Iscariot Systems

Aden Letchworth Ethan Pan Garret Rogers Jose Chavez Maayan Israel Malik Nasla

***Abstract*—We are creating a system that streamlines member management. The system allows for member tracking, value alignment, performance scoring, a clear chain of command, and data management. For the front-end work, we utilized HTML, CSS, and JavaScript. On the back end, we employed Java with the Spring framework, and for the database, we utilized MySQL. In this paper, we will delve into the analysis, design, and development of the project, along with the process involved in creating the system.**

I. Introduction

Iscariot System is a system that can be used to document everyone in an organization. The system allows for member tracking, value alignment, clear chain of command, and data management. Our goal was to establish a system that maintains persistent and accurate information about its members to eliminate discrepancies. More specifically, these discrepancies pertain to inconsistencies in rank details and community participation, hindering accountability among its members.

A. Problem Description

The organization confronts a variety of critical challenges, including the absence of a comprehensive member tracking system, hindering efficient membership management. Identifying individuals with misaligned values proves difficult, requiring the need for a more precise evaluation mechanism. The lack of a structured performance assessment framework challenges recognizing and rewarding contributions. An unclear chain of command impedes decision-making and initiative execution. Concerns about data management and privacy need urgent attention. Additionally, fostering networking and community building is a priority for a more vibrant and engaged community. Addressing these challenges is vital for maintaining a cohesive, values-driven community and enhancing overall organizational effectiveness.

B. Proposed Solution

In our ongoing commitment to enhance organizational systems, we implemented a tracking and management system. This innovative platform has been thoughtfully crafted to empower our organization by establishing a robust framework that not only facilitates efficient member engagement but also enables a comprehensive performance evaluation. The primary objective behind this system's implementation is to instill a sense of organizational structure, ensuring that each individual is fully aware of their standing and role within the company.

Furthermore, this system introduces a transparent and merit-based approach, allowing every member to understand the specific criteria required for advancing to the next rank or level. It seeks to create a fair and accessible pathway for progression within the organization.

For our leaders, this tool proves to be invaluable, offering a streamlined mechanism to monitor the progress of every individual within their specific branch. This, in turn, facilitates more effective leadership and management practices, fostering a positive work environment. The hierarchical structure instantiated by this system imparts a clear sense of purpose and motivation to every member of the organization, creating a collective drive to achieve excellence and participate in our community.

II. Analysis

A. System Breakdown

Iscariot Systems is a comprehensive system designed to address the challenges stated earlier. The system focuses on member tracking, value alignment, performance scoring, showing a clear chain of command, and data management.

Member tracking is important because it allows everyone in the organization to record their attendance, program participation, and affiliation. This is useful because in the organizations this system is designed for, there are rank systems in place that can only be reached through continuous participation in the organization and the member tracking system allows for the user's activity to be stored. Everything being stored for each user also provides leaders with real-time information that can further assist them in managing their subordinates and assist in making key decisions for the organization.

To address the challenge of identifying individuals with misaligned values, Iscariot Systems helps to identify members whose actions may not align with the organization's values through a performance evaluation mechanism. Included in the performance evaluation mechanism is a performance scoring feature. This feature evaluates members' commitment and trustworthiness and is mainly calculated based on one's attendance results. The scoring system correlates a higher attendance with a higher score, whereas lower attendance corresponds to a reduced score, taking into account the responsibilities assigned to each rank. If someone is absent from an event but has a good reason for it, they can speak to their superior and if they excuse the absence it will not impact the performance scoring. In addition, if someone attends an event but forgets to log their presence a trusted superior may add that attendance entry.

The system also allows for a clear chain of command. This feature mainly benefits the involved users of high ranks because they are in charge of viewing subordinates and making organizational decisions accordingly such as promotions or dismissals.

Lastly, the system ensures secure collection and storage of member data, addressing any privacy concerns anyone may

have. Alongside this the data is persistent which means their will be no confusion on the details of a user or event.

