

CampusCoin

A Student Budgeting Application

Professor Sharon Perry

CS 4850 - Section 01 - Fall 2023

December 3, 2023

<https://github.com/baileyloewe/CampusCoin>

[Video presentation](#)



Andrew Silva

Avanishkumar Patel

Bailey Loewe



Drashti Patel

Will Roach

Contents

Introduction and Project Overview	3
Requirements	3
Version Control	5
Development	7
Database Development	7
App Features and Pages	9
Contributions.....	13
Conclusion	14

Introduction and Project Overview

Our team decided to develop a student budgeting application for our senior project because we wanted to create a simple, yet effective piece of software that would assist students in making informed financial decisions based on their expenses and income. We felt that there was a need for a budgeting application that was specifically designed for students, as many of the existing budgeting applications on the market were either too complex or did not consider the unique financial circumstances of students.

The budgeting application's concept was not only to provide useful functionality, but also to be simple enough that there was a reasonable likelihood of meeting the functional requirements of our software and reaching a degree of completion. We also wanted to create an application that was easy to use and understand, so that students would be able to get the most out of it. However, because the team's prior experience with app development was in smaller personal projects with narrow scopes, the project would still provide a suitable challenge. We knew that we would be learning a lot as we went along, but we were confident that we could complete the project successfully.

Our team consisted of five members, which necessitated two deliverables: a mobile and desktop student budgeting application that was dubbed "Campus Coin." We divided the work into two primary teams, a team responding for developing the code base of the applications, and another team responsible for project management, documentation, and so forth. We worked closely together throughout the process, having weekly meetings where feedback and collaboration between the two teams took place.

Although our team was unable to fully implement all the features that we had initially desired, we were able to successfully implement the functional requirements, thereby developing a viable, fully functional application that fulfills the goal of our project: to provide students with an application that could be used to make educated decisions by utilizing their tracked financial details.

Requirements

For our requirements stage, we defined clear requirements in our requirements and design document to ensure a smooth transition from the research stage into the development stage. In the document we listed out various features, design constraints, and requirements for the app, detailing things such as what the launch page should include, how the dashboard should be laid out, and how income tracking should be managed, and more.

To accentuate this, we also developed a high-level project progress chart to define key sections for our project's deliverables, helping to track the completion of various deliverables for the project. While not extremely detailed, it helped provide a view of where we were at with our

visualizing your expenses by category. This flow gave us a baseline to work with to create a simple process flow with a palatable UI, alongside providing tasks and information that was easy to absorb.

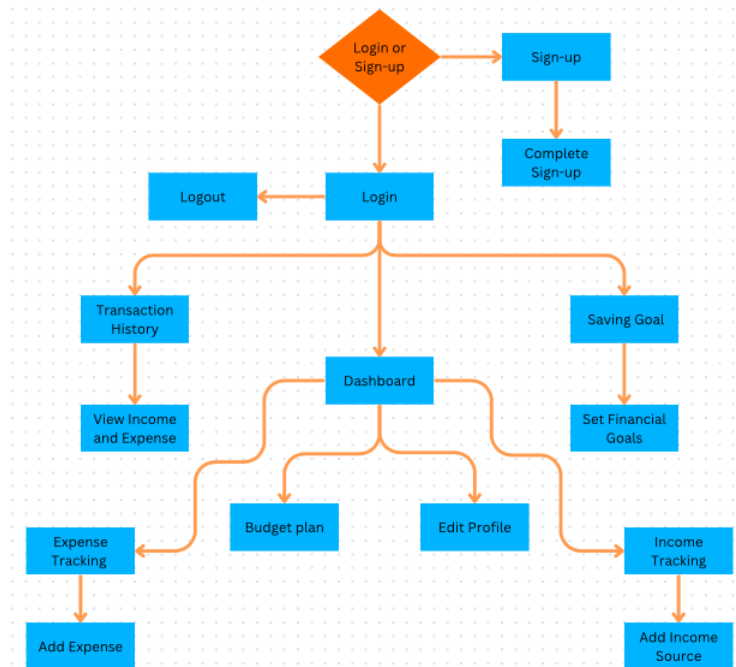


Figure 3: The initial CampusCoin process flow chart

Version Control

For version control we chose to use GitHub alongside GitHub desktop. We made this choice because it is simple, popular, and easy to use. Despite this, we had a few issues with it due to a lack of experience, such as issues with pull requests and managing branches, that needed to be troubleshooted before we could begin work on our project. Fortunately, Will had previous experience with GitHub through an internship and was able to coach the rest of the team through using GitHub effectively. After the initial short learning curve, GitHub proved to be a reliable and effective tool for version control and collaboration. We were also able to easily publish our senior project website through GitHub's website hosting feature. Furthermore, the feature of organizational hosting made it easier for all team members to join and collaborate throughout the project without much hassle through their personal accounts.


