CS 486C - Team Badgers Final Project Report

Northern Arizona University

April 29, 2021

Sponsor: Dave Hopfensperger, Hanz Yeazel, and Glenn Austin

Team Mentor: Sambashiva Kethireddy

Members:
Abdulrahman Alamoudi
Tristan Marcus
Logan Ritter
Robel Tegegne
Yuanbo Xu

Overview:

The purpose of this document is to provide an overview of the entire product. We will go over how we decided to approach this product based on our requirements as well as how we implemented it. We will also discuss our testing process, the timeline of our product, and future work that can be done. The goal of this document is to provide a recap of our capstone project in its entirety.

Table of Contents

1. Introduction 3

2. Process Overview 6

3. Requirements 8

 3.1. Functional Requirements 8

 3.2. Performance Requirements 11

 3.3. Environmental Requirements 13

4. Architecture and Implementation 14

 4.1. Technologies 14

 4.2. Main Components 15

 4.3. Architecture 16

 4.4. Component Detail 18

5. Testing 26

6. Project Timeline 47

7. Future Work 50

8. Conclusion 52

9. Appendix: Development Environment and Toolchain 54

1. Introduction

Since the beginning of March 2020, the world has been dealing with the SARS-Cov-2 pandemic, better known as Covid-19 or Coronavirus. The virus has affected many people's lives causing a massive change in work environments. Most people, even those who have not caught the disease themselves, have been impacted. Millions of people have lost their jobs due to the inability to attend work because their business is not considered essential and the state government is requiring the business to be closed. There are not a lot of places that are capable of migrating their employees to a virtual workspace, but even those who can are stumbling upon a new set of issues. The majority of companies that were able to operate virtually are finding that many employees are struggling with motivation, recognition, and many other things that they use to find daily in the office. Since virtual employees are now abruptly working on a laptop in their home “office”, they lack the social interaction that was constantly present in the workplace of old. In most cases productivity is down due to the lack of oversight, depression is up due to the lack of social interaction, and motivation is down due to the lack of recognition.

Our sponsor for this project is State Farm and more specifically Dave Hopfensperger, Glenn Austin, and Hans Yeazel who represent the Enterprise Technology Department of State Farm. In Enterprise Technology there are hundreds of teams and around 6,000 employees that work on a wide variety of projects to keep the company running smoothly while also keeping State Farm up to date in this ever growing world of technology. State Farm has always been a great place to work for employees primarily because of the respect and recognition they show their employees. In the office there was constant praise and recognition for everyone who produced quality work State Farm is known for. This came in the form of verbal praise, buying coworkers a coffee, or even sometimes ribbons to be displayed on your desk. Since the majority of Enterprise Technology is working at home virtually due to the Covid-19 Pandemic, it is not currently possible to award an employee with anything more than an email, or praise on a skype call. Mr. Hopfensperger and Mr. Austin have proposed that this product can solve a lot of the social problems presented in a virtual workspace. The following are a list of problems employees might face:

- **Belonging -** Staff will lose the sense of belonging within a company, and members of project groups will lack trust and responsibility if their efforts get unnoticed.
- **Morale -** Morale in many companies is at an all-time low as employees deal with the stress of the COVID-19 crisis in isolation. Slogging through their daily tasks from home, employees relying on video-conferencing to provide critical communications.
- **Acknowledgement -** Some companies have tried mechanisms such as newsletters or a "virtual winner's board" to provide the missing acknowledgement and feedback, but this is only mostly ineffective. Many will miss the email or neglect to visit the winners board.

To solve the problems listed above and more, we have developed a social media web application called the *Skills/Knowledge Badging System.* Key features of the system include:

- Secure login / User Authentication
- Badge delivery system (send/receive/remove)
- Badge printing
- Internal social currency (Kudos)
- Kudos Store to spend kudos.
- Email signature integration (Via custom generator)

These features were able to create a well rounded web application that helped solve many of our clients' issues. When State Farm employees use our application it will simulate working in the office as they will be well recognized and rewarded for all their hard work. This will in turn motivate employees to continue to perform at the high level State Farm is known for.

2. Process Overview

To successfully and efficiently develop the Skills/Knowledge Badging System, Team Badgers developed a plan to stay organized and on track. With a large number of documents and non-code related tasks that were valuable to our planning, we decided to dedicate our lead to solely writing documents as we felt he would be best for the job. Next we divided our team into front-end and back-end/database. The reason we decided to divide and conquer is because we felt both our front-end and backend were equally important and both needed to begin development as soon as possible. The roles of our team were as follows:

- **Logan Ritter -** Team Lead, Project Communicator, Documentation.
- **Tristan Marcus -** Development Architect, Quality Assurance, Lead Front-end Coder.
- **Robel Tegegne -** Release Manager, Lead Back-end Coder.
- **Yuanbo Xu -** Front-end Coder.
- **Abdulrahman Alamoudi -** Front-end Coder.

With our team structure in place we were ready to begin developing our product following our plan.

