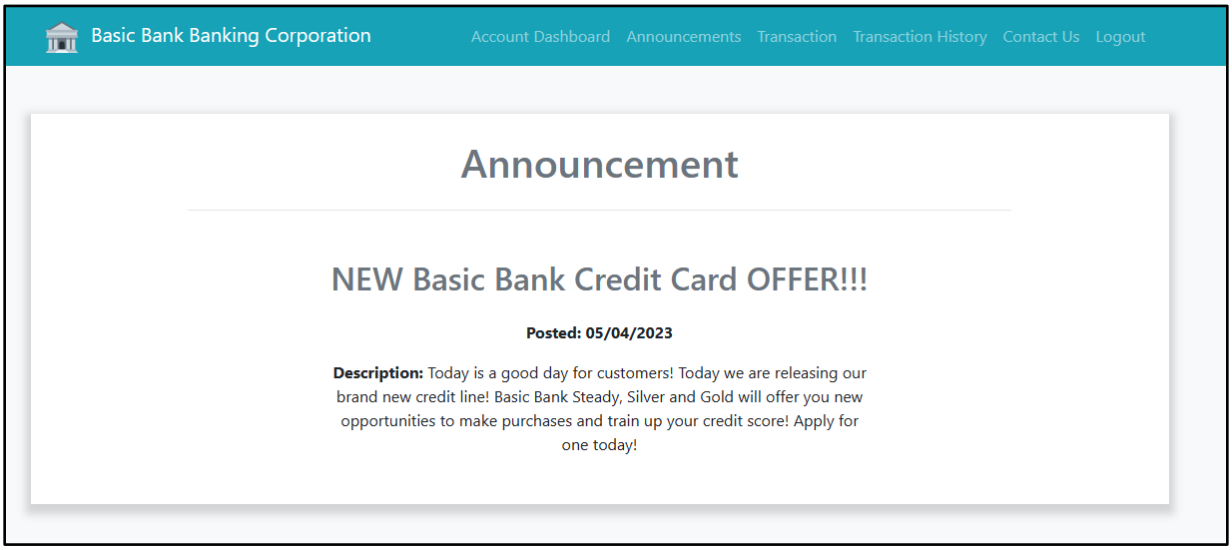
The financial calculator is currently at its most basic form. In the future, we plan on implementing more features for a user's accessibility. This would include giving more data like customers ending balance after inflation or graphic demographics (like charts). We would also like for users to input more data, this would include adding this like giving the user a wider option for compound type like monthly, quarterly, annually (instead of having it only per year)

Check and Open Announcements

Entering Announcements Dashboard:



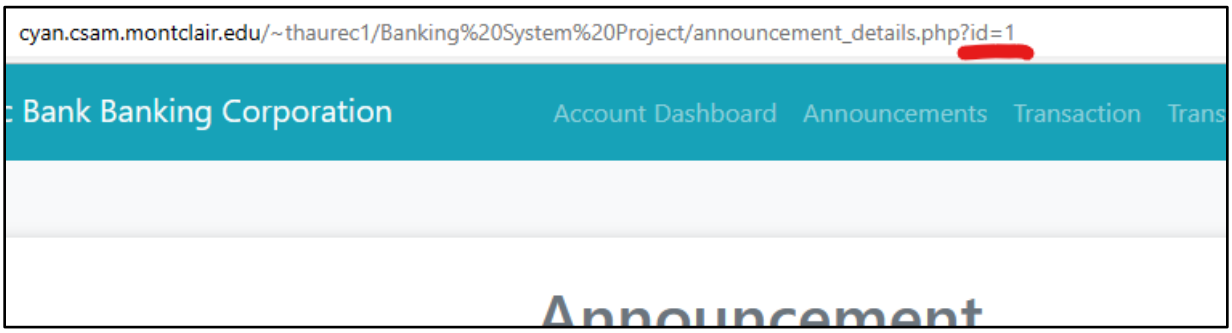
Redirect after clicking an announcement to view:



For this section, this was an easy modification to the code - at base level - this code would pull all of the announcements from the announcement table and list them all on the announcements page for each user. From here, we can access any of the posted announcements. How this works is that we select everything from the table, and all this information is then stored into an array called "**data[ ]**" that grows in size to accommodate each new entry that can be found. Next, we make a basic template of how each announcement should be displayed in the card format. This formatting is put into a foreach loop that prints everything in the array into its own card on the main page. Every new announcement added expands the array, and is then added onto the main page.

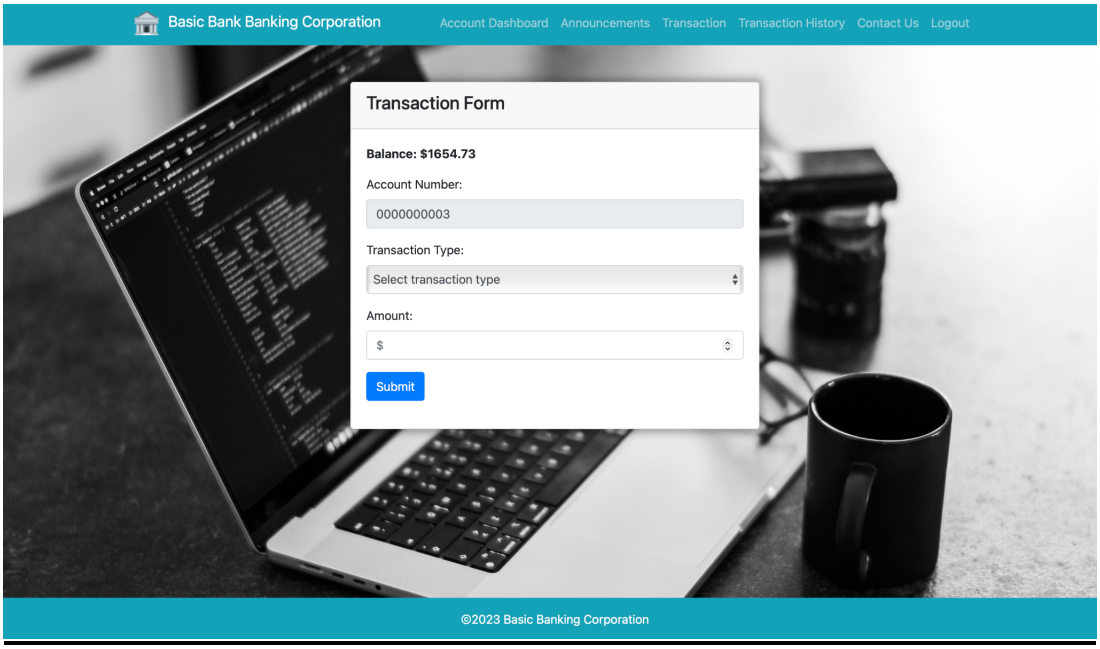
To view the individual announcement, there is a button on the card that takes the user to a display page that shows the announcement in full, along with information like the title and timestamp. Once this button is pressed, an array called "**announcement[ ]**" is made that holds the title, date posted, description and id of the announcement that were taken from the previous page. Another array called "**data[ ]**" is made to hold the details of this

information. The id from the announcement array is then used to fetch the necessary row that has the associated information that we need (this is also reflected in the URL when you click an announcement; with the id value showing up at the end:



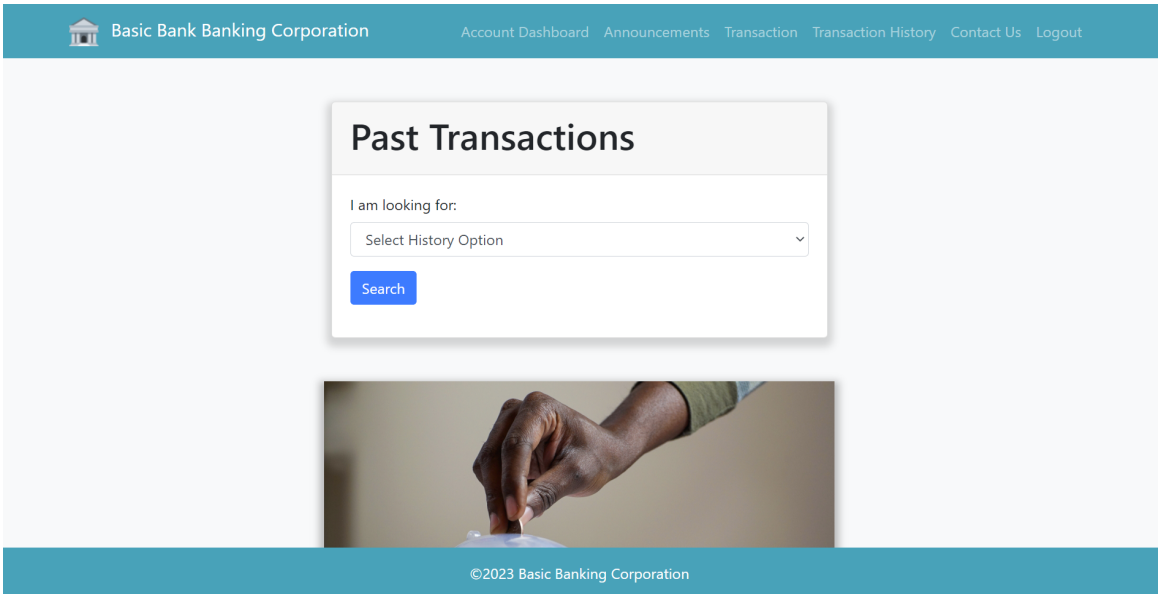
This id is used in a select statement that gets everything from the table that has the matching id number. From there, all the information for that matching ID is pulled into **data[ ]**, and then displayed on the page.

Make Transaction

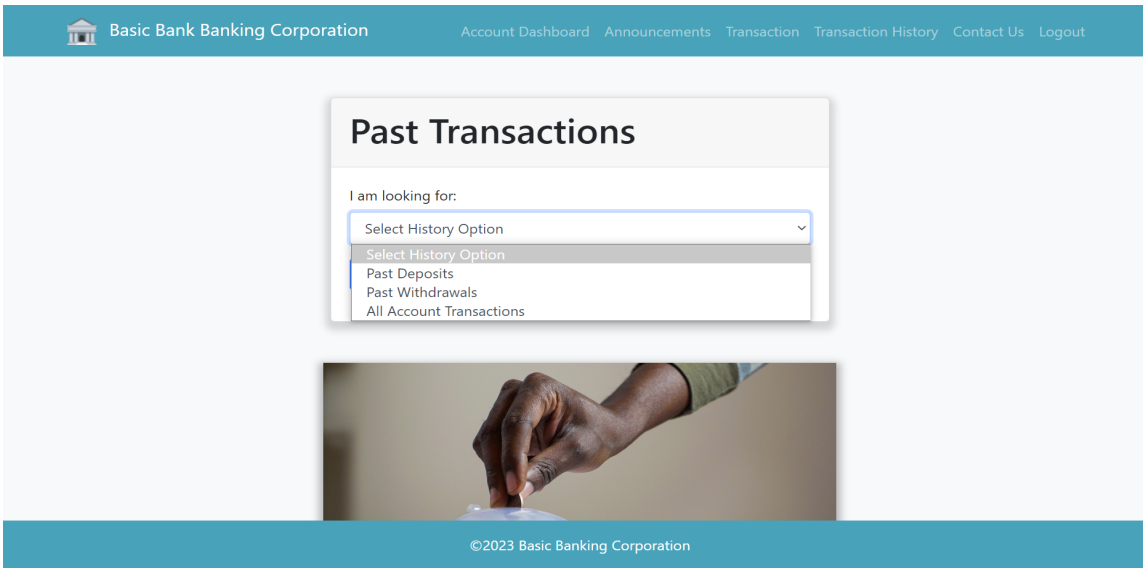


On a separate page on the nav bar, Make Transaction is designed for customers to deposit or withdraw money from their personal account. For a deposit, a user must select the Transaction Type to “Deposit” provided by the drop down menu. Below the amount you would like to deposit must be entered. On the top of the form there should be a current balance to the account. Once the submit button is entered, the amount requested should subtract from the current balance. Same goes with the “Withdraw” feature, only this time the amount entered will add to the accounts balance. Some limitations this section has is not withdrawing more than you have, inputting numbers passing the hundreds place, or inputting a negative quantity (for both withdrawals and deposit).