	baileyloewe Update index.html	✓ 90c322f 34 minutes ago	🕒 169 commits
📁	CampusCoin	Added proper back button behavior for expenses page	yesterday
📁	image	Add files via upload	last month
📄	.gitignore	Initial commit	3 months ago
📄	CampusCoin.sln	initial template commit	3 months ago
📄	GraphTestPage.xaml	changes	3 days ago
📄	README.md	Initial commit	3 months ago
📄	index.html	Update index.html	34 minutes ago

Figure 4: The CampusCoin main GitHub page

Commits on Nov 14, 2023			
Merge pull request #33 from willroach24/QoLChanges11/14/23	Verified	2f3217b	<>
baileyloewe committed 3 weeks ago			
submitted UserData to DB		33b6fd2	<>
aSilvaCodes committed 3 weeks ago			
updated db models		78c1863	<>
willroach24 committed 3 weeks ago			
email duplication check, updated db models		ddc0f90	<>
willroach24 committed 3 weeks ago			
Merge branch 'QoLChanges11/14/23' of https://github.com/willroach24/S...		11c3d63	<>
baileyloewe committed 3 weeks ago			
Various small fixes and fleshing out password reset		1c5e647	<>
baileyloewe committed 3 weeks ago			
Merge branch 'main' into QoLChanges11/14/23	Verified	8cb99cd	<>
baileyloewe committed 3 weeks ago			
Large changes		3ab22df	<>
baileyloewe committed 3 weeks ago			
Merge pull request #30 from willroach24/expensesPageWeek6	Verified	1691d8a	<>
aSilvaCodes committed 3 weeks ago			
began testing an expense page with a link to an expense report page		460b98d	<>
aSilvaCodes committed 3 weeks ago			

Figure 5: The CampusCoin commit history

Development

Database Development

Before writing any code, it was important to have the database solution established because the rest of the codebase would be reliant on the interconnection between our code and the database. After doing some research and analyzing our use case- it became clear that a SQL Server instance hosted on Azure would be the best option. The advantages were multifold, but the decision ultimately came down to two aspects. The first being that SQL Server has widely available documentation and is supported by most database frameworks such as Entity Framework. Secondly, Azure gives KSU (Kennesaw State University) students a free 100-dollar credit for hosting databases, this also came with other benefits such as automatically being integrated with KSU services, which allowed us to easily share and add our team members to the instance as database admins. Other solutions had differing advantages, such as increased simplicity, leading to a potential better fit with a smaller scope project such as our own, but we think we made the best choice overall.

We opted to use the aforementioned Entity Framework to connect our code to our database based on principles of clean code practices, security, and usability. Entity Framework allows for a straightforward transformation of database entities into code-based classes, and it also provides increased database query security by encrypting data and parameterizing queries. The use of Entity Framework in our code provides a strong and secure connection to the database.

```

4 references
public class RegistrationService
{
    private IDbContextFactory<CampusCoinContext> _context;
    0 references
    public string DbPath { get; }
    0 references
    public ObservableCollection<User> UsersCollection { get; } = new();
    List<User> usersList = new();

    0 references
    public RegistrationService(IDbContextFactory<CampusCoinContext> context)
    {
        _context = context;
    }

    /// <summary> Gets a list of all users from the database </summary>
    /// <param></param>
    /// <returns> A list of users from the database </returns>
    1 reference
    public async Task<List<User>> GetUsers()
    {
        var dbContext = await _context.CreateDbContextAsync();
        usersList = await dbContext.Users.ToListAsync();
        return usersList;
    }

    /// <summary> Registers a user in the database, then saves the database </summary>
    /// <param name="user">The user to be saved</param>
    /// <returns></returns>
    1 reference
    public async Task RegisterUser(User user)
    {
        var dbContext = await _context.CreateDbContextAsync(); // Create database context
        dbContext.Users.Add(user); // Add the user to the database
        dbContext.SaveChanges(); // Save the changes to the database
    }
}