B. Analysis Breakdown

Before jumping into the implementation of the project, we wanted to plan out all of the logistics of the project. We began by settling the implementation requirements. Given our familiarity with Java and MySQL we decided to use this for the back-end and database management system respectively. Additionally, since our goal was to make a web application, we recognized CSS and HTML as a must have. To connect all of these components we knew we'd need something like JavaScript in addition. After this was settled we discussed the system and familiarized ourselves with what we were building. This was an essential step before designing as we could make better design choices accordingly. With this in mind, we discussed the functional requirements of the system. We did this in a similar manner to the event decomposition technique where we tried to understand all of the events the system might respond to. Through this method, we systematically identified various system events and subsequently constructed a comprehensive use case diagram based on these insights. This helped us not only to build the system but decide what was out of the scope for this project. From the use cases, we got a clear outline of the systems processes. To transition this conceptual structure into a more actionable and programmable format, we employed a class diagram. Through this process, we got to view our system in an object-oriented manner where our use cases transformed into an interconnection of objects. This was an essential step in the building process as it not only gave an outline to build off but a new perspective on how we would think about the system. After these steps, we got an idea of the project's scale. This sprouted discussion about any additional technologies that might improve our developer experience and we decided on Spring for the back-end. There was further discussion about using React however we did not want to get lost in new technologies, and already had an unfamiliar road ahead of us.

C. Use Cases and Use Case Diagrams

Initially, we had 12 use cases, but as we progressed, we realized that we had to rethink and needed more use cases that were rank-specific and that better aligned with the requirements of our system. The initial 12 use cases were Log/Update Attendance, View/Update Personal Info, View Subordinates, Answer Polls, Create Polls, View Population of Local Branch, Confirm/Deny Attendance, View Social Score, View Others' Social Score, View Rank Requirements, See Subordinates' Requirements.

Originally, there were only two actors "Rank 2-7 User" and "General User" where "Rank 2-7 User" had a relationship to all use cases and "General User" only had subordinate-specific use cases such as "Create Poll". This seemed fine before we started implementation, but realized things needed to be changed once we started. First, we removed "Create Polls" and "Answer Polls" because we realized that would be

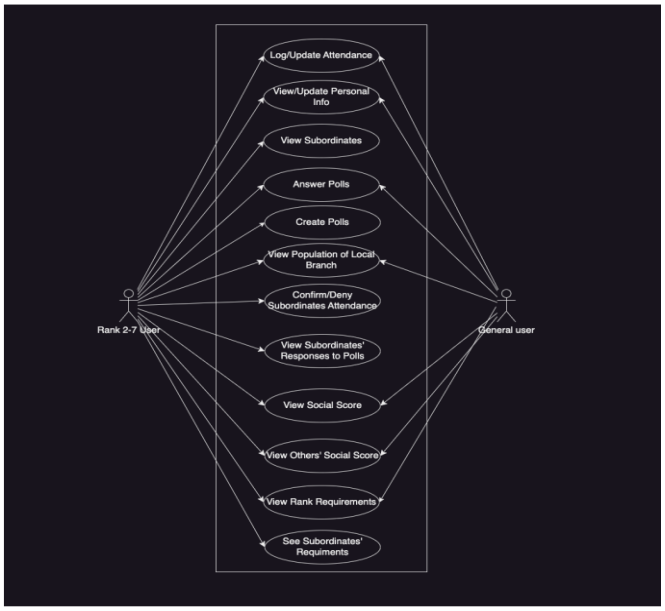

Fig. 1. Original Use Case Diagram.

out of the scope of our project. Secondly, we realized that we generalized too much and needed to break down the ranks more because they needed more specific use cases.