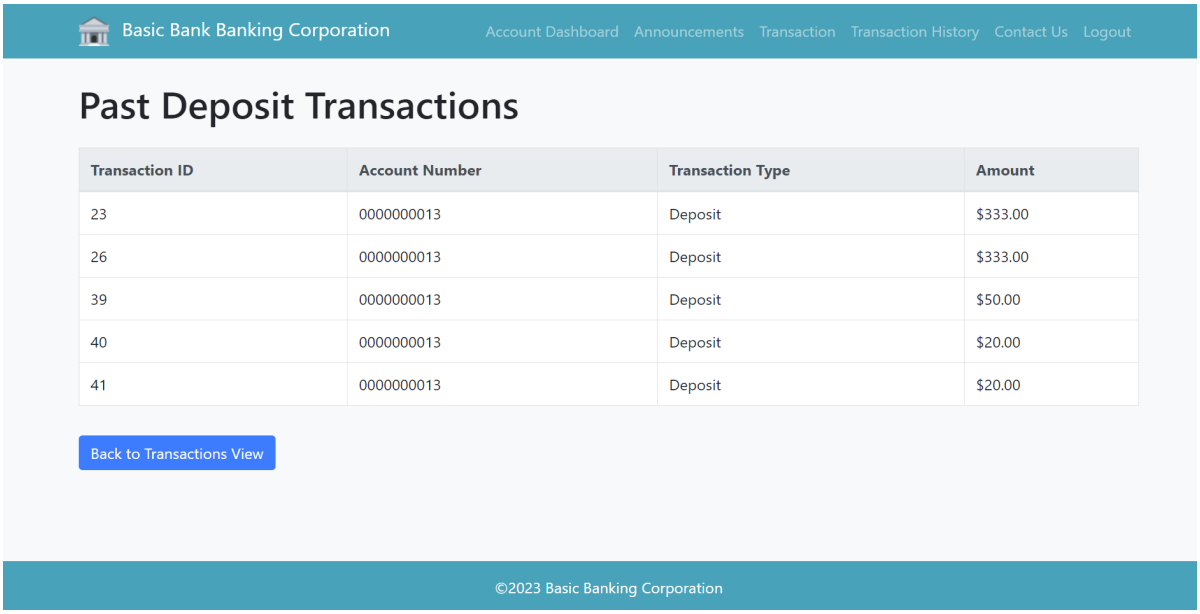
Transaction History



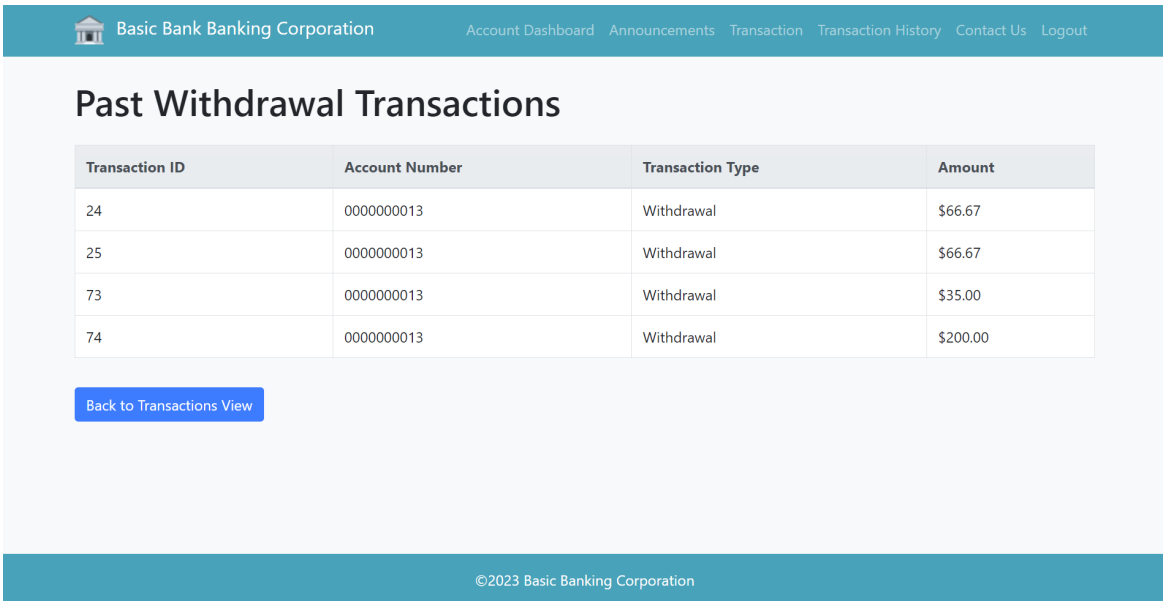
On the navbar you can see a link called “Transaction History”. When you click it you are sent to the above picture. From here you can click the drop down menu, seen below.



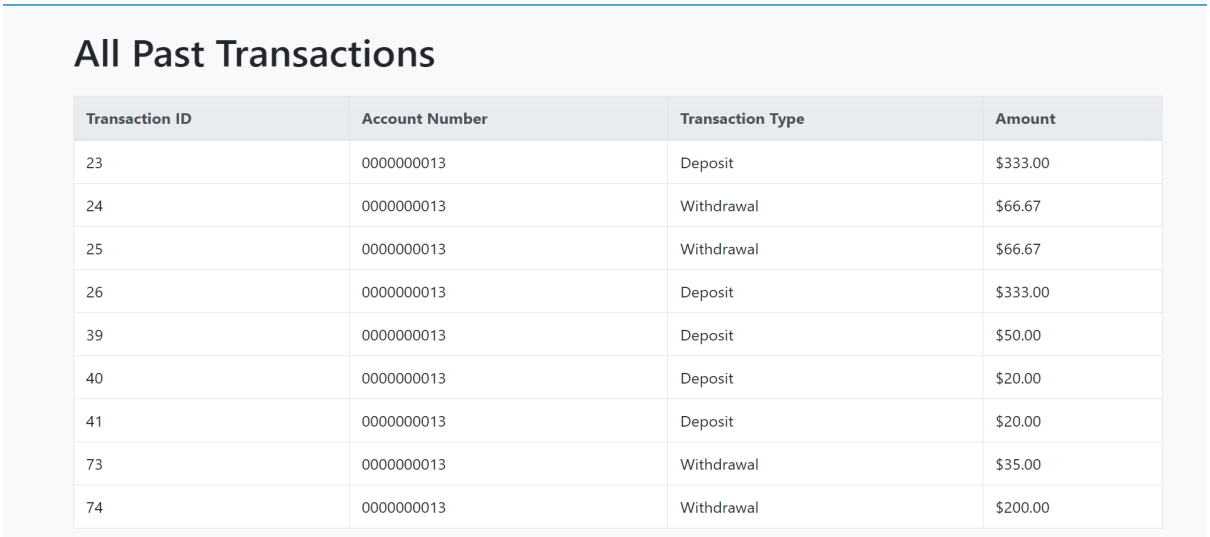
This dropdown menu gives you three options. You can see all the withdrawals connected to your account, all the deposits connected to your account, or you can choose to see both in one list.



In the above photo you can see all the deposit-type transactions on a given account. In the following photo you can see the withdrawals on the same account.



And finally, the third option allows you to see both at once.

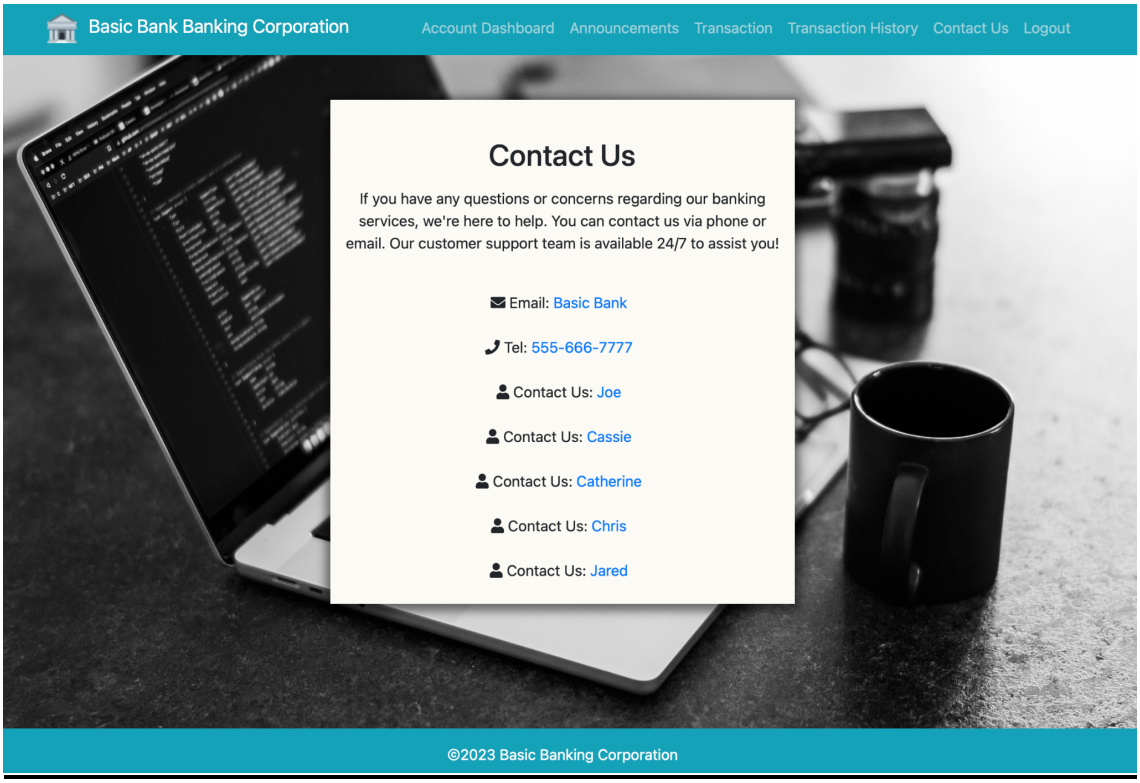


Note that the lowest transaction ID is at the top and the highest is at the bottom, so this list essentially shows the transactions from the oldest at the top of the list to the newest at the bottom of the list.

The “transaction history” section actually represents two pages. The first is where you choose between the three options on “historyview.php” which takes your choice and sends it to the second page, “results.php”. Based on your choice, the second page will

query the database for the transactions connected to your user account by your user ID and of the desired transaction type(s). Then it simply outputs the received transactions downwards in a table.

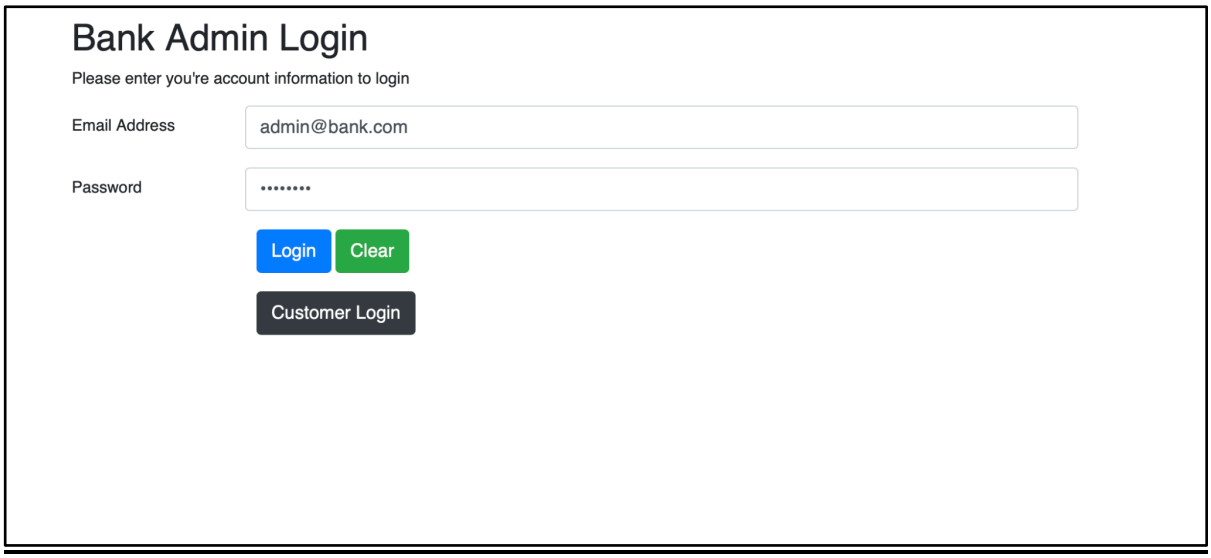
Contact Page



This page is simple and represents how on a company's website, the website will often provide a page where you can contact them and provide feedback. In our case, we provide a link to a dummy company email, a dummy telephone number, and each of our five Montclair school emails.

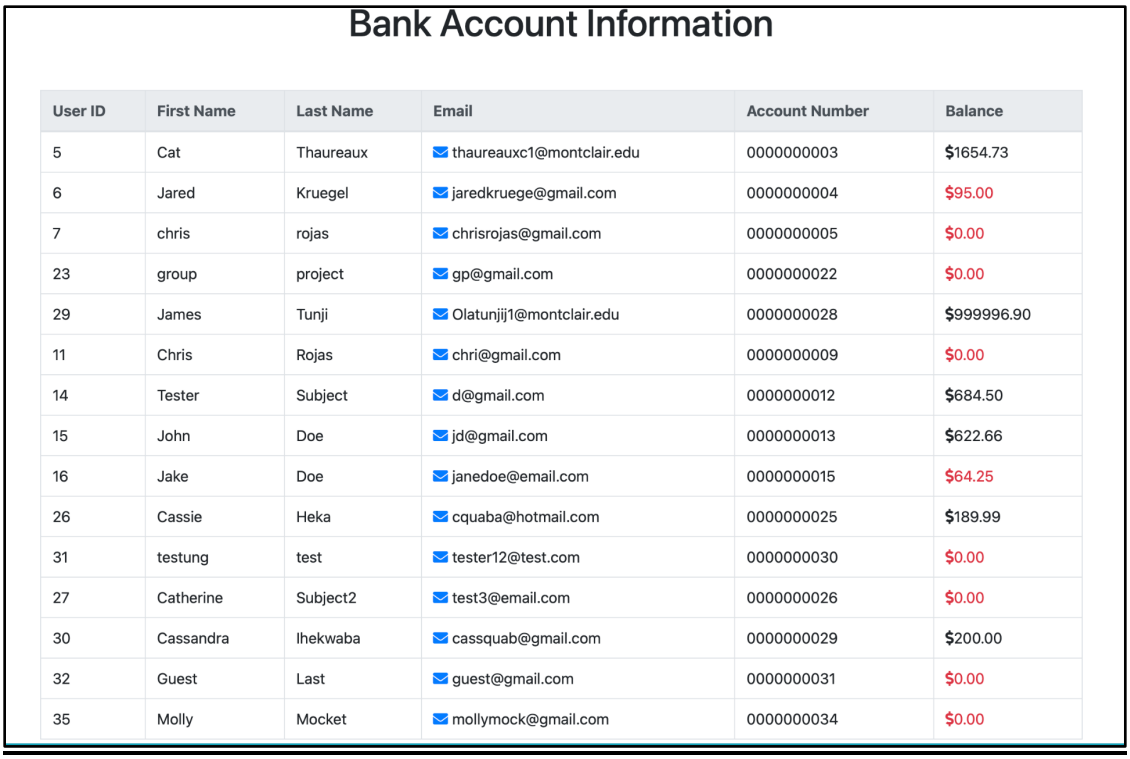
2. Admin Side Functional Requirements:

Login Admin Side



The login on the admin side of our website functionally isn't very different to the user side login however there are a few exceptions that do exist. One such exemption is that you cannot register an admin nor create an admin account anywhere on the website. An admin account must be created and injected into the database manually from MySQL. Also, when logging in as an admin there is another token that gets set that users cannot get themselves, this token identifies the user as an admin and lets the admin access pages where that token is required to view. In a future implementation, we can include a functionality on the admin side to create another admin account, simulating “hiring” an administrator to overlook the user side of the bank.