```

Figure 6: Entity Framework Code Example

Unfortunately, our database usage was higher than expected and our 100-dollar student credit ran out within a couple of months, so we had to temporarily switch to hosting the SQL Server instance locally on Bailey's computer. This switch was not smooth, and required time investment, software installations, port forwarding, firewall adjustments, and many other difficulties. However, after switching, changing from the cloud host to local host interfered with our Android application. Since Android forces apps to use TLS (Transport Layer Security) 1.2 for connections within the app. Fortunately, we were set up with TLS 1.2. Unfortunately, we did not have a valid certificate from a certificate authority, like Microsoft, which Android also requires. There are config options to circumvent this, but we were not able to do so due to .NET Maui's integration with Android not allowing certain configuration options to be appropriately altered. The cloud instance was automatically granted a valid certificate when we created it, but locally hosted instances are not, and they cannot be carried over when you switch. We decided the best fix was to swap back to Azure using another student credit, which promptly fixed the issue and allowed us to wrap up the project with both deliverables intact.

Application Development

The first step in laying out the project development environment was to prepare it for development, and to do so, we needed to outline a standard design pattern. We opted to go with MVVM (Model View View Model), as it is supported by default in MVVM, and is an industry used design pattern. The benefit of having this pattern was clear: it made new code easy to integrate (and remove) and eliminated the need for major redesigns or code refactoring in later phases of development as requirements grew, shrunk, and changed.

The MVVM architecture has three basic principles. The Models, or data behind, act as the data for your application. The Views, or UI, act as what the user sees and interacts with. The View Models tie both together, allowing for translation between a View and a Model, aka View Model. This functions by taking, say, a user entered expense in the form of a string into a text box, and converts it into a parameterized SQL query that is sent to the database to be stored. And vice versa, the Model will host data that needs to be sent to the view, so the View Model will grab the data, convert it to an appropriate format, and send it to the UI. This decoupling of the data and UI enables a vast amount of flexibility with our code and design, because changing a table in the database won't interfere with the View on the other side, or changing the UI will not break the Model interacting with the database, for example. Utilizing the MVVM pattern made the relationship between user interactions and the underlying data tables easy to add, remove, and alter, adding much needed modularity to our code.

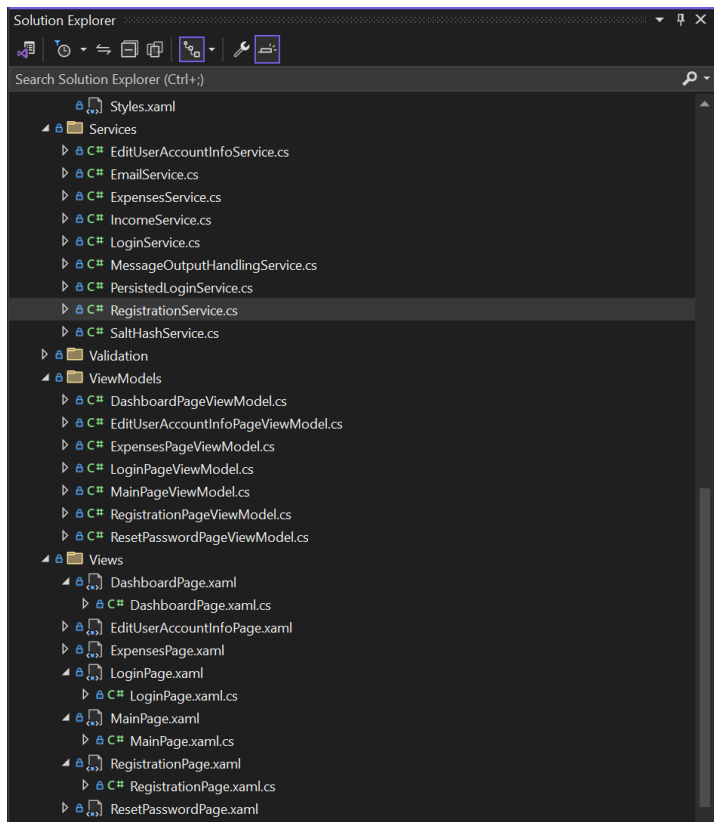


Figure 7: MVVM Design Pattern

Another key aspect of our design was the use of dependency injection. This pattern decouples classes from each other and is a way of designing your code so that instances of each needed class are provided as requested to any given class. This contrasts with having to instantiate every class within every other class yourself, which can result in complex, hard-to-untangle code that is not very modular. Dependency injection services are used for all aspects of our code that are not directly tied to the UI. For example, services are used to handle logging in, connections and queries to the database, encryption of passwords, and so forth. Overall, dependency injection allowed for the seamless integration of services, which are essential for interacting with the database and other features.