To ensure our team was staying on task and on track we had a variety of meetings throughout each week. We met as a team three times a week where we discussed the current progress of the product and worked on it in a group setting. This allowed us to stay on the same page as one another. We also met with our team mentor once a week where we gave progress updates and received feedback. Our mentor also gave us a rundown on upcoming assignments and helped keep our team on pace. Lastly we met with our clients every other week where we showed them our product and received both functional and visual feedback on our application. This was very valuable to us as our clients are end users of our product. To keep our code organized we used a version control service that easily allowed our team to share and update the code while keeping it organized and safe. For our documents, presentations, and other non-code related items we used a cloud-shared drive that all members could easily access whenever they needed. The tools we used to help control the flow of our team and project are as follows:

- **Discord -** Our team used discord for our three weekly meetings. It was also used as a chat service where questions were asked and answered and notifications were posted.
- **Zoom -** Our team used zoom to meet with our mentor and our sponsors. Zoom provided a quick and easy way to meet and share our product to receive instant feedback.
- **Github -** GitHub was chosen for version control, allowing us to all have access to a shared repository and easily manage updates and merges.
- **Google Drive -** For documents, presentations, and other non-code deliverables, we used Google Drive and its provided tools. This allowed any team member to access and edit our documents any time they needed to.

With the provided team structure and organization tools our team was able to successfully develop our product with little to no confusion.

3. Requirements

Over the course of the Fall 2020 semester, we met with our sponsors every other week. During these meetings we acquired higher level requirements both functional and non-functional. We came up with 6 core functional requirements and many nonfunctional requirements that were applied during development. During our Spring 2021 semester our team developed new functionalities, discovered new requirements, and deemed other requirements unnecessary. This pushed our team to update and refine our functional and non-functional requirements.

3.1 Functional Requirements

Functional requirements are specific functionalities the system must provide to support the domain-level requirements. The following is a list of some of our core functional requirements. This is a summarized list and does not contain every detail. If you would like to see a more detailed version of our requirements please visit our website and check out our Requirements Document.

FR1. Interesting and Responsive GUI

The front end of this application serves as a middle man for communication with the user and their desired task. Having a responsive GUI is necessary for optimal navigation and function carryouts. Specifically the following requirements will serve to make a necessary and sufficient GUI:

- **Single page application -** Users should be able to access all information on a single page within the application. This means incorporating the use of popup windows, mini interfaces, and drop-down navigation to fully optimize the implementation of a single page.

- **Waterfall feed -** Similar to other popular social media platforms displaying to the user all badges earned from every user in a formatted message in the waterfall feed. The feed should automatically update whenever a user earns a badge and should be displayed in order from newest to oldest.
- **User Badge Library -** Displaying to the user all the badges they have earned in the past. The library should automatically update whenever the user earns a badge and should be displayed in order from newest to oldest.

FR2. Secure Login/User Authentication

The Secure Login page will allow users to create or login to their accounts. This will help provide a secure application with secure accounts. Passwords are never stored in the database and we use a json token access system to make sure users are authenticated. The following is the requirement for the secure login/user authentication:

- **User Login and authentication -** Giving the user the ability to log in to the application. The username and password must be authenticated with a specific regular expression and upon successful login, the user is given access to the application. Also allow new users to register an account.

FR3. Badge Delivery System

The badge delivery system will send and award a user conveniently within context to fully recognize that user for their achievements. In other words, badges let others have an idea about the members' experience in a good looking way. The following is the requirement for the badge delivery system:

- **Badge Delivery System -** Giving the user the ability to send and receive badges. Implementing the ability to successfully send a badge to another user and showing the new badge in the user's library alongside updating the waterfall feed.

FR4. Badge Printing

Badge printing system offers a convenient way to print the users badges that they have earned. The following is the requirement for the badge printing:

- **Badge printing -** Giving the user the ability to print a badge they have earned. A user should be able to view the badge as a Standard PDF image type where they can save and/or print it from the application.

FR5. Internal Social Currency

In addition to badges, State Farm has asked us to include some form of social currency that users can send, barter, and spend. We have decided that this currency will be called Kudos and when users earn enough they can redeem them for some form of reward. The purpose of Kudos is to provide another form of appreciation that can be given out more often and in higher numbers than badges. Kudos will be given out for the smaller things whereas badges are reserved for the larger milestones. The following are the requirements for the Kudos System:

- **Kudos Sending -** Giving the user the ability to send and receive Kudos. Implementing the ability to successfully transfer Kudos from one user's bank to another. Showing the updated Kudos count from each user's bank.
- **Kudos Spending -** Giving the user the ability to spend their earned Kudos. Implementing the ability to successfully purchase items from the Kudos store. Showing the updated Kudos amount in the user's bank.

FR6. Email Signature Integration

Badges will be integrated into email through some sort of generated signature. We would like to give the user full flexibility on what they would like to include in their email signature. When creating a signature, a user can type their desired text and then select up to 3 of their badges to be displayed in the signature. Doing this will allow an employee to show off their accomplishments outside of the badging application. The following is the requirement for the email signature integration:

- **Email Signature Generator -** Giving the user the ability to generate an email signature and highlight some of their earned badges. We must show that the user can select up to 3 of their badges and create a custom layout for their email signature. Then the user must be able to save their signature as an image or copy the generated email signature.