View Bank Accounts

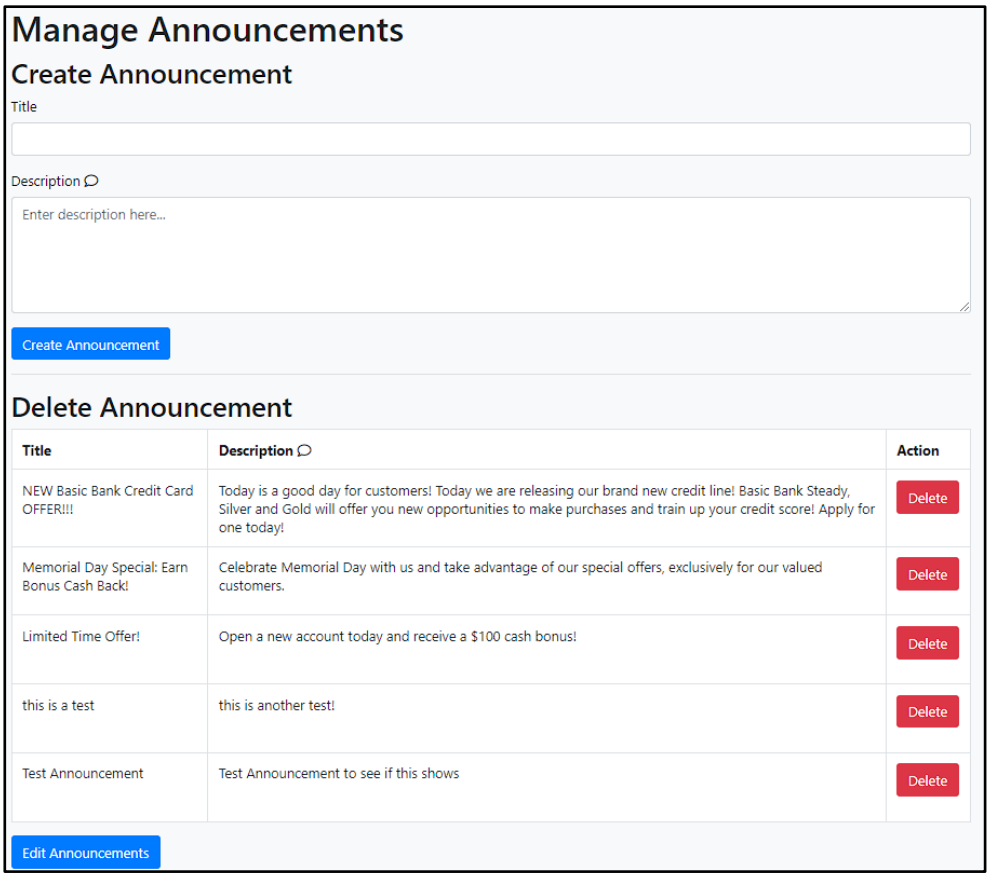


Bank Account Information

User ID	First Name	Last Name	Email	Account Number	Balance
5	Cat	Thaureaux	thaureauxc1@montclair.edu	0000000003	$1654.73
6	Jared	Kruegel	jaredkruege@gmail.com	0000000004	$95.00
7	chris	rojas	chrisrojas@gmail.com	0000000005	$0.00
23	group	project	gp@gmail.com	0000000022	$0.00
29	James	Tunji	Olatunjij1@montclair.edu	0000000028	$999996.90
11	Chris	Rojas	chri@gmail.com	0000000009	$0.00
14	Tester	Subject	d@gmail.com	0000000012	$684.50
15	John	Doe	jd@gmail.com	0000000013	$622.66
16	Jake	Doe	janedoe@email.com	0000000015	$64.25
26	Cassie	Heka	cquaba@hotmail.com	0000000025	$189.99
31	testung	test	tester12@test.com	0000000030	$0.00
27	Catherine	Subject2	test3@email.com	0000000026	$0.00
30	Cassandra	Ihekwaba	cassquab@gmail.com	0000000029	$200.00
32	Guest	Last	guest@gmail.com	0000000031	$0.00
35	Molly	Mocket	mollymock@gmail.com	0000000034	$0.00

View Bank Accounts was specifically designed for the admin to view all users on the platform. Although this page doesn't require admin to input information, the website does output the user's ID number, first and last name, email, account number, and the current balance they hold. The page has a function for the email, where each user's email can be clicked on and redirected to your email application to quickly send any emails, if needed. Every time a user is added, the bank account information will be refreshed and load the current users. This goes the same with deleting any users. A neat feature implemented in the design is highlighting all balances below $100 dollars, as an indicator for a low funded account. This allows for better usability, offering a visual depiction of the user's current balance.

View, Add, Edit, and Delete Announcements

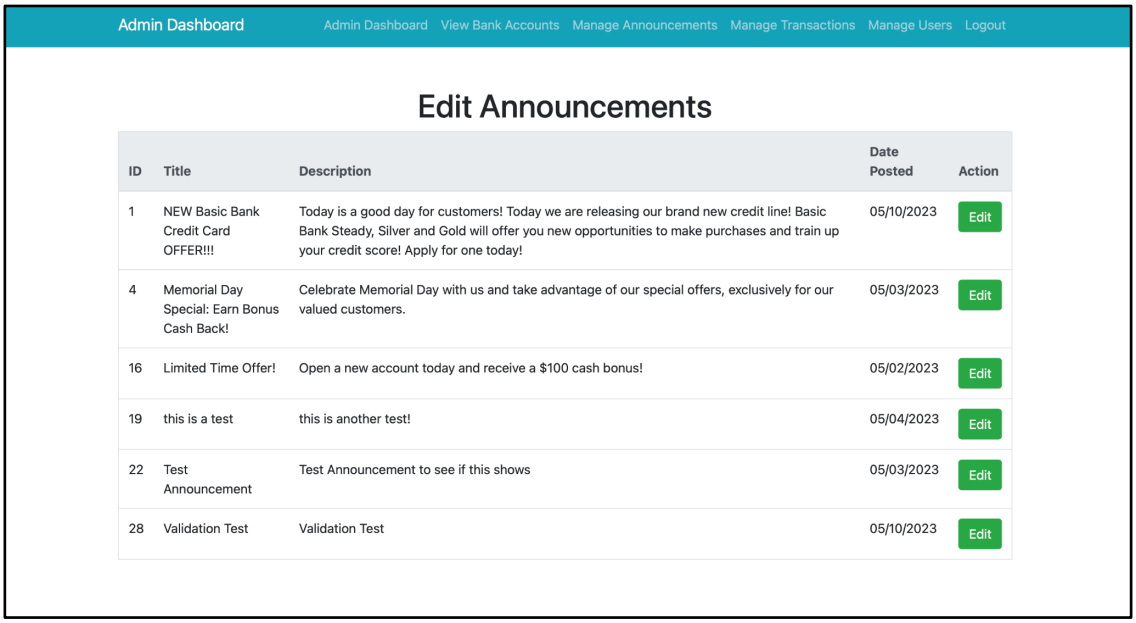


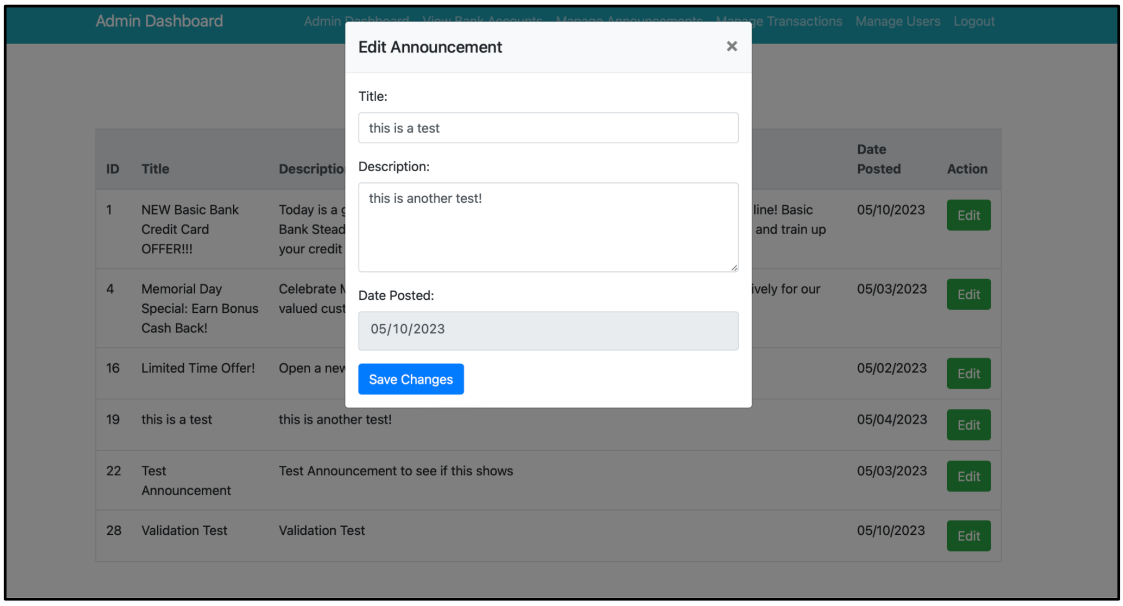
From the screen shots above and below, you can clearly see that there are 4 options available for the admin:
Create announcements - This requires the two fields above the list of announcements to be filled, both a title and description are required for the announcement to be created. Since the announcementID is auto incremented in the database, this will automatically occur when the user creates an announcement.

Delete announcements - If you select the delete button on the right side of any given entry in the list it will then delete the entry therefore removing it from the list and the back end table where it would have resided. The page will also be automatically refreshed after the admin deletes the user, removing the table row on the bottom of the page.

View announcements - The admin is also able to view all current announcements that are stored in the database as well as presented on the bank user's side of the website.

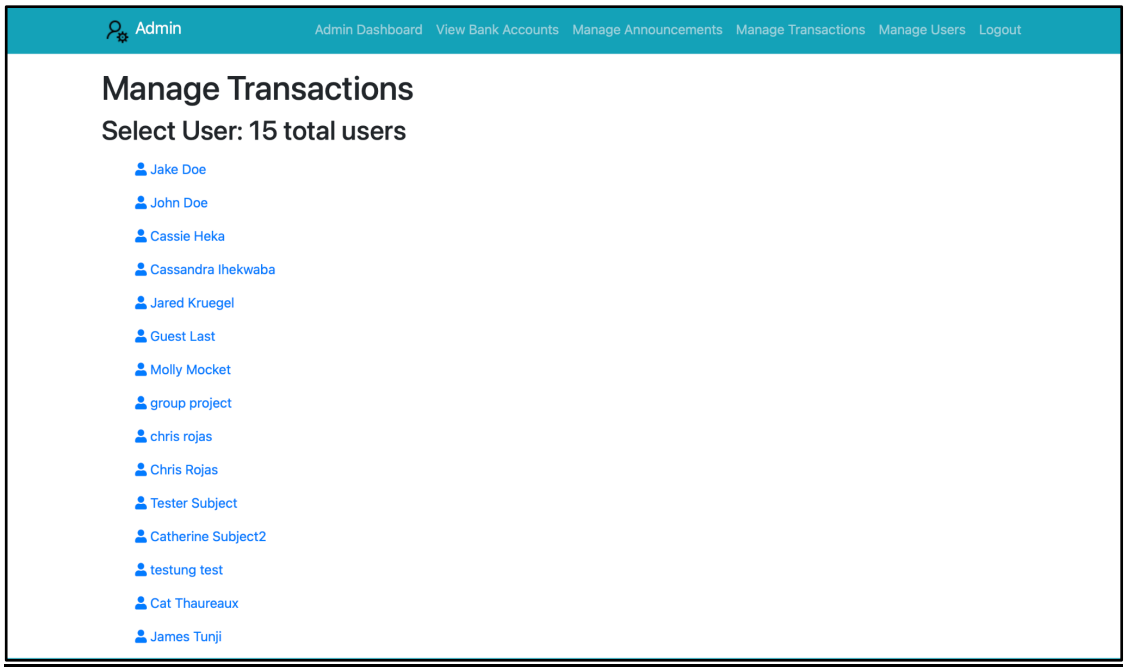
The title of the announcement as well as the description of the announcement is displayed.

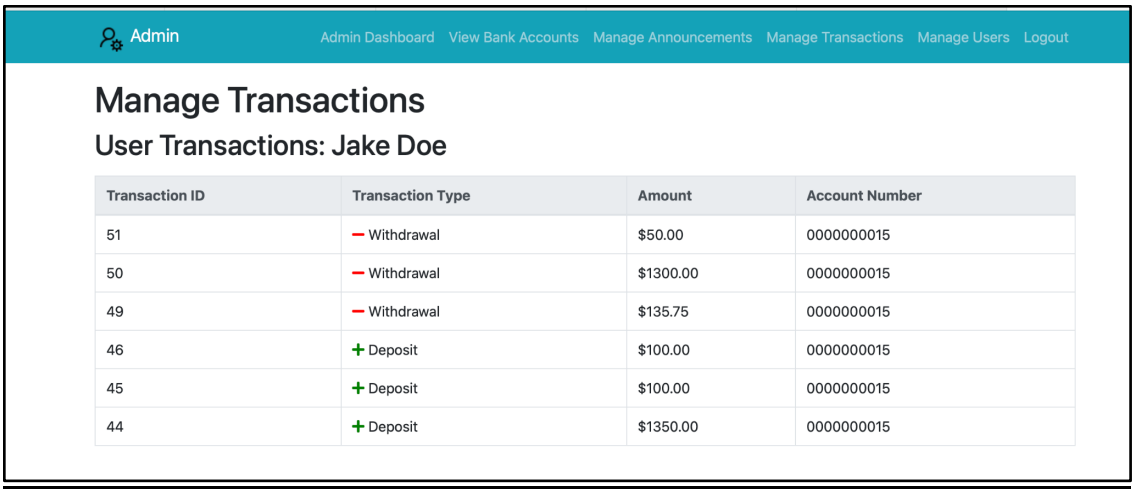




Edit announcements - This allows you to edit the announcements shown in the list. When the admin clicks the edit button next to a. announcement, a bootstrap modal will pop up with two editable input fields. The date field will automatically have the current date when editing the announcement, and the field is read-only. Logically, it made more sense to implement it this way, rather than allowing a user to input any date. The announcementID will also remain the same when editing since it is a unique, primary key, so it does not need to be included in the modal.