App Features and Pages

In the login page, developed primarily by Andrew, users can enter their credentials in the email and password fields on the login page. Once the data is entered, users can click the login button, which will launch a query to the database to verify the information they provided. Users are promptly logged in if their login information matches a user or rejected if not. The page also has a "Forgot Password" button that, if users forget their password, takes them to a page where they can reset it, developed by Bailey. This has them type their email, which then sends them a verification code via email that they can enter the app to reset their email. Users can also set the app to automatically login using the "Remember Me" option while logging in. There is also a "Sign Up" button at the bottom that makes it simpler for people who haven't registered to get to the registration page.

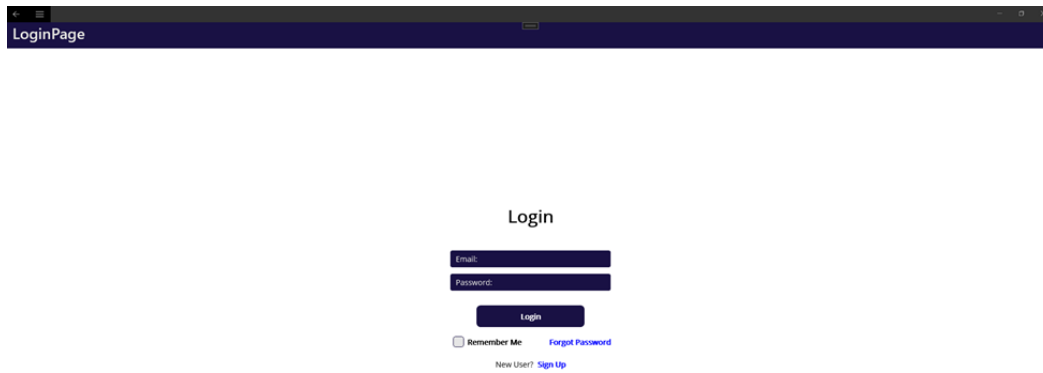


Figure 8: The CampusCoin login page

The registration page, developed primarily by Bailey, works similarly to the login page. However, it has a few new features. When registering, users must enter their email, password, phone number, first name, and last name. These all have minimum requirements of some form, such as emails being in an email format and valid, passwords being 10 or more characters, phone number being only 10 digits, etc. Any attempts to register without meeting these requirements are kicked back. Once fixed, you can register, which requires you to verify your email before doing so. A code is sent to your email, which you can enter into the app to complete your registration. You are then directed back to the main page to log in.

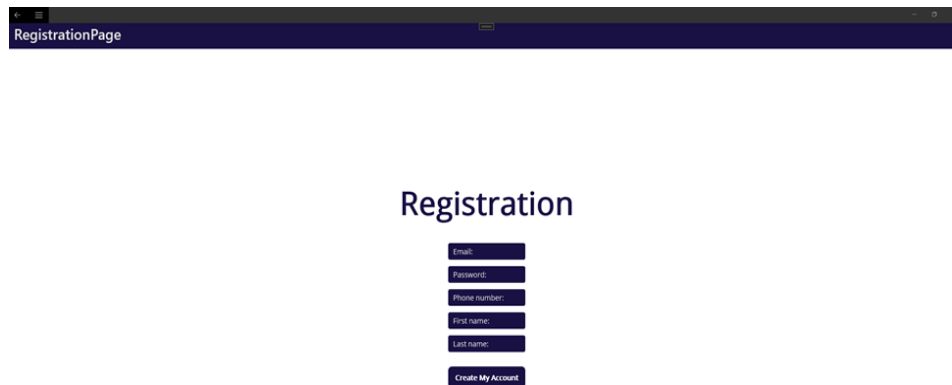


Figure 9: The CampusCoin Registration Page

After logging in, you are taken to a dashboard page, which was developed primarily by Will. Clicking the "Add Expense" button in the dashboard allows access to the expenses page. The expense page includes multiple categories to accommodate various spendings. Users can enter