After making the changes, we now have 17 use cases that are more detailed for certain ranks. The actors are now split up into "Rank 0-1", "Rank 2", "Rank 3-4", and "Rank 5-7". The use cases are split up into four sections, relation to all, relation to three, relation to two, and relation to one. The use cases for "relation to all" are Login, Sign up, View Basic Branch Details, View Basic Branch Details, View Social Score Details, View Rank Details, View Account Details, Edit Profile, View Subordinates' Requirements, View Attendance, Clock In. These are general and do not require special permissions. The rest of the use cases are specific to the user's rank in the organization and as the ranks increase, the use cases get more exclusive.

D. Domain Model Diagram

After determining the use cases, we went on to create a Domain Model Diagram. This diagram represents the static structure of a system and the classes' relationships with each other. Here is where all the information about each class is located like their attributes, methods, and associations. It is quite important to plan/visualize these things because they allow everyone working on the system to understand what the goal of the system is. This diagram is where all the main components of the system and planned out and it helps put the people working on the project on track and if anything arises while implementing the diagram can be updated so everyone is on the same page. The diagram we presented consists of four classes: User, Branch, Ranks, and Attendance.

Similar to the use cases and the use case diagram, as we progressed we found that we needed to make some changes

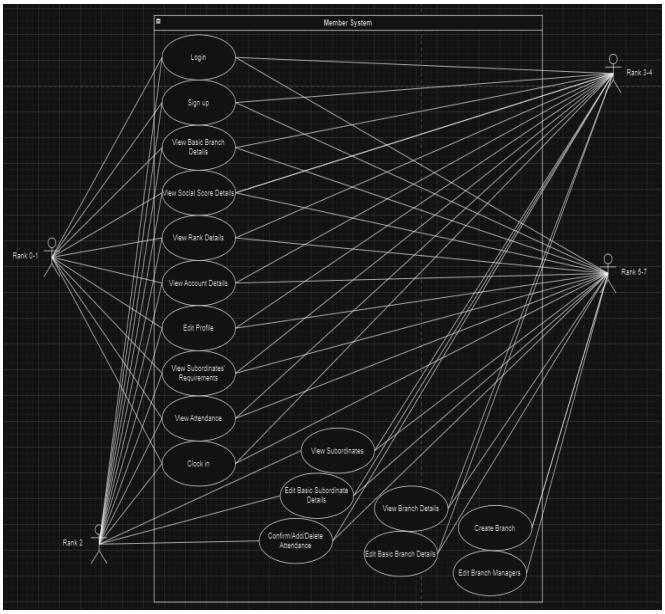

Fig. 2. Updated Use Case Diagram.

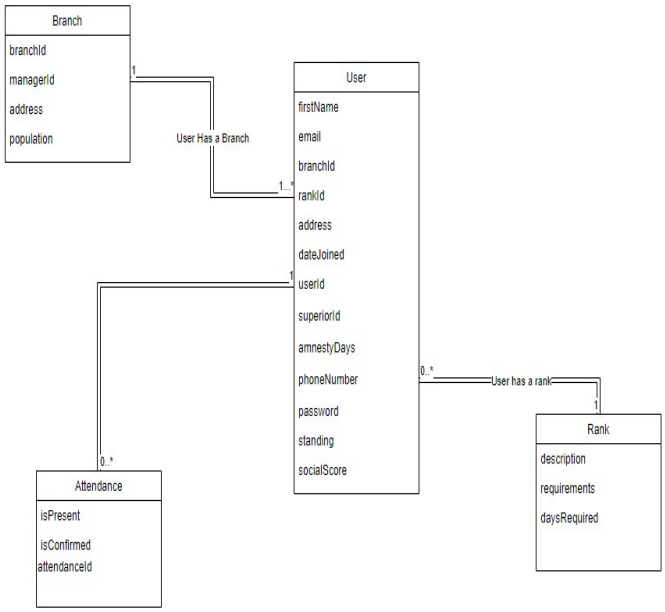

Fig. 3. Original Domain Class Diagram.

to the diagrams. Luckily only minor adjustments needed to be made to the diagram. We had to either add, remove, or update attributes as we realized the need for it for some classes. For the User class, we removed the attribute ”standing” and inferred it with logic and we added the attribute ”lastName”. The Branch class kept the same amount of attributes but we removed the attribute ”population” which could be derived with queries and added ”name”. The Rank class had an attribute added and one edited. The attribute added was ”RankID” and the attribute that was replaced was ”description” changed to ”name”. All these adjustments made happened during the implementation which displayed the benefits of an agile approach.