View User Transactions

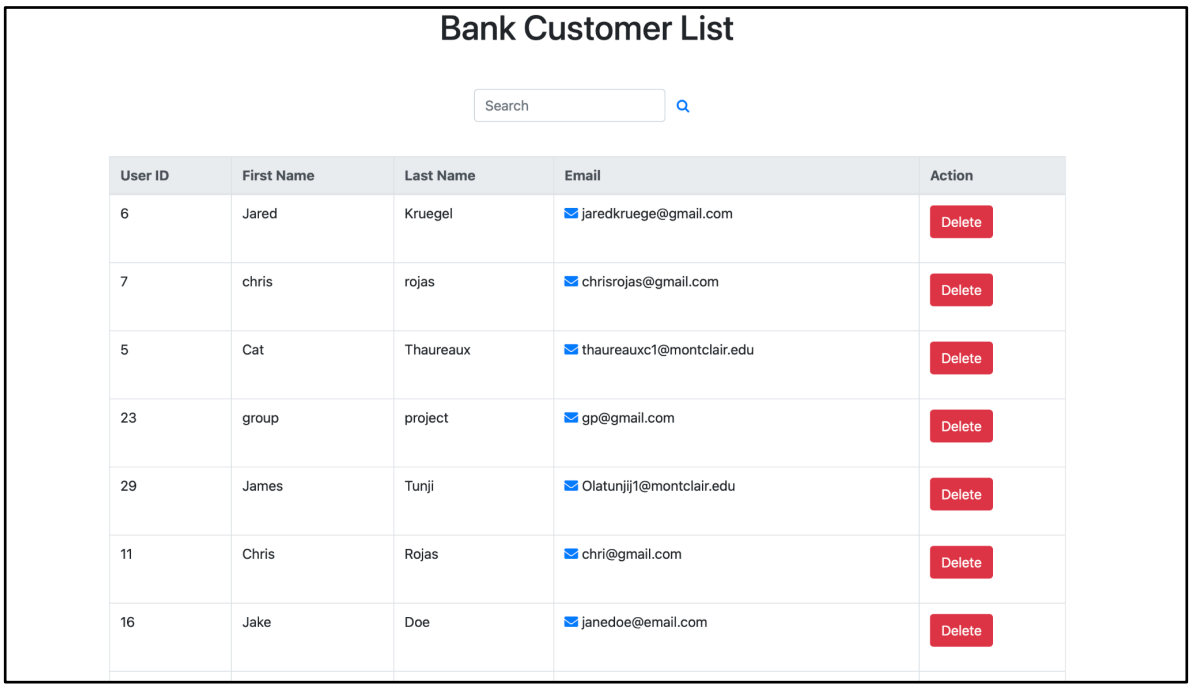


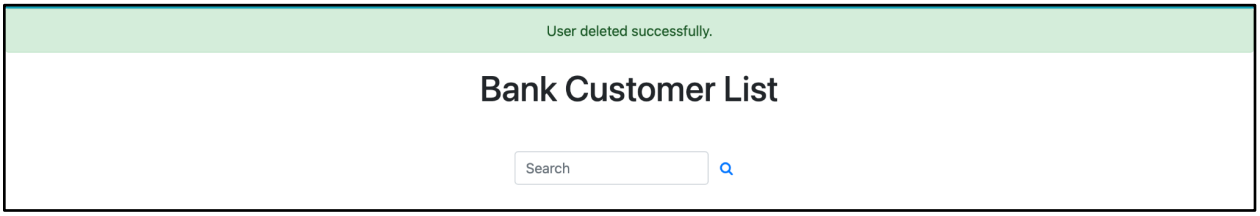


Manage Transactions, an Admin feature, is designed to view the signed up user's history solely based on deposits and withdrawals. When we open the page up, we are led to a list of all possible users, each with embedded links when clicked on. If we want to see more details on a specific user, the admin must click on the user. The current page will then automatically reload, displaying the user's name, and below will have a table that shows the Transaction ID, Transaction Type, Amount (held at the

time), and Account Number. The admin is only able to view the user's history; no input or changes from the admin can be done on this page. A **green +** or **red -** depicts whether money was added to the account in the form of a deposit, or subtracted from an account, depicting a withdrawal. For future implementations, we can include filtering the user's past transactions based on deposit, withdrawal, and all transactions. Having a filtering option will be useful when there are hundreds to thousands of users signed up for a bank account, creating many different kinds of transactions.

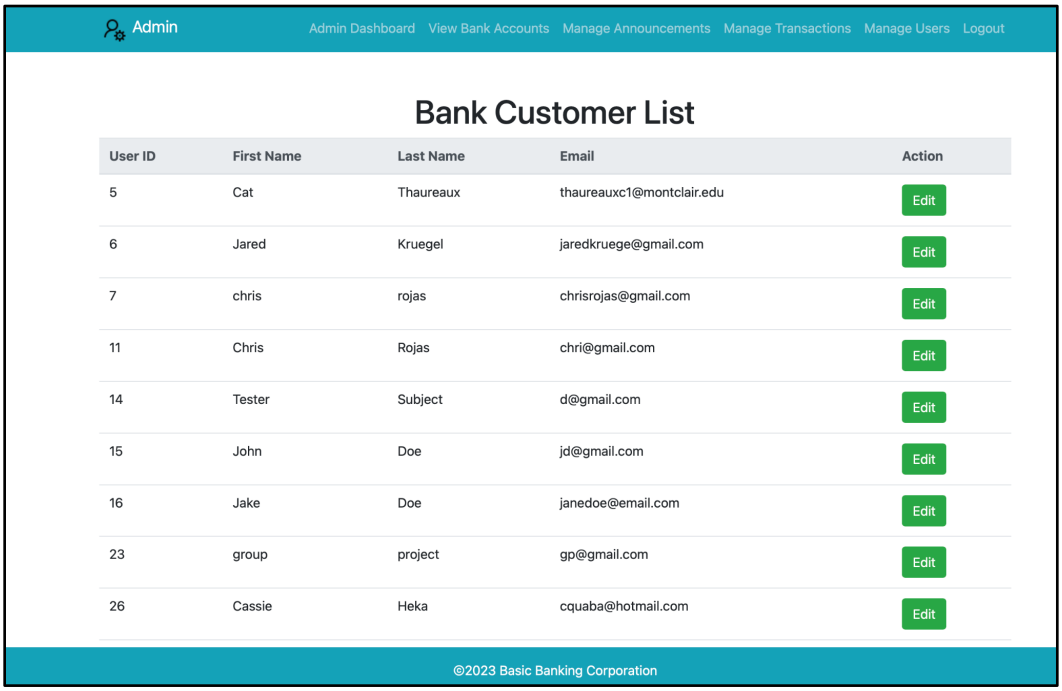
View, Add, Edit, Delete User Subscriptions:

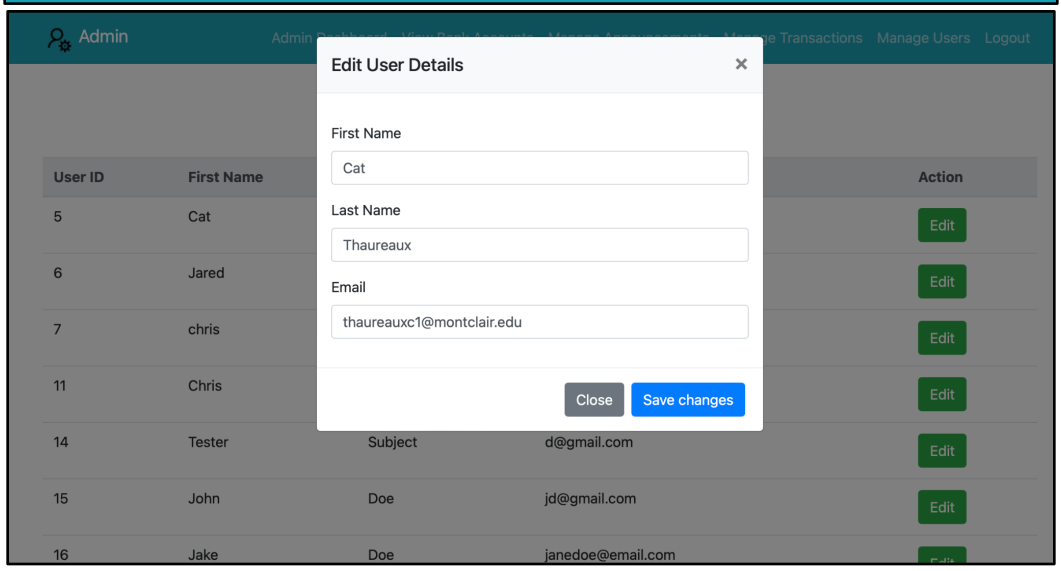




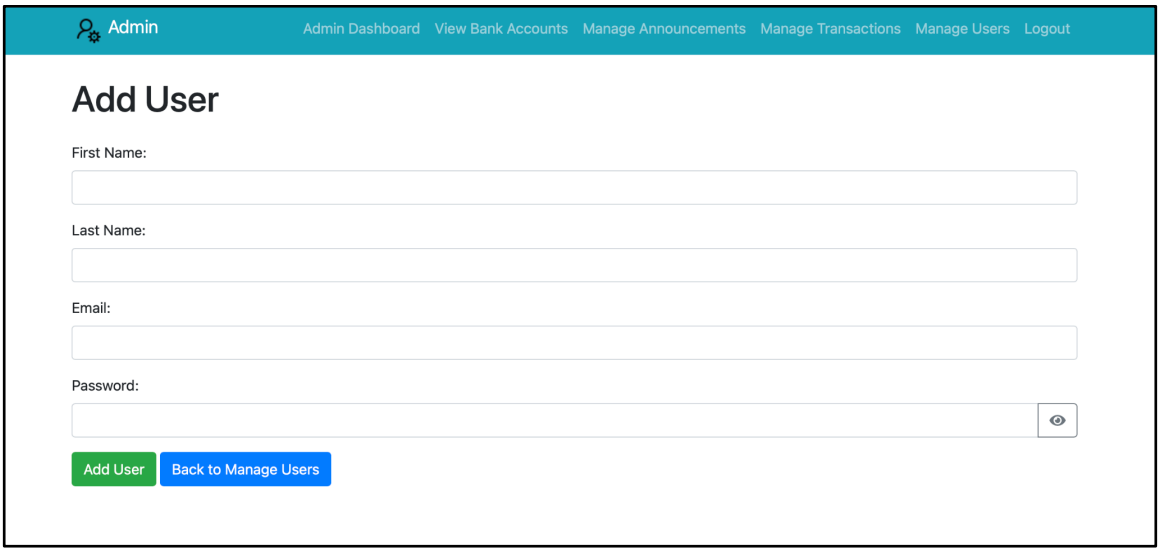
View User Subscriptions: This page allows the admin to view all user subscriptions for the bank web application. A search bar is also available to filter through the users. This is a helpful feature especially when there are a large number of users registered with a bank account. The search bar filters through words that are like the first name, last name, and email of a registered user. Admins can also search numbers, such as the userID.

Delete User Subscriptions: The admin can also delete a user from the system and the database. This feature will also delete the associated bank account that is created when a user registers for an account. This user will no longer be able to log into their account and all of their information will be deleted.





Edit User Subscriptions: The admin can edit user subscriptions. A bootstrap modal will pop up and the admin can modify the user's first name, last name, and email address. An error message will pop up if the admin attempts to edit the user's email and change it to an email that is already registered for an account. The email field also makes sure that a correct email form is input.



Add User: This add user page offers the same functionality as the user registration page. The admin can enter a name, last name, email address, and password for a new user. Error messages will pop up if the admin attempts to create a user with an email that is already taken by another user. The password also offers the same complexity requirements, such as containing at least one alphabetic letter, one number, one special symbol, and containing at least 6 total characters. A JavaScript functionality offers a password reveal option when the admin clicks the eye icon on the right of the field. In a future implementation, when an admin creates a new user and the user attempts to login to their account, they will receive an message stating that they must change their password before continuing onto the account. This will offer additional security so that the admin does not have knowledge of the user's private password after the user is able to enter their account and create transactions.

II. Interface Requirements:

Application Type: Web-Based
Front-End Software: HTML/CSS, JavaScript, Bootstrap, jQuery, AJAX
Back-End Software: MySQL/phpMyAdmin
OS: Windows, Mac OS
Supported Browsers: Chrome, Safari, Microsoft Edge
Editor and Compiler Used: Microsoft VSCode

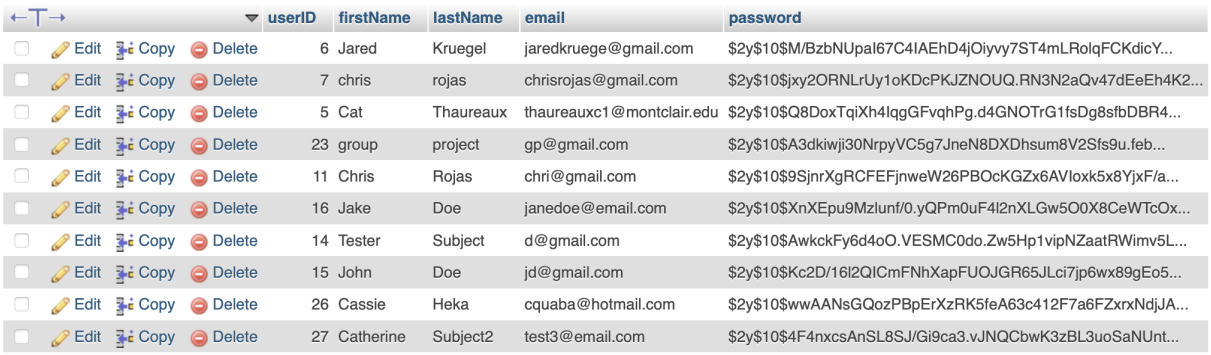
III. Non-Functional Requirements and Testing:

Since our group chose to implement a banking system, we decided that we needed to focus on security for our nonfunctional requirement. Throughout the system, we implemented different functions and techniques to prevent malicious attacks, protect private information, and validate all kinds of fields to make sure that the input is valid and sanitized. We also made sure that there would not be performance degradation by implementing a token in the transaction page that prevents multiple transactions to be sent to the server. Another non-functional requirement that we focused on was the UI/UX design of the site, making sure that it is both user friendly and offers a positive user experience for both banking customers and administrators. We incorporated bootstrap, CSS, JavaScript, jQuery, an icon library to make the site easy to navigate.