the corresponding expense amount in the designated field after choosing from one of the six available categories. Users have the option to include a brief description of the submitted expense in the description entry field. The application submits the expense data to the database when the "Submit" button is clicked after completion. The user can then choose to submit another expense or go back to the dashboard. Beneath this, you will find a visualization of your expenses by category, and below that you will find charts of your expenses and of your income by month, with a comparison between the two. There is also an edit account information page when the user clicks "Settings" in the side widget, which allows the user to edit all of their account information.

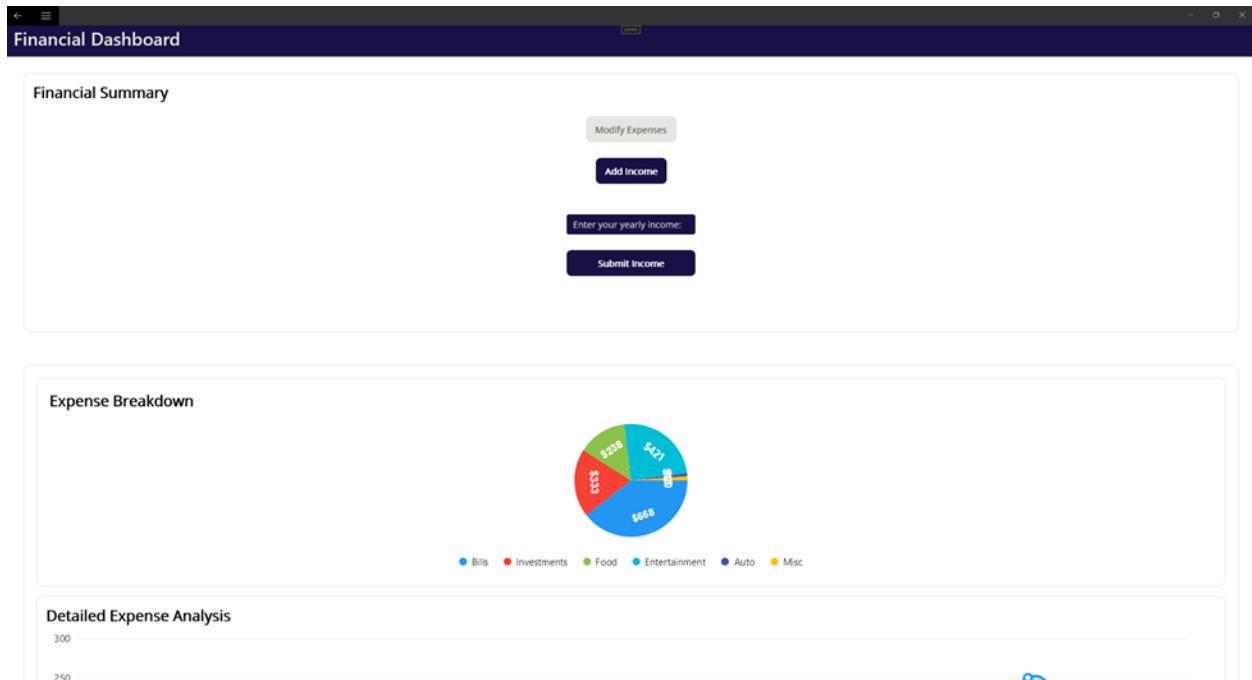


Figure 10: The CampusCoin Dashboard page 1

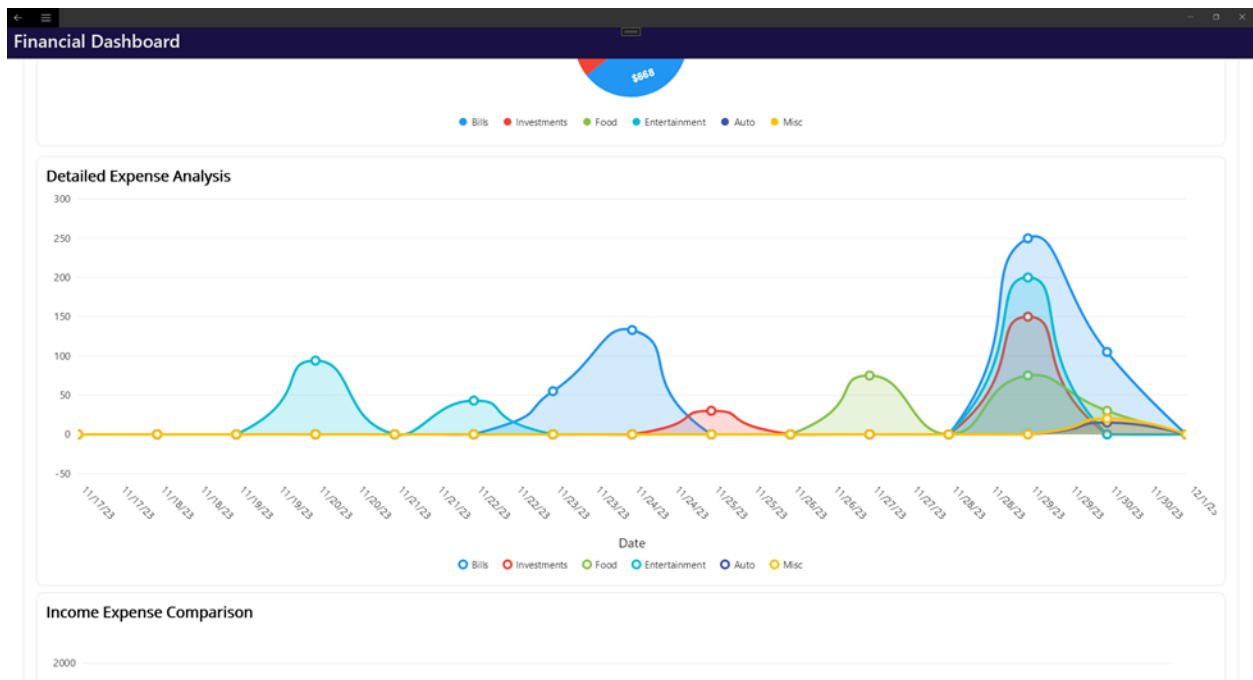


Figure 11: The CampusCoin Dashboard page 2

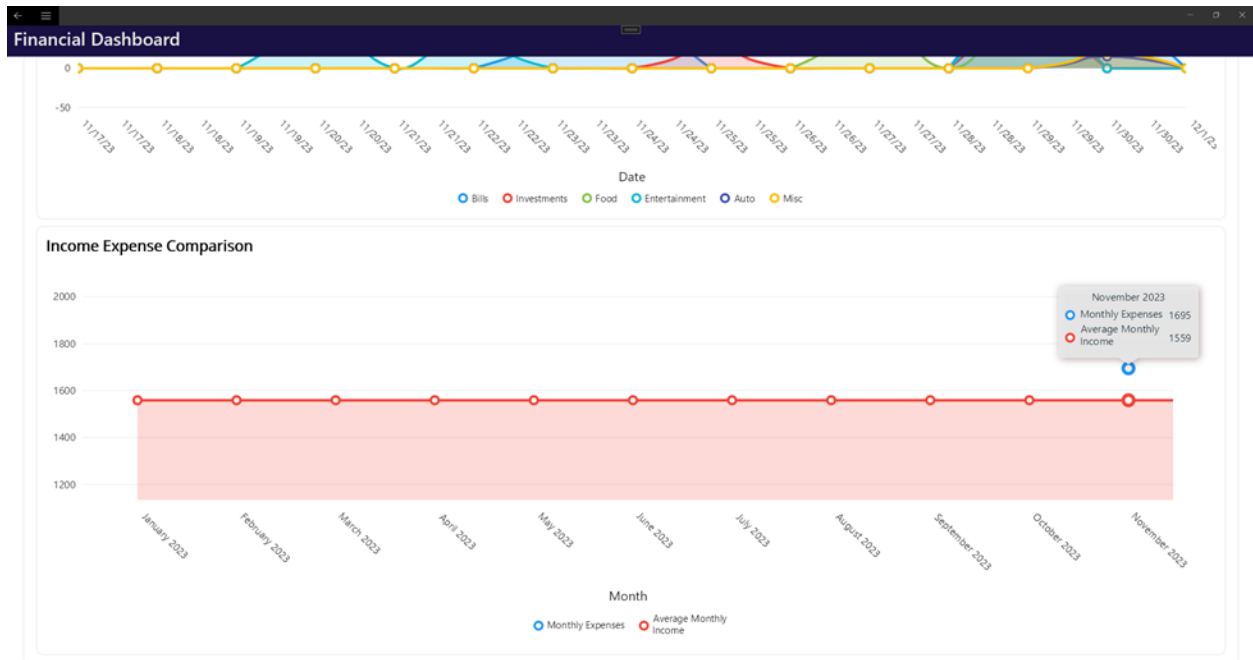


Figure 12: The CampusCoin Dashboard page 3

Contributions

Will started the project by creating the first template, which required elements like setting up Entity Framework, implementing the framework for dependency injection, and creating the initial Azure cloud hosted SQL database. He developed the database tables and schematics, assigning automatically generated primary keys, appropriate foreign keys, and indexes. These functioned as the foundation around which later work was constructed, providing the backbone of our project. Will also worked on integrating the code with entity framework and successfully integrating it with the SQL Server database. He then created the main home page, dashboard page, and numerous services and initial data validations. He also designed the graphs and charts which visualize the users' financial data. Dependency injection was also implemented into the code by Will, which improved the readability, modularity, and testability of the code. He also designed the graphs and charts which visualize the users' financial data. In addition to these core contributions, Will also developed the main tab drag out UI, the main color scheme and logo, and the creation of an easy use page navigation system. These user-facing elements were essential to the overall success of the project, as they made the application more accessible and user-friendly.