The first class is the User class. This class is very important, it is the heart of the system. This is where everything about the user is stored. Without this class, none of the implemented features would work because they are reliant on a user's information.

Next is the Attendance class. This class allows for the tracking, ranking, social score, etc. Most features in the system are determined by one's attendance, like social score and rank, and without it, none of the calculations would work resulting in the system being useless.

Third is the Rank class. This class is key because your rank in the organization plays a huge part in what you can or can't do. As one ranks up in the organization, they gain permission to do certain things in the system like being a branch manager and having subordinates, which is something some of the lower ranks cannot do.

Lastly, is the Branch class. This branch allows for internal sections in an organization. Many organizations can spread across the country or even the world so they need to be able to split and track each branch.

All of the classes stem from the User class. User to Branch has a multiplicity of one because every user can only be part of one branch and Branch to User has a multiplicity of one to many because a branch can have one or more users in it. While it may seem like Branch to User should be zero to many, for a branch to be created it has to have at least a manager for the branch resulting in at least one user. User to Rank has a multiplicity of one because while a user can go through different ranks in the organization, they can only be one rank at a time, and Rank to User has a multiplicity of zero to many because a rank can have zero or more people in it. Lastly, User to Attendance has a multiplicity of zero to many because every user's attendance is tracked and they can have an attendance of zero or higher, and Attendance to User has a multiplicity of one because the attendance tracks each user separately.

E. Architecture

The architecture we used for this project was a standard Spring layered architecture. This architecture is a subset of three layered architecture and has a well define best practice and file format. Starting from the Model or Entity section this is where we defined all of our classes in the system. These shape the databases structure with a ORM or Object Relational Mapping. We defined all attributes here and define any foreign keys, primary keys, or any other database constraints. In the data access layer we have the repository section. This section maps to each entity in the database and generates the basic CRUD operations while allowing for more advanced operations to be defined. Only some queries were defined manually but the majority where crafted with the method naming schema of Spring Data JPA. In the service layer of the architecture we accessed our repository classes and defined

any business logic on top of it. The service layer is the data access point for the rest of the back end, it's an intermediary to the repository layer. We finally had the controller layer where we defined our REST endpoints for the front-end to access. These endpoints were then called by our front-end JavaScript with fetch requests and manipulated the HTML pages accordingly.

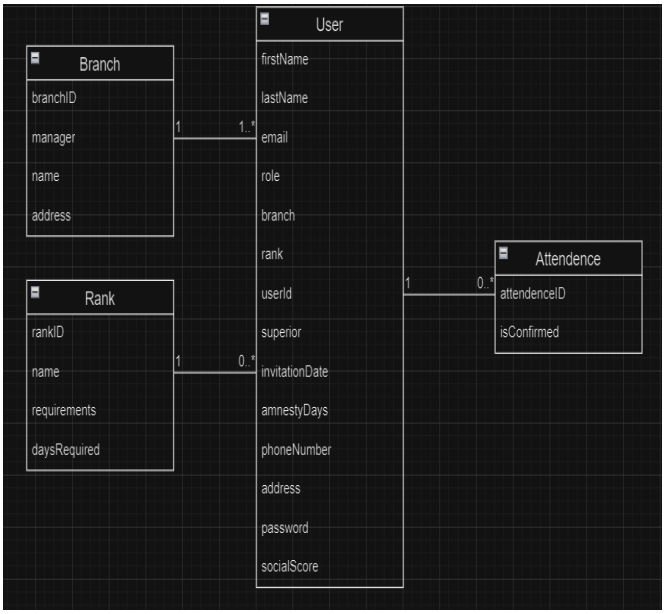

Fig. 4. Updated Domain Class Diagram.