1. Hashing Passwords

For the password field, the passwords aren't displayed in plaintext as per requirements, we hashed these passwords with the bcrypt algorithm (in PHP for MySQL, this is

PASSWORD_DEFAULT), which posts a 60-character mixed string in the accounts table:



	userID	firstName	lastName	email	password
Edit Copy Delete	6	Jared	Kruegel	jaredkruege@gmail.com	$2y$10$M/BzbNUpaI67C4IAEhD4jOiyvy7ST4mLRolqFCKdicY...
Edit Copy Delete	7	chris	rojas	chrisrojas@gmail.com	$2y$10$jxy2ORNLrUy1oKDcPKJZNOUQ.RN3N2aQv47dEeEh4K2...
Edit Copy Delete	5	Cat	Thaureaux	thaureauxc1@montclair.edu	$2y$10$Q8DoxTqiXh4IqgGFvqhPg.d4GNOTrG1fsDg8sfbDBR4...
Edit Copy Delete	23	group	project	gp@gmail.com	$2y$10$A3dkiwji30NrpyVC5g7JneN8DXDhsum8V2Sfs9u.feb...
Edit Copy Delete	11	Chris	Rojas	chri@gmail.com	$2y$10$9SjnrXgRCFEFjnweW26PBOcKGZx6AVIoxk5x8YjxF/a...
Edit Copy Delete	16	Jake	Doe	janedoe@email.com	$2y$10$XnXEpu9Mzlunf/0.yQPm0uF4l2nXLGw5O0X8CeWTcOx...
Edit Copy Delete	14	Tester	Subject	d@gmail.com	$2y$10$AwkckFy6d4oO.VESMC0do.Zw5Hp1vipNZaatRWimv5L...
Edit Copy Delete	15	John	Doe	jd@gmail.com	$2y$10$Kc2D/16l2QICmFNhXapFUOJGR65JLci7jp6wx89gEo5...
Edit Copy Delete	26	Cassie	Heka	cquaba@hotmail.com	$2y$10$wwAANsGQozPBpErXzRK5feA63c412F7a6FZxrxNdjJA...
Edit Copy Delete	27	Catherine	Subject2	test3@email.com	$2y$10$4F4nxcsAnSL8SJ/Gi9ca3.vJNQCbwK3zBL3uoSaNUnt...

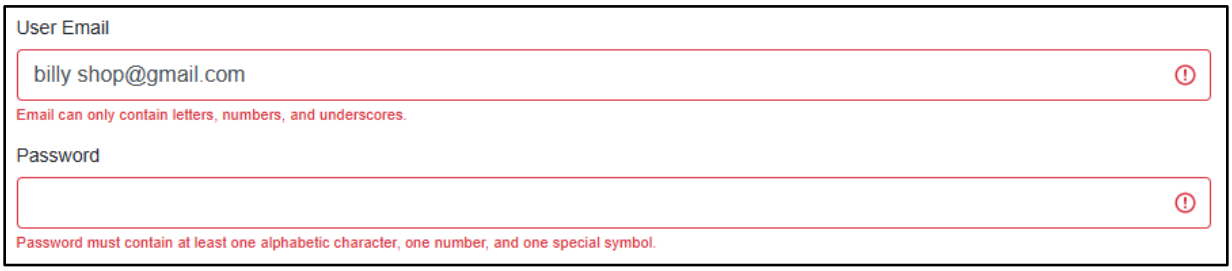
When the same user is logging in, the row with the matching user email is selected, and the password from that same row is compared in a function called “password_verify()”. We use this function to check the correctness of the password so the user can be permitted to enter their session. This function takes in the value that is already present in the database that is related to the email, and also hashes the user-entered password so that we can compare the hashes - if they match, the user is then redirected to their user page.

For sign up, the password is stored into a variable and hashed with the **“password_hash()”** function. Once hashed, it then goes into its row with matching user credentials. If the password entered does not match the password chosen when choosing your account password, you will be notified that it does not match:

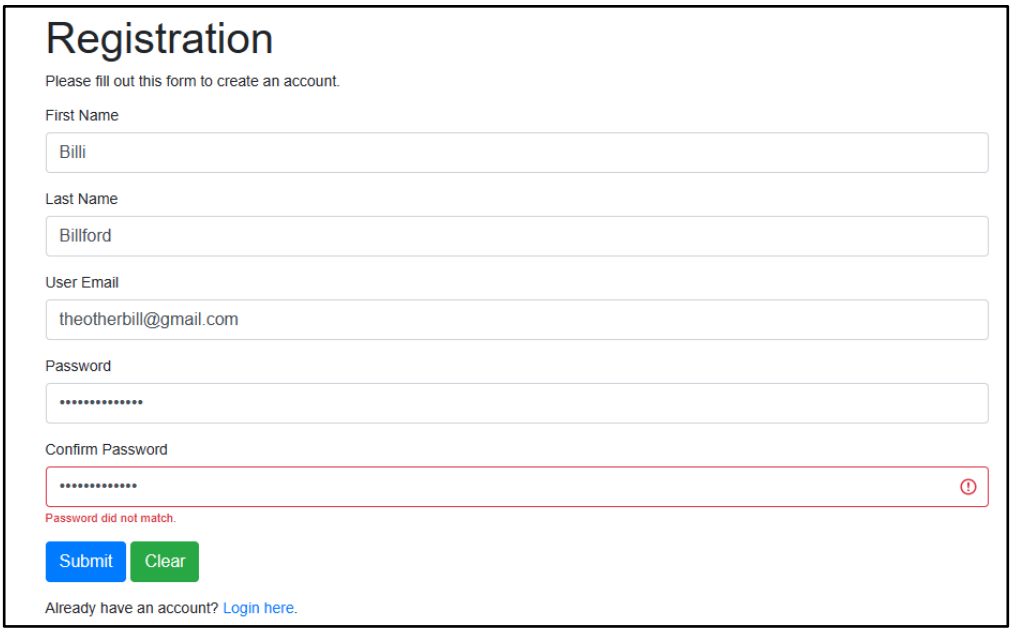
2. Input Field Validation

For input fields, we have measures installed to make sure that the information that is asked for is checked and entered correctly by the user. For users, the entries are checked to make sure the patterns that are scanned for are either present or not:

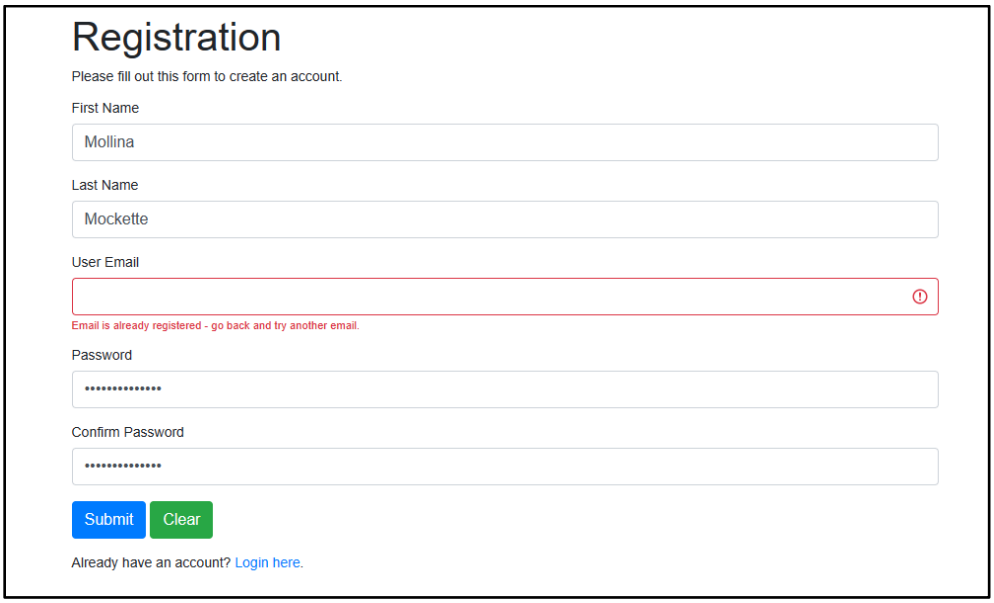
Sign Up - Incorrect Email/Password:



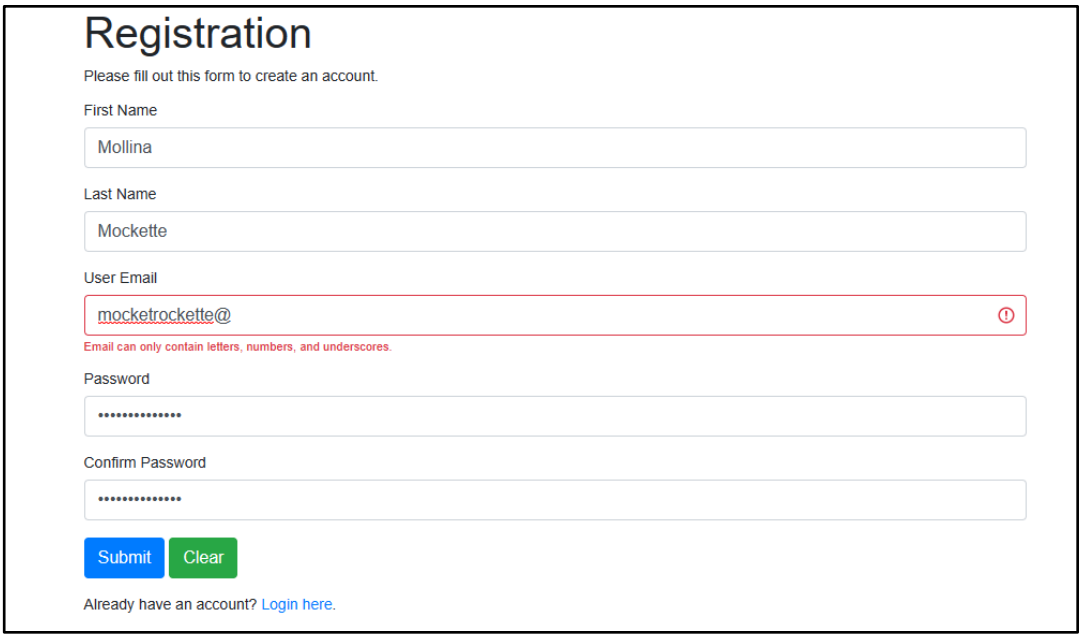
In this example, the email has a space in between the name, so it is rejected, and the user is made to correct it. For the password, it didn't have all of the four requirements to make a strong password. The password must contain at least one alphabetic letter, one number, one special symbol, and must be over 6 characters long (Itwillbecool7!). The password is also hidden with asterisks when inputting the password in the field, supporting better security for user' sensitive data. If you enter these values correctly, but let's say, you messed up the password confirmation, it will tell you that the information doesn't match:



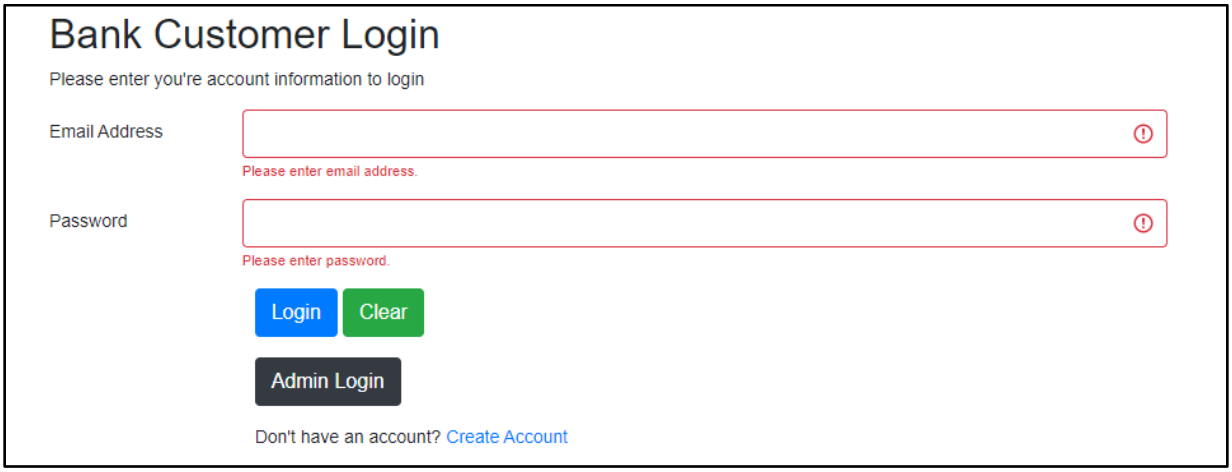
If one tries to make an account with an already existing email, the site will also tell them this, and request that they use an unused user email to continue account creation:



If form entry is incorrect for email, the pattern checker will catch it and an error telling the user what characters are permitted will be returned:



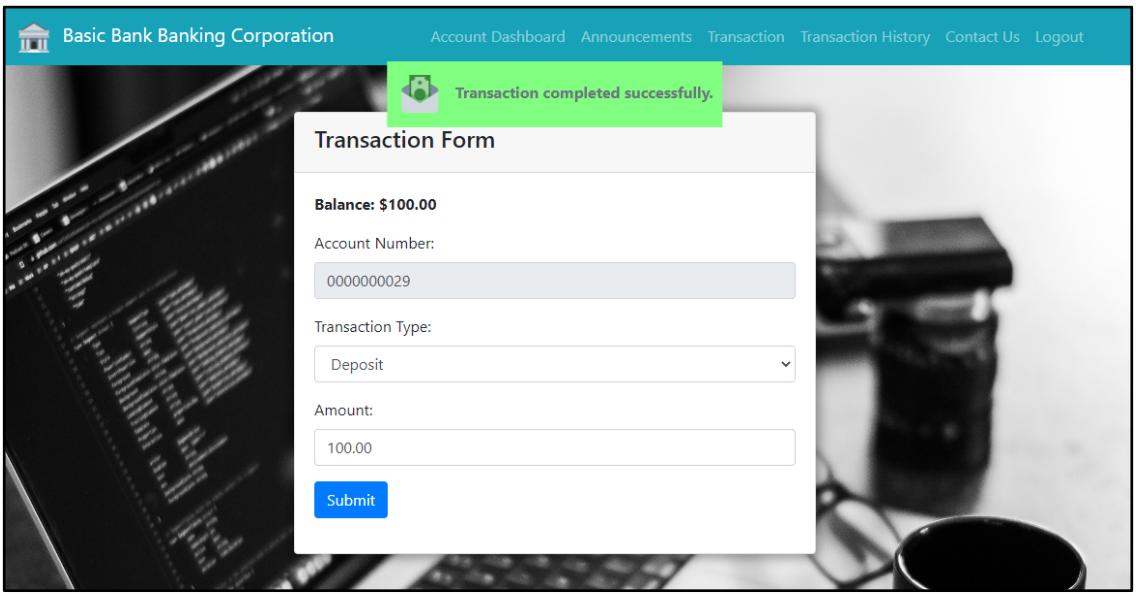
If we go to the login page, if we are missing either the email or the password, the user will be prompted to enter their information. This is a simple field check with PHP code that checks if the fields are empty or not on submission:



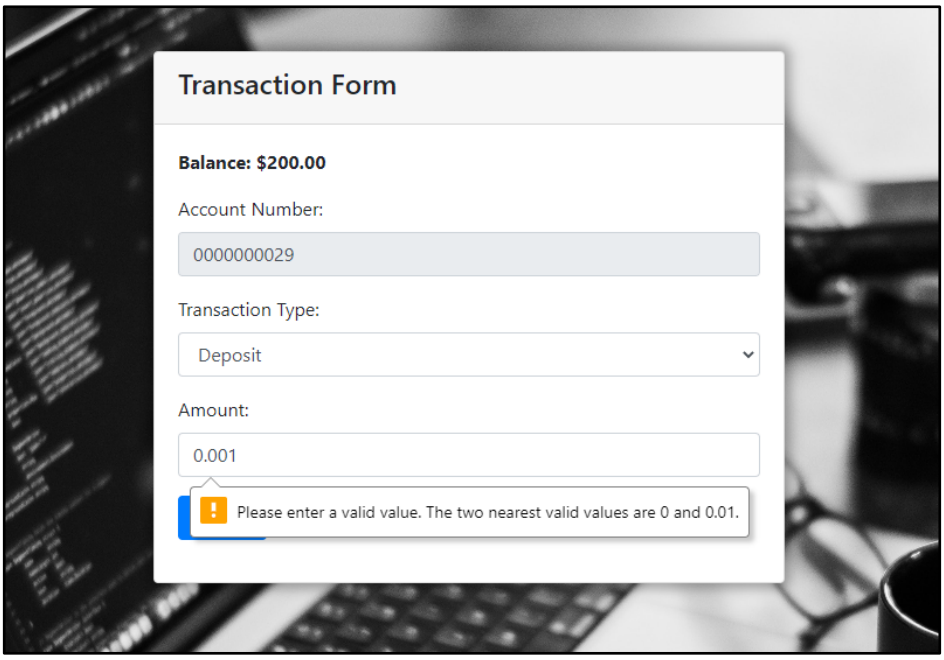
For the admin side, this same type of form validation exists, as well. If an incorrect username or password is entered, they will be told of this, and be told to reenter the correct information. For admin side form submission after login, things may be slightly different, but that will be shown later.

Once we are logged in on the user side, there is one area in which user entries will be accepted: the transactions page.

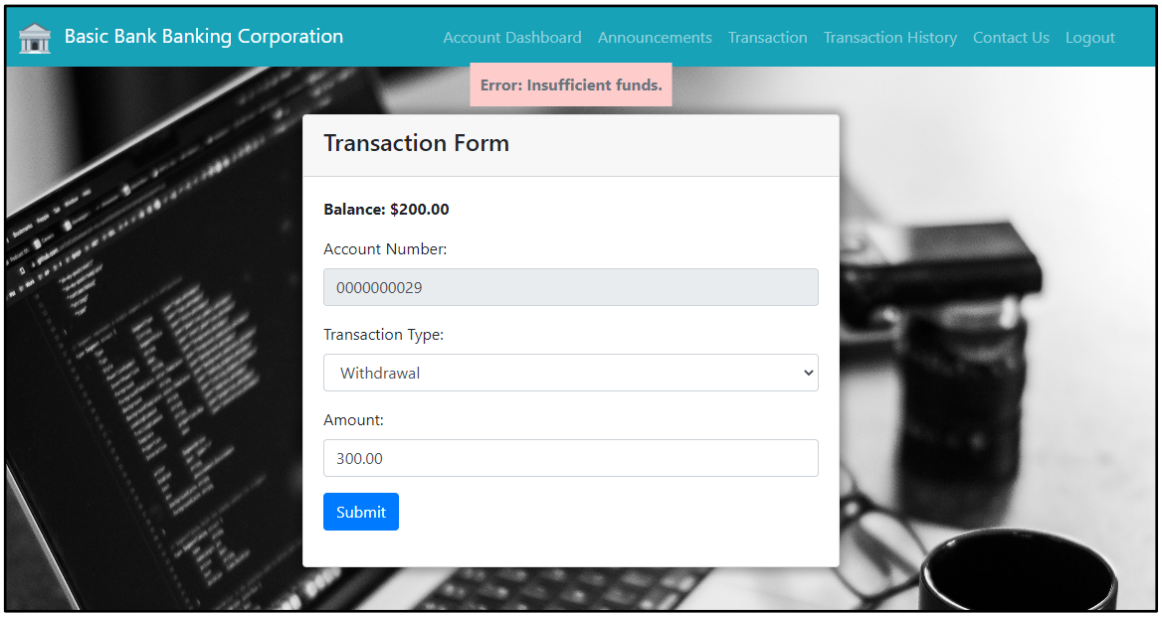
Normal input:



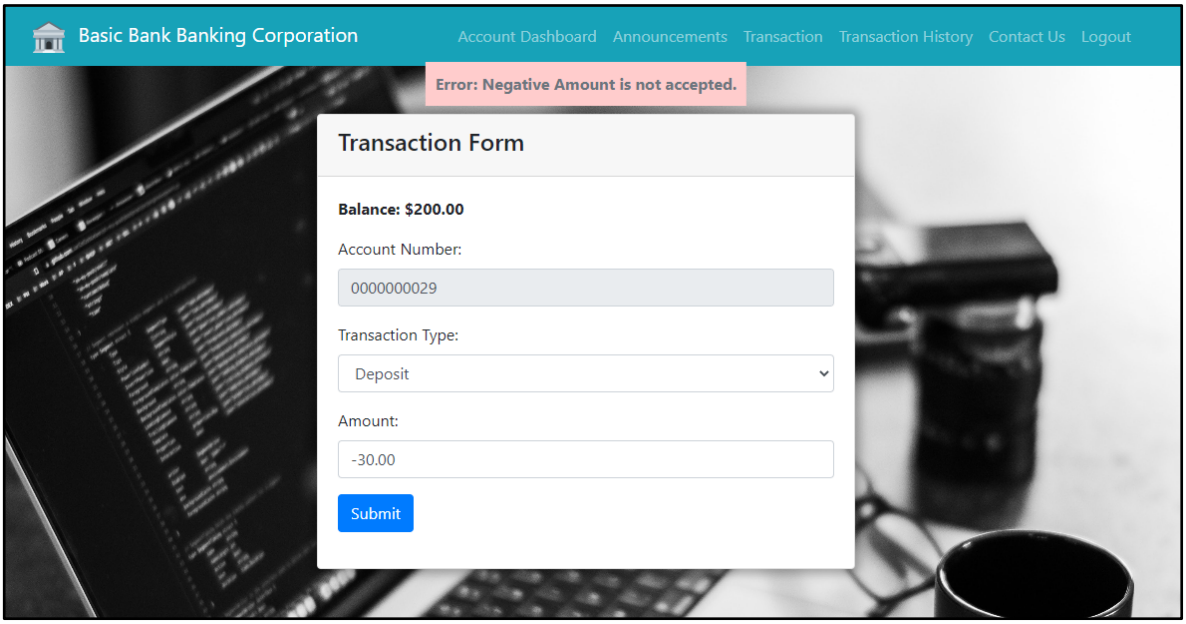
Amount lower than one cent:



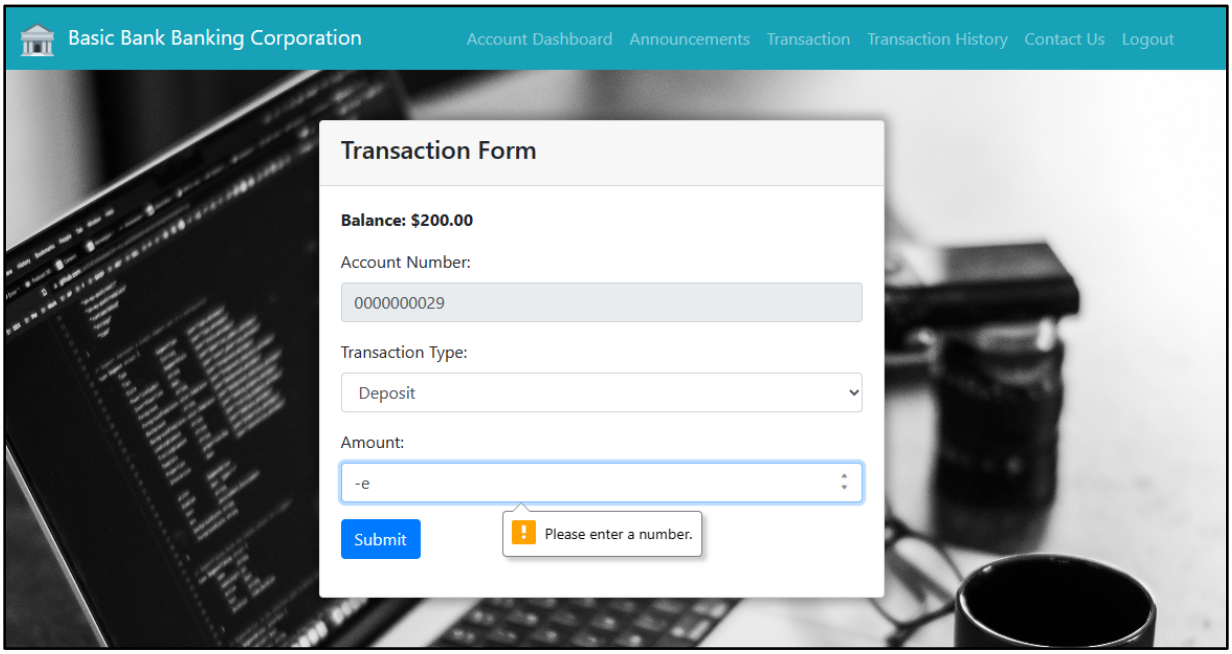
Account balance insufficient:



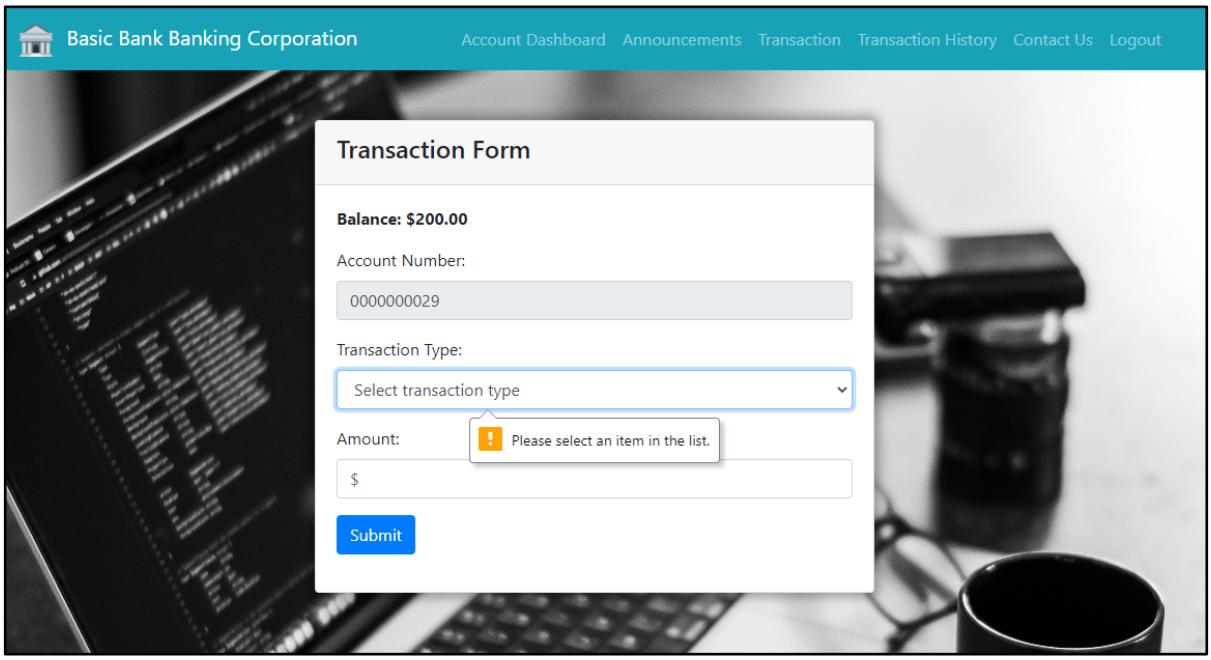
Negative value:

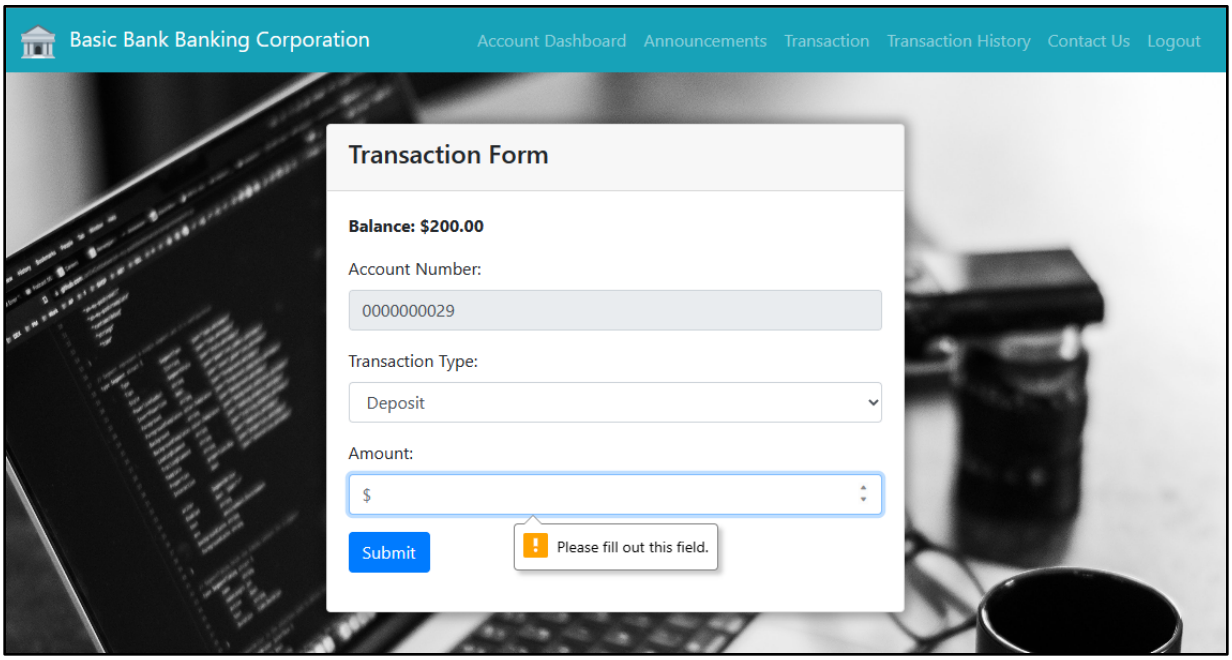


Non-Numerical Value:



Nothing:





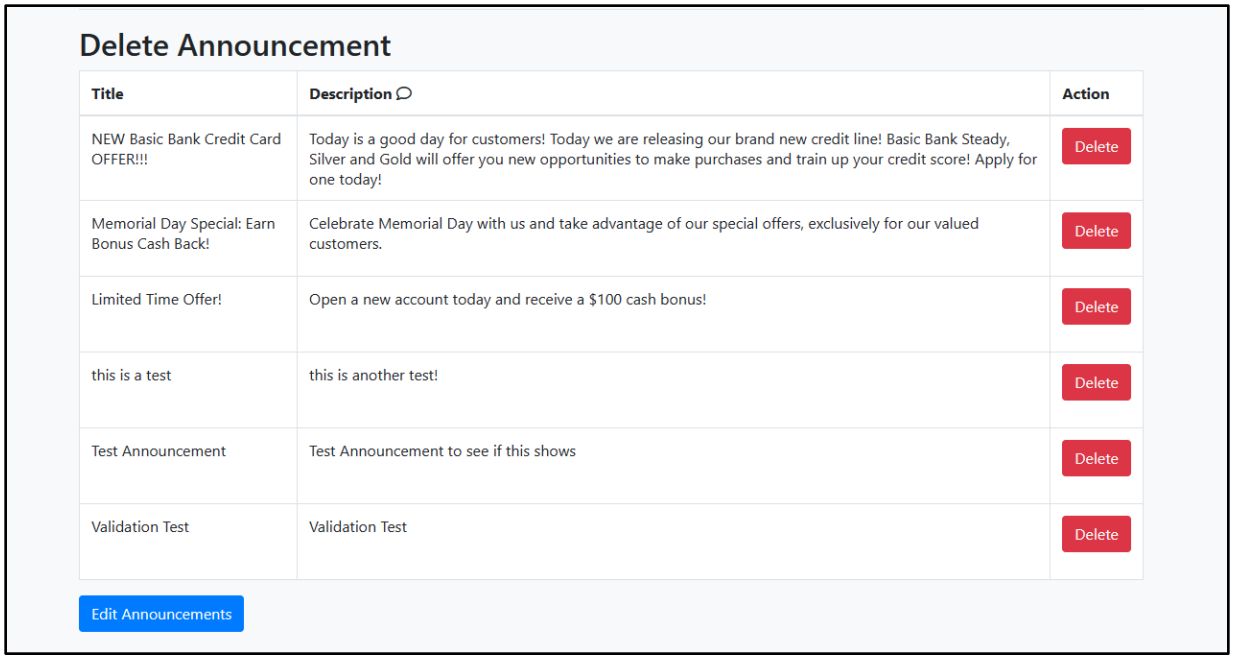
The implementation of all of these all revolved around just checking user input before attempting to process anything in the database itself or having a “required” flag on the HTML section of the code so that it can’t be skipped. We check if the input is acceptable/correct. If it is, it is run, if not, it is rejected by the site before being sent out to change anything in the database.

For admin, there are several different points on site where admin submission is accepted in fields. These fields are for announcement creation, announcement editing, adding a bank customer and editing a customer’s information.

Announcements: To implement this validation check, it has the same “required” flag on the html side as it exists on the user side that won’t let the admin skip entry. Besides this, any characters are permitted to be used in either the title or description.

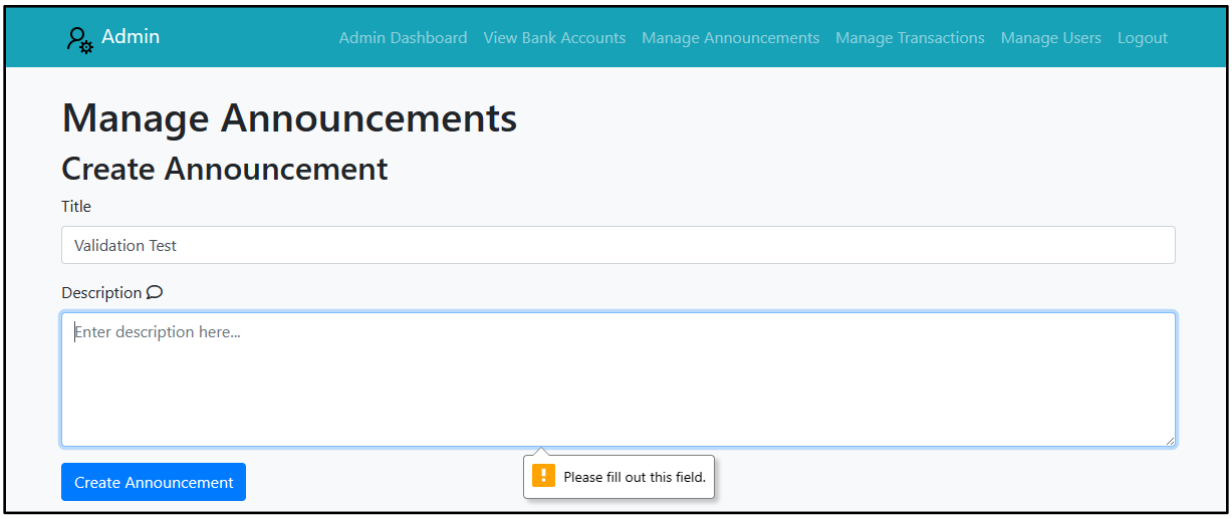
Normal Entry:

New Announcement shows up at bottom:

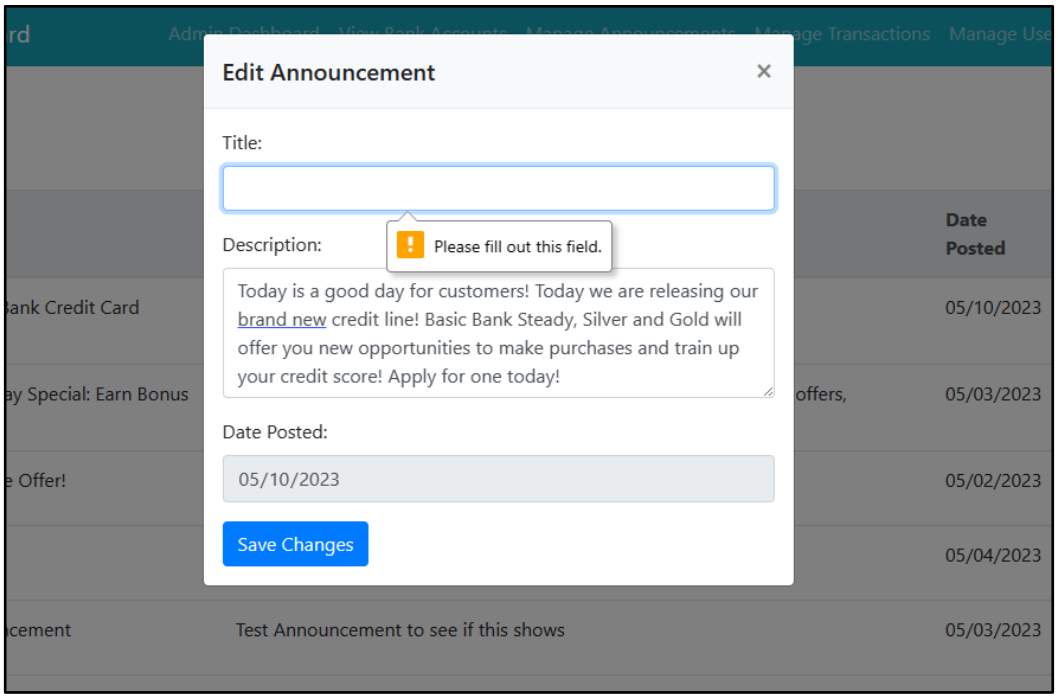


No Title:

No Description:

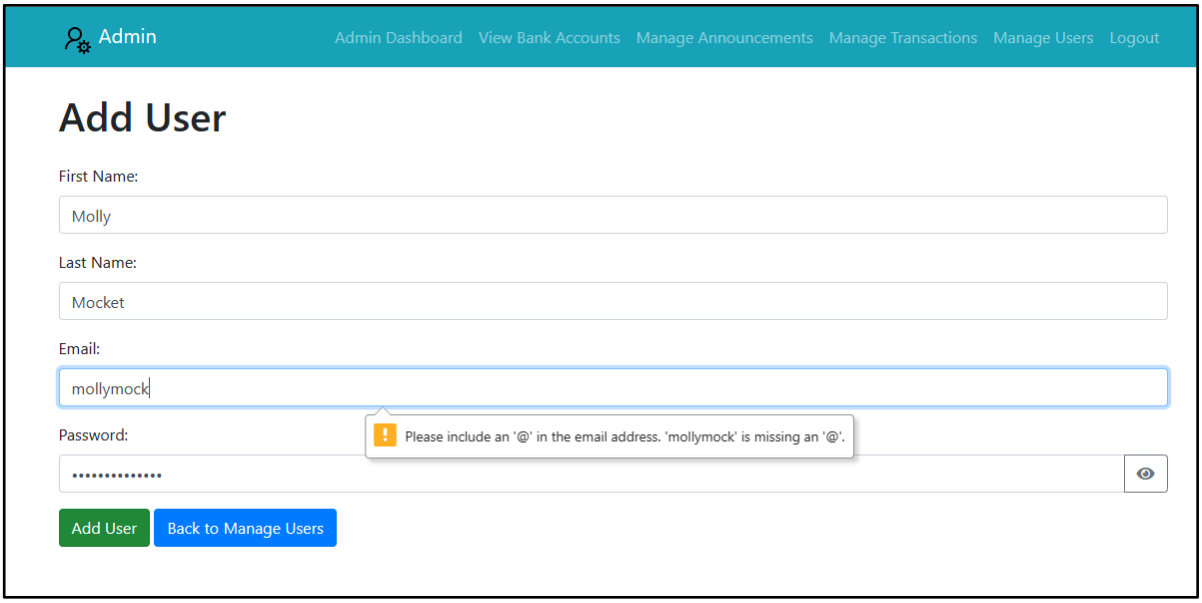


Announcement editing is the same - The title itself is a required field that must be present. If not, the announcement will not update:



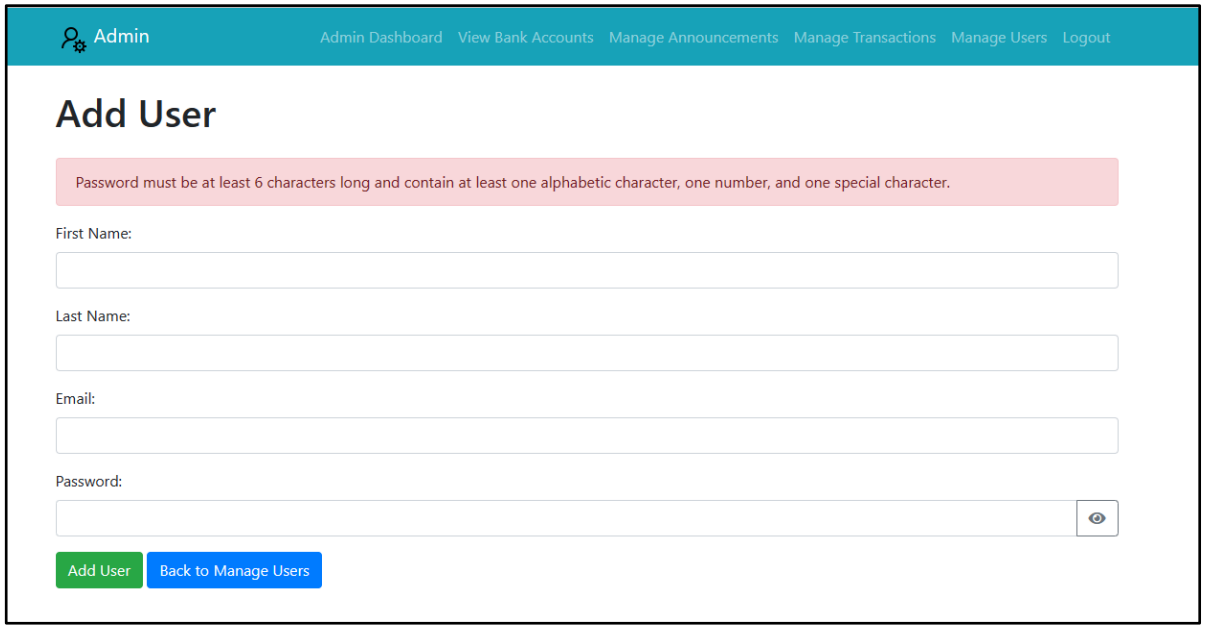
For account creation on admin side, the step for creation is the same - they have entry requirements that must be followed, or input won't be accepted:

Incorrect Email Entry (no @ included):