Bailey primarily worked on the base and extended functionality of the program, in addition to creating the registration page, account settings page, password reset page, and numerous services and parts of the validations. The registration page required setting up the five required user input fields and ensuring that the validation was done correctly by using the validation attributes he and Will created. Bailey also worked on making sure that all the features on every page worked, such as the account verification process during signup, the remember me button and persisted login, the forgotten password functionality, and signing out. He also created an email service to be set up to accomplish the sign-up process, where it generated a verification code that is sent to the user's email. Entering this code into the application allows the email to be authenticated and the user to sign up. Bailey also put security measures in place like hashing and salting passwords. To do this, each user has a distinct salt generated during the signup process. The password is then hashed and salted before being sent to the servers. This procedure guarantees that plaintext passwords are kept exclusively inside the application (and only for the lifespan of the current page), protecting them from unwanted access. He also added persistent login support, which ensures users stay logged in while navigating through the app, as well as a "Remember Me" function which lets users be remembered when they relaunch the app (skipping the manual login process and going straight to their dashboard).

Andrew primarily worked on the login page and the expenses page. The login page prompts the user for their username and password and checks the database for the entered information, logging them in if it matches a user in the database. The expenses page ties to the UserExpenseData table in the database, which ties to his page's categories, like "food" or "bills" expenses, and has an entry for the cost of the expense, as well as an optional description of the expense. Andrew also worked to ensure the functionality of the expenses page worked as expected, such as allowing multiple entries in a row and confirming the user's submission.

Drashti worked on the initial documents required for this project at the beginning stages including project selection and project plan with the assistance of Avanish to fill in necessary items during the final stages of each document. She contributed designs to the requirements and design document along with assisting with putting together the final draft report, final report, and taking responsibility to submit before the deadline. She also made contributions to the project prototype presentation.

Avanish worked on filling in necessary items during the final stages of project selection and project plan. He also added requirements to the requirements and design documents along with creating the website for the project. In addition, he contributed to the final draft report along with the final report package. Lastly, he added contributions to the project prototype presentation.

Conclusion

In conclusion, our team's experience in developing "Campus Coin," a student-centric budgeting application, was a resounding success.

We faced several challenges in the process, including the difficulty of transitioning from small, narrow-scoped projects to a much larger scoped project created from the ground up. However, our collaborative efforts throughout the course of the semester, and our division of labor into coding and documentation teams, enabled us to overcome these challenges and meet the core functional requirements of our project.

The result of our hard work is a practical, user-friendly application that is tailored for students. Campus Coin provides students with the tools they need to make informed financial decisions based on their income and expenses. This is a valuable resource for students, and we are proud to have played a role in its development, and grateful for the role this project played in our growth as students and developers.