III. Findings and Suggestions

A. Findings

Working on a project of this caliber, there was plenty of issues that arose that we didn't anticipate. Planning ahead with all the diagrams and descriptions is something that is integral to smooth development, which we didn't realize beforehand. Especially if someone has never worked on a project like this, it is essential to plan beforehand and communicate. Without all of the steps that came before the design and implementation stage, the process would not have gone as smoothly. All the steps we took before gave us a blueprint to follow and if we were ever stuck or lost, we can look back at the diagrams to guide us or spark some ideas in our mind.

We also took advantage of speaking and learning from people more knowledgeable. We took advantage of resources like the professor, YouTube and colleagues who were more familiar with the technologies. This is definitely one of the best things you can do. It is inevitable for novice web developers to get to points in the development process where they are stuck and have no idea what to do. Consulting with people who have experience and picking their brains about the project definitely helps keep everyone on the right path.

B. Future Goals

When we started the project, we were a little too ambitious with our goals so we had to reduce the scope for the sake of the class, but we would still like to implement the features we were thinking about. The main components we want to add in the future are increased security, working forgot password page, polls, photos, and we want to increase performance.

We attempted to complete some of the ambitious goals we had but we fell short due to time constraints. We have all the back end code for a JWT or JSON Web Token authentication system. We did however achieve the encryption of all passwords however we had the infrastructure to generate and authenticate tokens. However the integration part proved to be very difficult as we'd have to add many additional checks which we didn't have the time for. We also started on a forget password page which can send emails to the user with Spring Mail, but we didn't flesh out the functionality of generating a unique link that expires which the user is able to change their password from. The only way to change your password at the moment is to log in and change it on the profile page.

There were also components that we didn't even start that we hope to have included in the future. Polls are something that we wanted to include because we figured it would be useful if a superior wanted to send one out to their subordinates to get their thoughts on something, but we quickly realized it would be too much for the time we were given. It was more substantial than we anticipated and we encountered difficulties when trying to understand and implement it. When brainstorming it this seemed like it's own subsystem itself which would be very ambitious. We decided it would be best to leave it as an optional feature that we could implement if we have time, but since we did not it will have to be implemented in the future.

We also wanted to include a way to add photos, like a profile picture the user can add or to customize their profile page. This feature isn't too difficult to implement, but it was not a priority since it had nothing to do with the functionality of the system. Although we didn't get to it we designed the user interface with profile pictures in mind. Customization features in general would be a great addition to the system since the users will be frequently using it and we want some sense of self.

Additionally, we want to increase the performance of our system. We want to do a thorough examination of all the JavaScript code to optimize it for improved performance and the ability to handle higher levels of traffic. There is also optimizations we could have made with social score in order to not re-calculate it every time. In addition we had a lot of abandoned code left in the CSS files which made navigation a bit difficult.

IV. Conclusion

A. Challenges

Working on a project of this caliber, with little to no experience, in a group of six, with only limited time to learn and implement everything can lead to its fair share of challenges. One big challenge that we faced the whole time was time conflicts for team meetings. It was very rare that we had a meeting where everyone on the team was able to make it.

This made it difficult to fully discuss everything as a group and often led to decisions being made without everyone knowing about what was happening. Much of the planning that was done in person had to be communicated over platforms like Discord to the members who could not make it to the in-person meetings and like I said, most of the time the decisions were already made and that member didn't have the opportunity to add input.

Another big challenge that arose was the learning curve for new technologies. While some of us had some experience with certain technologies, most of us had to learn completely brand-new software and tools. For example, we had to start implementing the user interface while still learning HTML and CSS so that resulted in us making major changes as we progressed. In addition the resource for Spring were often outdated or not the best quality. This was especially apparent when trying to implement things like security where there was little to no up to date resources.