3.2 Performance Requirements

This section will cover the performance requirements with respect to functional requirements. Metrics will be determined based on speed, error margin, and accessibility. This is a summarized list and does not contain every detail. If you would like to see a more detailed version of our requirement please visit our website and check out our Requirements Document. The expected outcome of performance follows:

PR1. GUI Performance

The GUI must prove to be easily navigated and accessible to all users. When developing our GUI we must take into account that some users may have impairments such as color blindness, blindness, and other impairments. Therefore we need to make sure our GUI is friendly to all types of impairments and their web assistants. All main sources of information can be accessed by default or within 1-2 clicks. This promotes the idea of a single page application. All elements should load quickly and the application should be easy to learn.

PR2. Accounts Performance

The account system will be highly functional and user friendly allowing for fast login times and little to no hassle with application access. With regards to speed, first time users will take 20-30 seconds to log in. Second time users will minimize login time to 10-15 seconds, and experienced users will be able to log in within 5-10 seconds. This time is user dependent; application loading times will be constant in that logging in to the application will take less than 2-3 seconds.

PR3. Badges

The Badge system will display the badges as images in high resolution so users can note the difference between each badge and what distinguishes it from the other badges. Also users will be able to see the badge information in a clear way that is easy to read and understand the value of what that badge is for. The badge name will be clear to read and easy to find, and the user would be able to find the issuer name of that badge alongside the date issued.

PR4. Social Currency Performance

The social currency (Kudos) system and store will be highly functional and have a few requirements we must achieve. A user's balance should load in 2 or less seconds upon logging in.. Whether sending, receiving, or spending Kudos, all changes will be seen in real time.

PR5. Security

The security of the application will be provided in the Login page where we will use a json token based authentication system to verify every user. This removes the possibility of decrypting encrypted passwords that might otherwise be stored in our database.

3.3 Environmental Requirements

Environmental requirements are non-functional requirements or constraints set on the developers by the clients. Fortunately for Team Badgers, State Farm did not have many environmental requirements for us. This is a summarized list and does not contain every detail. If you would like to see a more detailed version of our requirement please visit our website and check out our Requirements Document. The environmental requirements are as follows:

ER1. Single Page Application

The application must be a single page with popups that perform the functionalities. The reason for this is to make the application simple and easy to understand. Applications that are multiple pages can be confusing and hard to navigate and learn.

ER2. Badges

Badges should be unique and hard to replicate/photoshop. Badges should also be protected with copyright and must be unique to State Farm. Violating copyrighted intellectual property could bring devastating damage to an organization.

4. Architecture and Implementation

The product that our clients with State Farm have tasked us to build is a single page web based internal social media platform that can award their employees with badges and rewards for continuing to provide quality work from their offices, into this new online nature. The team has decided that the product would excel best as a single page application that has a multitude of pop up models to perform all the tasks within the application. We would like our product to look similar to Facebook with a waterfall feed down the middle of the application, and functions to the left and right of the feed, so that it follows familiar design patterns. Building our product this way will provide simplicity and eliminate confusing navigation which will cut the learning curve of the application tremendously.

4.1 Technologies

In order to implement the envisioned product the team has selected a few softwares and tools based on their familiarity as well as how helpful they will be for our product.

- ReactJS - Starting with the front-end UI we have decided to use ReactJs as it provides an easy to use front end framework. With ReactJs we can incorporate HTML CSS and JavaScript without any hassle.
- MongoDB - For our database we have decided to use MongoDB because of its ease of use and the team's familiarity with it. MongoDB will store and update all the data from every user and will be readily available for our front-end to use and manipulate.
- NodeJS (Express) - In order to have communication between our front-end and our database we have chosen NodeJS with the express framework as our back-end language, to perform all our functionality and data updating.

- Gimp - Lastly for unique website and badge design our team has decided to use Gimp, which is a graphic design application. Using Gimp we will be able to create customized badges and unique images for the application.

With these technologies we will be able to implement our key features successfully.

4.2 Main Components

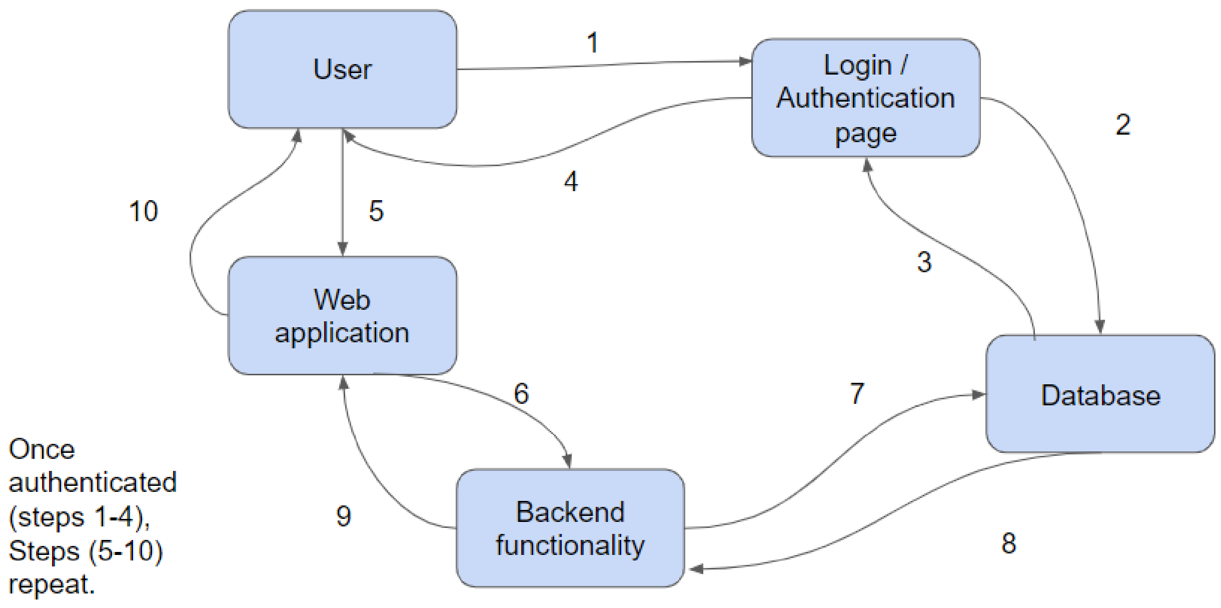
While the team hopes to implement as many features and functions as possible in the time we have available, to make this product usable and functional our team has decided on five main features that the product could not go without. These features include:

- Badge Delivery System - This is the core feature of the product. Our badge delivery system will allow users to send, receive and earn badges for various milestones and accomplishments that can be achieved while working at State Farm. Badges are meant for larger milestones.
- Kudos Social Currency - We will have an internal social currency called Kudos that will allow employees to send their coworkers Kudos to be redeemed for rewards set by their managers. Kudos are meant for the day to day accomplishment.
- Badge Printing - We would like for every user to be able to download and print any of the badges that they have earned. This will allow a user to show off and be proud of their accomplishments.
- Email Signature Generator - Users will be able to generate a fully customizable email signature incorporating their earned badges and Kudos into the signature. Doing this will allow them to show off their accomplishments to their fellow employees.
- User Authentication/ Log in system - Every user will be able to create their own secure account. This will help provide security to the application.

Using these technologies to implement our five main features we will be able to develop a unique and interactive social media application for State Farm.

4.3 Architecture

Alongside the technologies used, we will need a strong architectural plan to reduce road bumps in the future and to get a good grasp on how our product will work computationally. Each of our components for this product will need to communicate with one another for this to be successful.


User
Login / Authentication page
Web application
Database
Backend functionality
1
2
3
4
5
6
7
8
9
10
Once authenticated (steps 1-4), Steps (5-10) repeat.

(**Figure 1 -** Basic Communication flow between each of our components)

In Figure 1, we can see how a user would interact with our product and the order in which each component will be used. The flow in which our users would interact with our product would be as follows (Each step number is associated with the number in Figure 1):

1. A user opens their browser and goes to our web application and is greeted by our login page.

2. Given the user already has an account, the login page will take the information provided by the user, and verify with the database that the user has an account and is authenticated
3. The database will confirm back to the login page that the user is authenticated
4. Then the user will be sent to the web applications homepage.
5. Once in the homepage the user performs one of the various functions.
6. The web application calls the backend functions associated with the front-end function performed by the user.
7. The back-end the performs the function, then updates/pulls from the database
8. The database sends confirmation to the backend
9. The back-end then updates the front end with what it needs to show the user
10. Finally the front end provides the user with updated information, or with the product of the function they chose to perform. (ie. a confirmation that a badge was sent, the reward they redeemed with kudos, etc.)

While it is vital that all of our components are communicating properly, each of the components has its own responsibilities that it needs to perform. For each of our components we will discuss the main responsibilities and features.