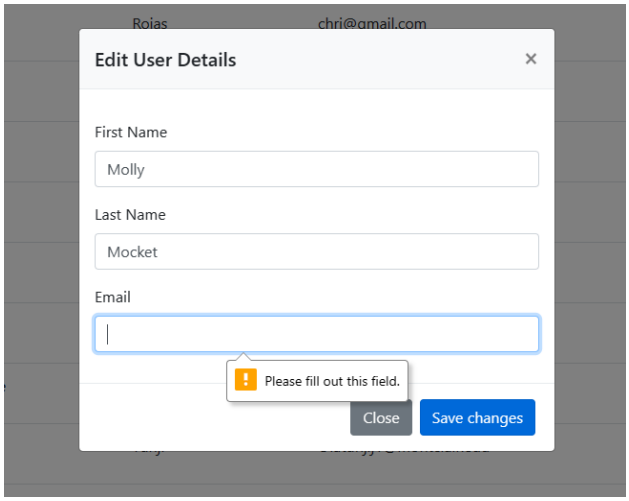


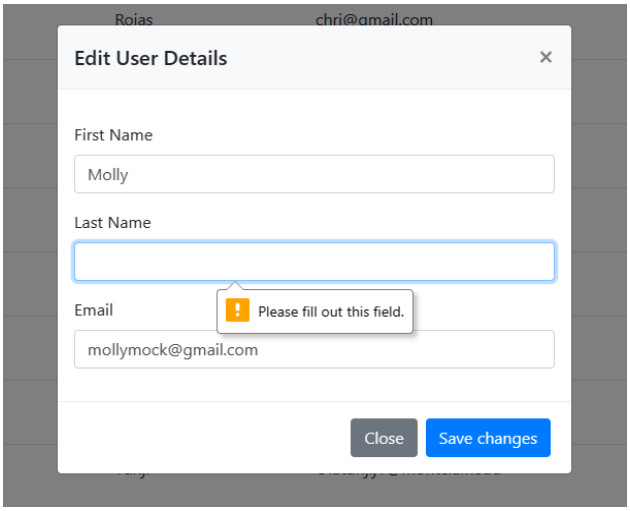
Nothing after the @:

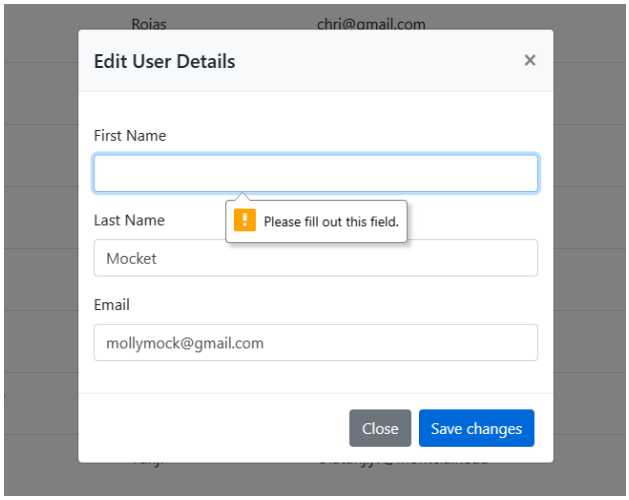
Incorrect Password/Short Password:



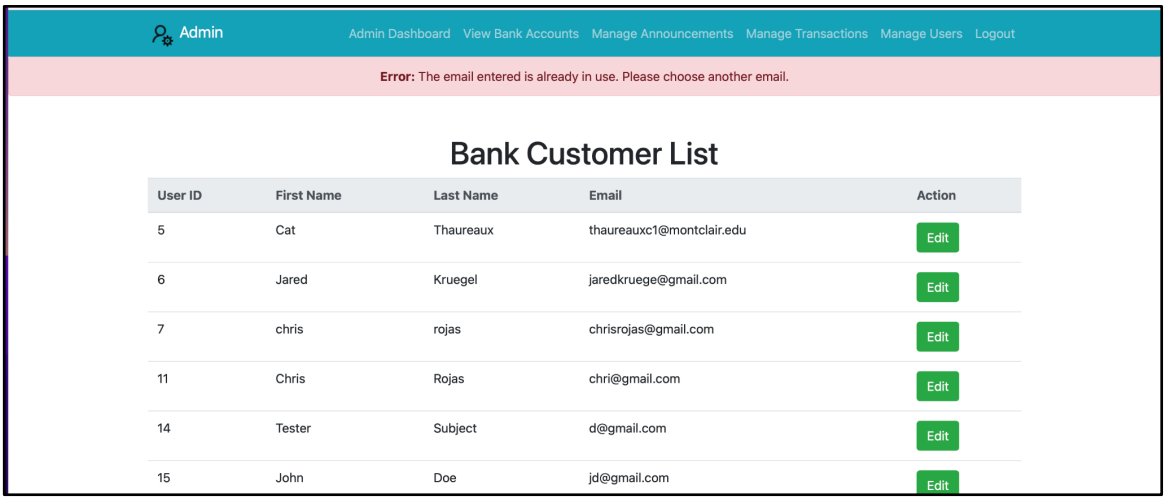
If we want to edit user credentials, we can only change the name and email. If one of these fields is empty, it will not accept any updates:







Besides this, any characters are permitted for form editing as long as they follow any required formatting.



If an admin attempts to edit a user's email to an email address that is already registered by another user, an error message will populate, and the query will not take place.

3. Cookie Sessions

On the website in order to make each account not share the same main page, every user once logged in has their own session generated off of their ID and username. With these sessions, the only information that they can access, and view is strictly unique to those identifiers.

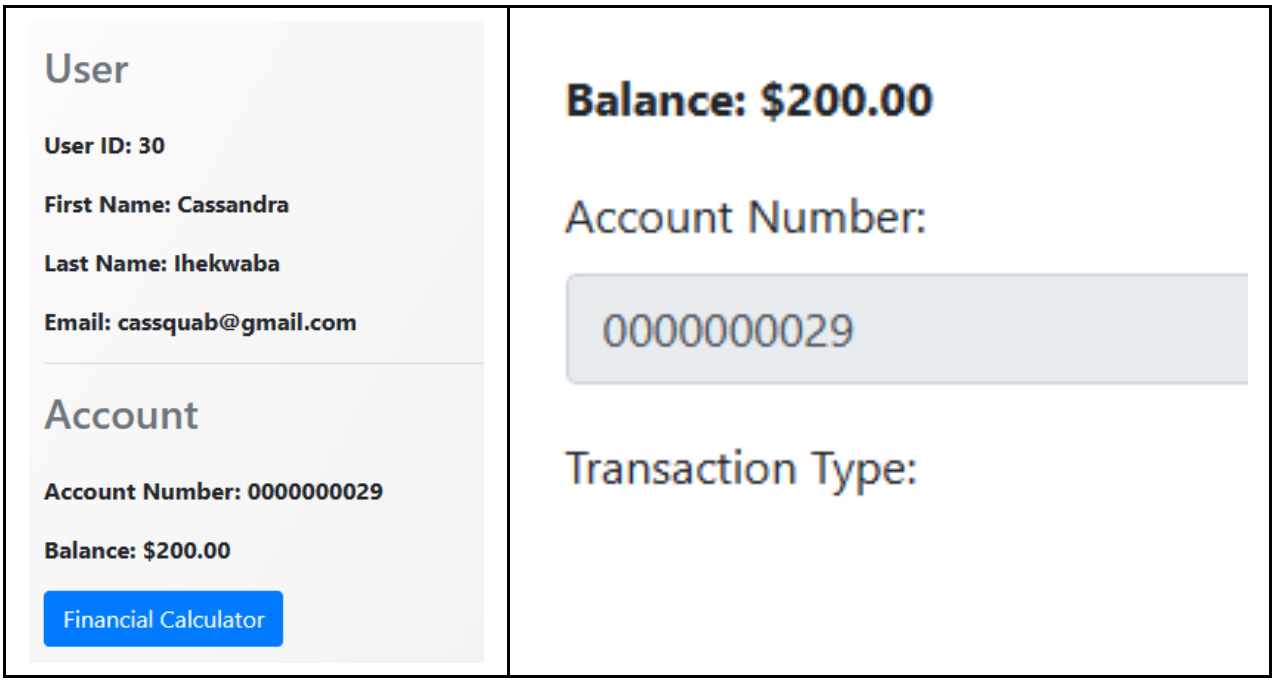
To do this, session variables had to be made and certain user identifiers had to be bound to them after successful password matching if account details are found:

```php
if($result->fetch()){
    if(password_verify($Password, $hashed_password)){
        // Password is correct, so start a new session
        session_start();

        // Store data in session variables
        $_SESSION["loggedin"] = true;
        $_SESSION["userID"] = $userID;
        $_SESSION["Email"] = $Email;

        // Redirect user to welcome page
        header("location: user_homepage.php");
```

The session that one is logged into is not immediately visible in the URL like it would be for an announcement, but there are a couple indicators that do show and link the session to you based on the credentials that show up on your dashboard and the ID linked to you:



As can be seen here, the login session variable is set to “true”, then the user id and email of the logged in user are all held in these variables after creation (or start) of the session, as indicated by session_start(). This ensures that after the session is made, the session is then set to operate under these cookies. I can change and exit this tab as long as I remain logged in. However, as soon as the session is logged out, the variables then dump their data and start fresh again. This is also true for admin side - these same session variables and code rules apply since a session has to also be made for each individual admin, even if they are seeing the same information. They can stay in the

same session as long as they don't log out of their account. If they do log out, the same rules still apply, and they have to log back into re-access their session.

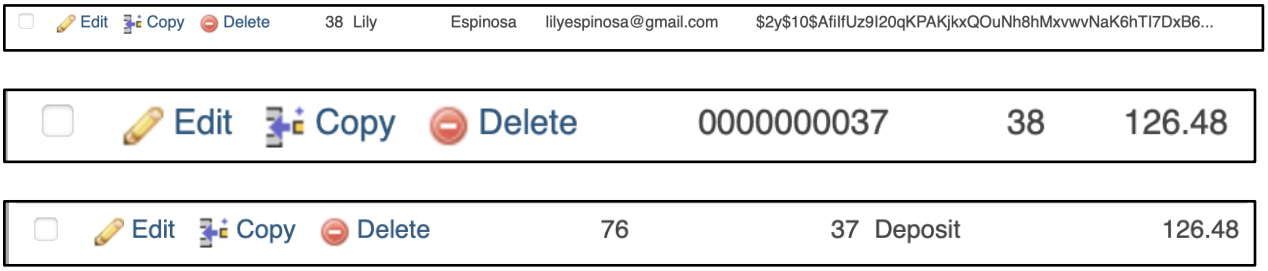
4. Form Submission Security:

A token is used to ensure that a transaction can only be submitted once, preventing user errors such as double clicking the transaction button or even protecting against malicious attackers. This would be a large security concern as well as a logical error concern if a bank user is able to click the transaction button twice before the form submits, allowing the transaction to go through multiple times in just a second. Attackers can also use this to their advantage by using automated, high-clicking speed softwares to overwhelm the application with multiple requests, ultimately committing a denial-of-service attack. In this attack, an attacker can attempt to overwhelm the web application, in this case, clicking the submit button in the transaction page at a high-speed rate, sending the server 100s to 1,000s of requests, eventually blocking other users from accessing the website.

To implement a prevention measure in the transactions page of the site, a random token is generated and stored in the $_SESSION array, where other session variables also reside including the userID of the currently logged in user. When the transaction form is submitted (when the user clicks the accept button), the token is sent to the server and used to validate any other form submission requests that are accidentally or maliciously sent to the server. By incorporating the validation token, we can prevent both users that either have malicious intent or users that accidentally submitted the form multiple times.

5. Verify the Correctness of the Output:

One of the main computations that occurs in the site is the transaction page. This feature is not only modifying the transactions table in the database, but it is also updating the balance attribute in the accounts table. This information needs to be verified and tested for the correctness of the output. When a new user has been created and is logged into their account, they will have $0. When they add money into their account, their balance will be updated with the correct amount. The images below show the data that has populated in the table for that specific user with the correct amount input into their balance.



IV. Roles of the Team Members (Contributions)

Catherine Thaureaux:
-SQL code
-database_connection.php
-add_user.php
-admin_homepage.php
-admin_login.php
-admin_navbar.php
-calculator.php
-edit_user_v2.php
-manage_announcements.php
-manage_bankaccount.php
-manage_transactions.php
-manage_users.php
-registration.php **
-results.php **
-transactions.php**
-update_announcement.php
-user_homepage.php
-user_login.php**
-front end design for user and admin side

Cassie Ihekwaba:
-announcements_pg.php
-announcement_details.php
-historyview.php
-results.php
-user_login.php**
-registration.php**
-navbar.php (Navigation bar for User Side interaction)
-Slight Visual Modifications to historyview.php before final adjustments were made.

Jared Kruegel:
-contact.php
-transactions.php** (added token to prevent double form submissions)
-user_homepage.php
-HTML/CSS

Chris Rojas:
-index.html
-transaction.php**
-HTML/CSS

Joe Chalet:
-logout.php
-LoginCheck.php
-user_login.php**
-debugging

**** References contribution/collaboration to a file**

V. Project Experience

Our group was asked to design a user interface for a banking application, while mainly focusing on the functional and non-functional requirements. Were given a semester (16 weeks) to complete the banking app, alongside check-ins and milestones (such as the midterm presentation). Throughout our experience I feel as if our group worked in a timely manner and completed the tasks before deadlines. Although some group members didn't know how to start the project due to the limitation of knowledge (like PHP), we still managed to create prototypes and figure out how we would complete implementation of the banking application.

There were some challenges we faced, which included bugs in our code (specifically regarding security). However, since our group had such a diversity in knowledge, we were able to help one another with these problems. A great example is Jared helping Chris with a token with the transaction, ensuring a customer cannot create multiple transactions by clicking the button twice. At the end of the project, we were all satisfied with the work our team put into the project. We did a great job with testing the site, using many of the methods learned in class (like blackbox testing and smoking testing). Towards the end of the project, we held a meeting to run through the site together as a user, finding any errors or bugs present within the site. With 5 different people's knowledge and perception, we were able to create a list of things that needed to be changed, whether it was related to UI/UX purposes, usability purposes, or security purposes.

VI. Works Cited

- https://stackoverflow.com/questions/4614052/how-to-prevent-multiple-form-submission-on-multiple-clicks-in-php
    - References establishing a token on the transaction page to prevent repeated forms clicks when we would only want one click, increasing the security of our transaction system.
- https://www.php.net/manual/en/function.password-hash.php
- https://getbootstrap.com/docs/4.0/components/navbar/
- https://www.tutorialrepublic.com/php-tutorial/php-mysql-login-system.php