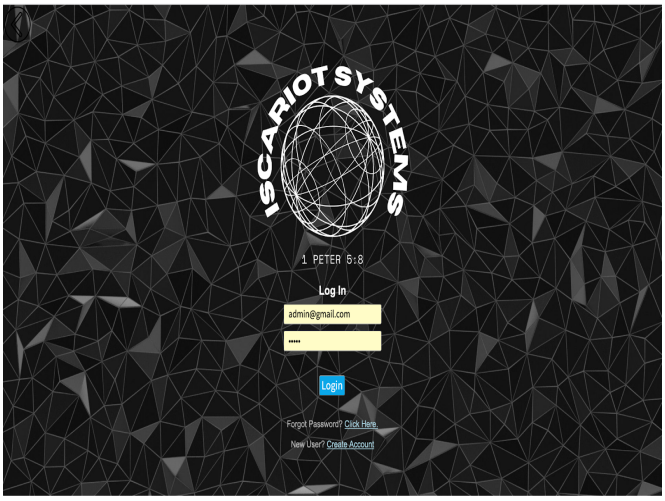

Fig. 5. Original Login Page.

The original login page didn't look like a professional website, you could tell that it was done by someone with limited knowledge.

As we continued on in the project and learned more, we wanted to have a look that was more consistent with the rest of the website once logged in and have a more minimalist and modern look to it.

As shown in Fig. 7 we simplified the login page greatly, but it made it look more professional. The page previously looked cluttered with the background choice being sporadic and busy. We also did the same for the registration page. We took this project seriously so we wanted it to look very professional and traditional to something you would normally see online.

One troublesome challenge was everyone except one person was having MySQL connection issues. We couldn't independently write and test the code, we had to write the code and have the member who could run the program to test the code and then tell us what was wrong so we could fix it and do the process over again until it worked. The main problem with

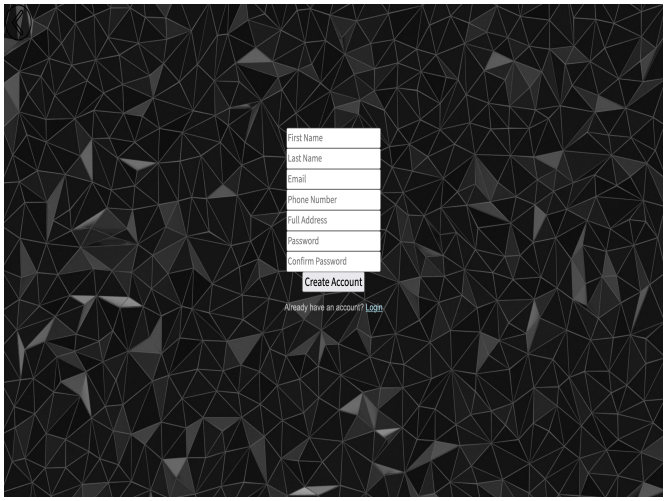

Fig. 6. Original Register Page.

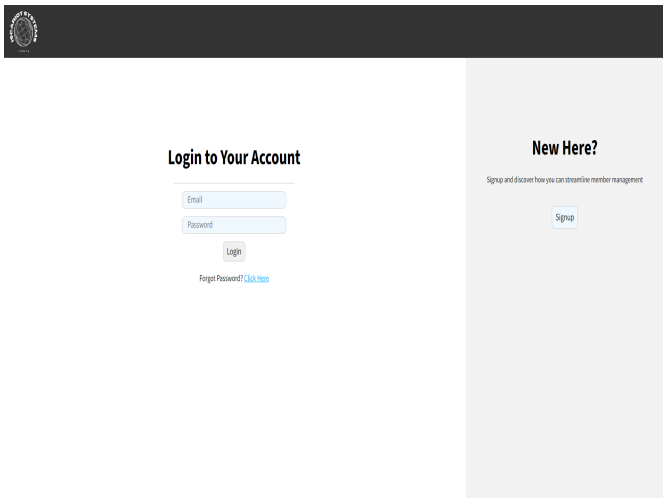

Fig. 7. Original Login Page.