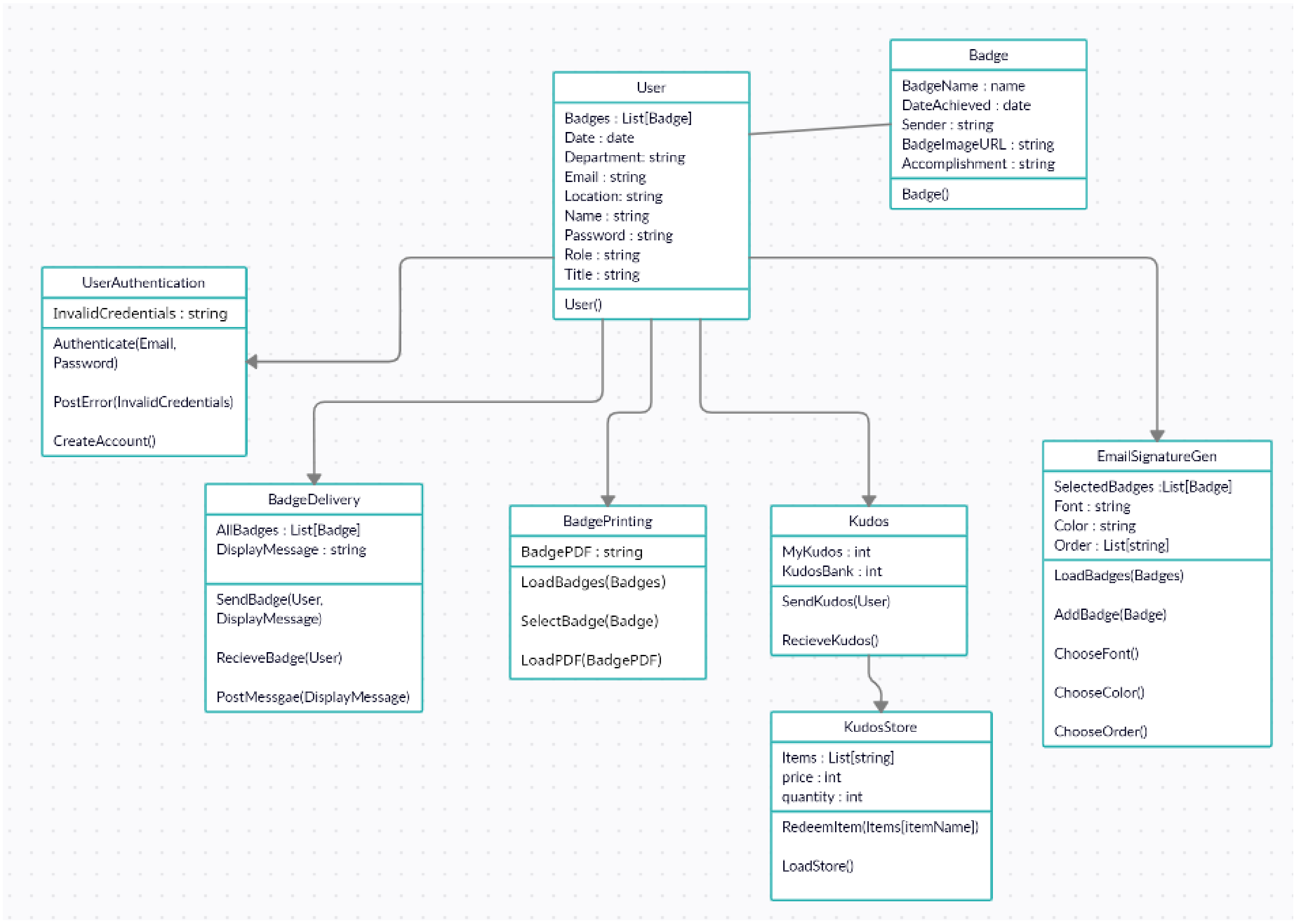
- Login web page - This web page is separate from our application and its main responsibility is to provide extra security and force users to make an account with their State Farm/Google email. A key feature for this page is that it will use Google authentication after an account is created. The login page will communicate with the database to assure a user is authenticated and the right information is sent to the homepage when the user logs in.

- Web Application - This is the front end single page application that the user will interact with. Its responsibility is to provide an easy to use UI and communicate with all the back-end functions and display information to the user. All the five main features of our application will be accessible for use. Since this is a single page application, each of our functionalities will come up in pop up interfaces, or modals, for our user to interact with. Below is a sketch of our envisioned UI (Figure 2).
- Back-end - The back-end will be responsible for communication between the database and the front end. It will also perform all the functionality of our application. This will be the backbone of our product.
- Database - The database is used to store all users information and progress. The database will provide and update information as needed. Communication will occur with the back-end which will push the data to the correct places on the front end.

With the given architectural plan we will be able to build the product as intended avoiding major setbacks and speed bumps that could appear in development.

4.4 Component Detail

As mentioned before, our product will consist of five major components that will provide all the functionality of our application. In this section we will describe the responsibilities and service that each of our components provides as well as how they fit into the architecture. Those components are the badge delivery system, the Kudos system, badge printing, email signature integration, and user authentication. We will also be implementing a Badge object that provides the structure for each of our badges. We have created a UML diagram of our components in **Figure 2** below.

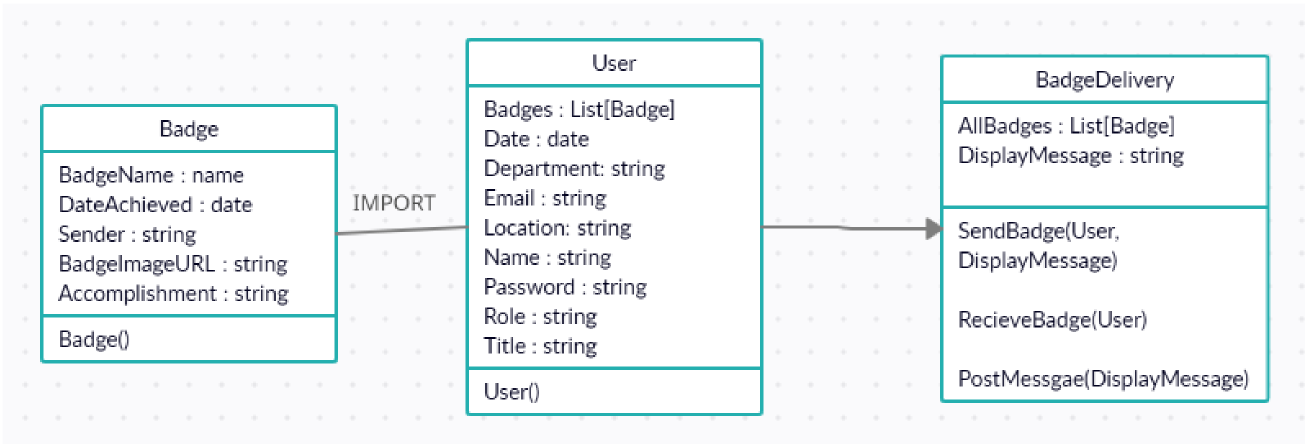


(**Figure 2 -** UML diagram of our system components)

The main model that each of our components extends is the User model. This model holds all the information from the user such as their name, email, title, location, and more. This information is set when the user creates an account. The User also imports the Badge object and stores a list of all the badges the user has earned. The badge object holds information such as the badge name, the accomplishment achieved, the date achieved, and the badge image URL.