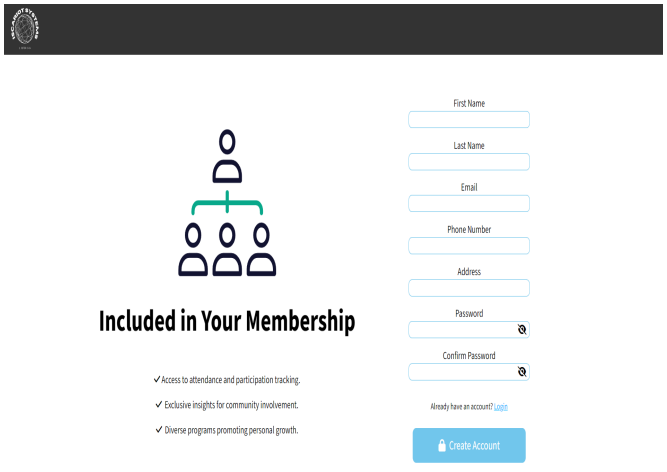

Fig. 8. Updated Register Page.

this is everyone is doing work at different times resulting in sometimes taking hours to get code tested because we had to wait for one person to test it and give the feedback. In addition everyone wasn't working on the same technologies which made it hard to test when you weren't familiar with those technologies. We were able to overcome this by making the testers more involved with the entire project in order to debug and have faster code turnout.

B. What we Learned

Throughout this process, we learned a lot as a group and individually. We learned how to effectively collaborate/communicate with each other. We learned how to collaborate on a scale beyond typical college student projects, emphasizing the importance of design, communication, clear problem statements, group meetings, and team dynamics. As mentioned before it was difficult to find a time where everyone was available for team meetings so we had to learn how to effectively communicate through text and still be able to get the work done.

We also enhanced our web development proficiency through this project. Most of us went from having no knowledge to contributing to building a functioning website. The project provided us with a deeper understanding of web development, particularly in the realm of front-end development. This project required a great amount of web pages and functionality which required us to really understand web development. In addition it also gave us the confidence to use a front-end framework for future projects. Without the foundational knowledge this project provided we wouldn't be able to appreciate a front-end framework in the same way. After this project we have a lot of curiosity about frameworks like React or Svelte which we think would've helped us build something even more significant.

Furthermore, our learning extended to the development of back-end infrastructure for websites and applications, providing us with a comprehensive understanding of system architecture. This acquired knowledge ensures a streamlined approach in our future projects, where we won't need to revisit fundamental concepts like endpoints. Our exploration into Object-Oriented design has significantly enriched our programming skills, offering not just tools but a holistic perspective on coding. Additionally, this journey has ignited our curiosity to explore diverse back-end technologies, such as non-relational databases, fostering a more informed and expansive approach in our forthcoming endeavors.

On the flip side, we gained a real understanding of the time required for implementing each component of a website. This insight will significantly influence and inform our future decisions. It provided us with a more mature perspective on systems, leading us to consider prioritizing a few impactful use cases over the complexity we initially undertook. We now understand doing a few things perfect is better than doing a lot of complex things, which made us admire simplicity.

Lastly, we learned about the significance of CRUD (Create, Read, Update, Delete) operations. CRUD operations provide a straightforward and effective way to manage data in software applications. CRUD operations serve as the foundation for many aspects of application functionality and data management. Everything that is done in our system involves CRUD whether it is adding a user, removing a user, displaying the user's information, displaying a branch's information, changing a branch's name, and so much more. They are all following the format in one way or another.

C. Overview

Iscariot Systems was created to tackle glaring issues that didn't have solutions to them. Its goal was to create a dynamic tracking and management solution designed to empower organizations with a robust platform for efficient member engagement and performance evaluation. From analysis to design to implementation, each step was essential and key to the success of the system. Shortcuts could not be made otherwise the end result wouldn't have resolved the issues at hand. While we did not complete everything we had in mind, we were able to create a system that does everything we wanted it to tackle.