Badge Delivery System

Our badge delivery system will be the main component of our application. The system will be an extension of our User which imports the Badge object. Its main job is to provide badges to the user when they earn them. It is also responsible for displaying the badges earned by users in the waterfall feed. A user will be able to see their badges on the front-end on the right panel of the application, as well as the badges they can still earn. Badges can also be sent to other users depending on the badge. This part of our application will be hosted on the front-end but all the functionality will be done on the back-end which will use the database. Below is the UML diagram of our badge delivery system (**Figure 3**).

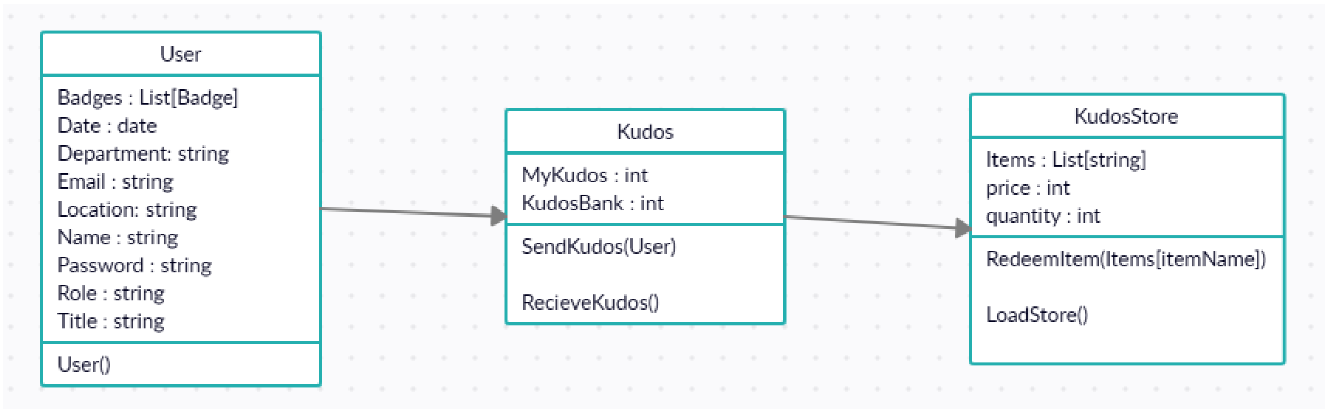


(**Figure 3 -** UML diagram of the badge delivery system)

The badge delivery system inherits all the attributes from User. Additionally it has a list of all the badges available and a message variable that constantly changes to the most recent message that came with the last badge sent. Some of the functions our system can perform are SendBadge(User, DisplayMessage), ReceiveBadge(), and PostMessage(). SendBadge(User, DisplayMessage) will take in the user you wish to send a badge to and the message you want to send with it. It will then call the ReceiveBadge(Badge) function on the receiver's side which will handle the updating of the receiver's badge list. Lastly it will call the PostMessage(DisplayMessage) function to add the message to the waterfall feed which all users can see on the homepage of the application.

Kudos Social Currency

The next component of our application is the Kudos system. This will be a second way for users to earn rewards and recognition from our application. This will work differently from the badges in the sense that Kudos will be given out on the daily by coworkers for many different reasons. Kudos are meant to be sent for smaller accomplishments where the badges are sent for larger milestones. Our Kudos system will be responsible for the sending and receiving of Kudos to every user. It will also be responsible for the Kudos store where managers can set rewards and employees can redeem them. Like the badge delivery system, the Kudos system will be hosted on the front end of the application with all the functionality being done in the back-end with help from the database. Similar to the rest of the components the Kudos system extends the User model. Below is the UML diagram of our Kudos system (**Figure 4**).



(**Figure 4 -** UML diagram of the Kudos system)

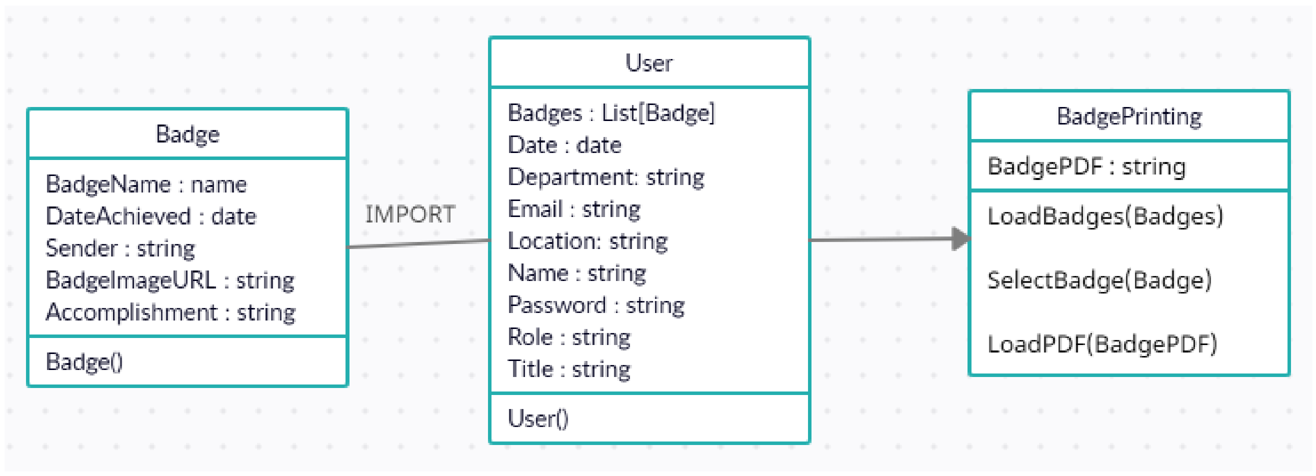
Starting with the Kudos themselves, there will be two different integers kept. The first is called MyKudos. MyKudos is for the users Kudos and will be where Kudos sent to them will go as well as the Kudos they will use to redeem rewards. The second is the KudosBank which is the kudos available to send to coworkers. This amount will be set at a certain number depending on the manager and will be renewed weekly. The only functions of the base Kudos class are

SendKudos(User) which sends Kudos to the provided user and ReceiveKudos() which updates the receiver’s MyKudos. For example, UserA has 1000 Kudos in their KudosBank and they use SendKudos(UserB) to send Kudos to UserB. Let's say they chose to send 100 Kudos, UserA’s KudosBank will now have a balance of 900 and UserB’s MyKudos will have a balance of 100, given they have not received Kudos before

Next we have the KudosStore which has a list of all the items that were set by the manager, the quantity of each item, and the cost of each item. The only functions that an employee can use from the KudosStore are LoadStore() which happens automatically when the page is loaded, and RedeemItem(items[itemname]) which redeems the provided item given the user has enough Kudos. An important detail is that each user's store will be different depending on what their team's manager sets for rewards.

Badge Printing

The next component of our application is the badge printing model. The goal of this component is to allow the user to access the PDF of every badge they have earned so they can download, print and display their badges in the real world. When a user chooses the badge printing model on our application the first thing that will happen is a popup with all their earned badges will appear. They will then select a badge then select the confirmation box. It will then open up the badge PDF in a new window for the user to do with it as they please. This component will be hosted on the front-end and will use the back-end to parse the database for the selected badges. Below is the UML diagram for our badge printing component (**Figure 5**).

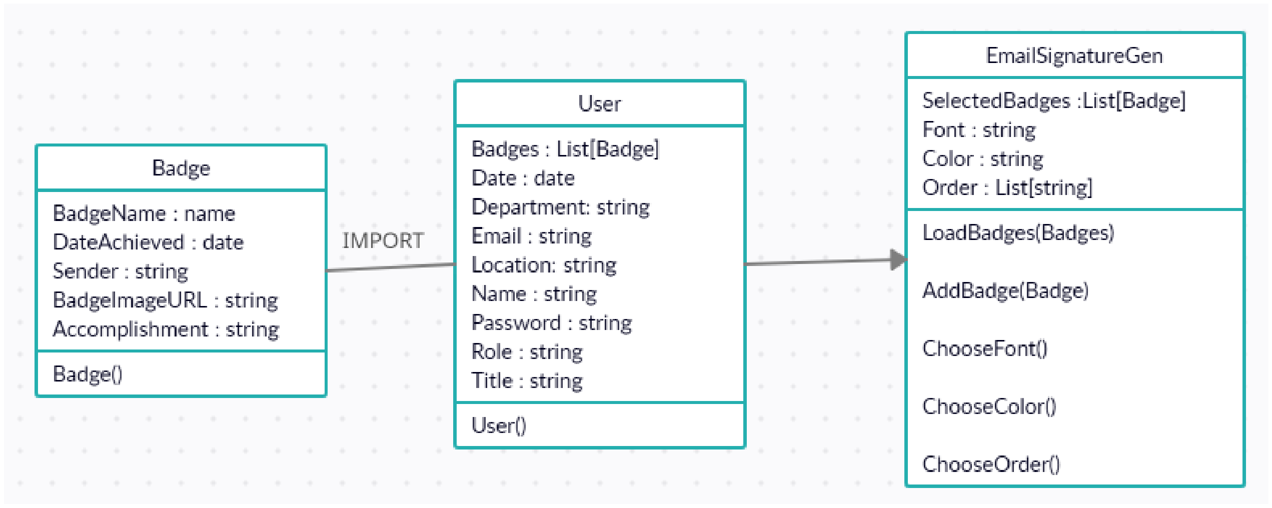


(**Figure 5 -** UML diagram of the badge printing system)

Like the other components the BadgePrinting component extends the User model. The only unique attribute the BadgePrinting model has is called BadgePDF which is a string containing the URL link to the selected badge's PDF. The functions provided by this model are LoadBadges(Badges) which displays all the users earned badges, SelectBadge(Badge) which grabs the users selected badge and sets the correct URL in the BadgePDF attribute, and lastly the LoadPDF(BadgePDF) which brings the user to the provided PDF link.

Email Signature Generator

Another component of our application is the email signature generator which allows users to generate a customized email signature using their earned badges. This will provide another way for the users to show off their accomplishments. Unlike the other components, the majority of the email signature generator functionality will take place on the front end. The only back-end use will be the loading of the badges to be selected for the signature. We want to give our users customizability by allowing them to choose the font, color, and order in which they would like the parts of their signature to be displayed. Once a signature has been generated it will be available to copy as a whole and saved by the user. Below is the UML diagram of our email signature generator component (**Figure 6**).

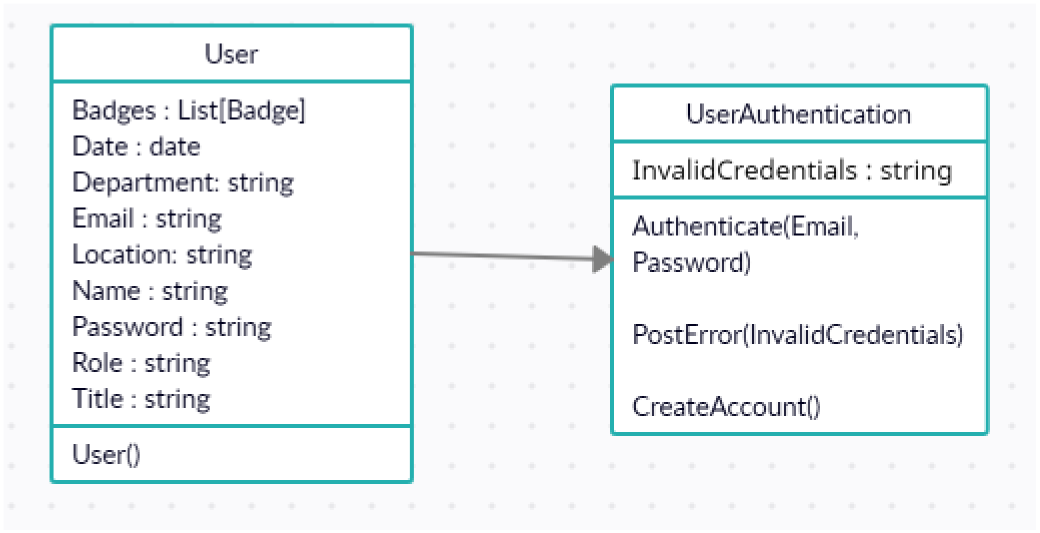


(**Figure 6 -** UML diagram of the email signature generator)

The EmailSignatureGen model extends the User model. It additionally provides four more attributes, SelectedBadges which is a list of the badges selected by the user, font, color, and string. When a user chooses to generate an email signature the following functions will be executed. First LoadBadges(Badges) will load all the badges from the user to be selected. The user can then choose up to 3 badges and AddBadge(Badge) will add the selected badge to the SelectedBadges list. Finally the user will use the ChooseFont(), ChooseColor() and ChooseOrder(), to customize the signature. Now the signature would be ready to save and copy.

User Authentication/Log in system

The last main component of our application is the user authentication system. The sole responsibility of this component is to allow users to create accounts and provide an extra layer of security by forcing users to log in to their accounts every time the application is opened. This is the only component that will be separate from the single page application as we do not want to give someone access to the application if they have not been authenticated. This component will be hosted on the frontend and will speak directly to the database to authenticate users. Below is the UML diagram for our user authentication system (**Figure 7**).



(**Figure 7 -** UML diagram of the user authentication system)

The UserAuthentication model extends the User model. The only attributes used from the User model by the UserAuthentication model are the Email and Password attributes given the user already has an account. Once a user enters in their information the Authenticate(Email, Password) function will be used. If they already have an account and they typed in their information correctly they will be directed to the application homepage. If they enter in the wrong information the PostError(InvalidCredentials) function will be called and the predetermined message, InvalidCredentials, will be displayed to the user. Lastly if the user is new and needs to create an account, the CreateAccount() function will be used which will allow a user to enter in their information and a new User will be created. For more security we will be using a token authenticator. For example when a user creates a new account a random unique token will be generated for their account and every time they log in, the token will be compared to the one in the database to authenticate them.

5. Testing

In this section we will be discussing why and how we are going to perform three types of testing. The first being our Unit testing which will go over each key component of the product and test their functionality. The next type of testing will be our integration testing where we will ensure that all our components are communicating and working together properly. Lastly we will test our usability which will ensure our product works perfectly from the user perspective. For the unit testing, the team will carry out the testing by manually testing each unit and component. They will make a list of all the functionality and go one by one making sure each component does what it's supposed to do and throws errors when needed. For integration, we will make sure that all our components are communicating properly by ensuring that our waterfall feed is displaying actions on multiple users applications and also ensuring that the correct badges and kudos are displaying where and when they need to be displayed. Lastly for our Usability testing, each member will create an account on the application and will use the application as intended. They will perform actions that are supposed to work and attempt to perform actions that should break the application. The majority of our testing will take place in the Unit testing as our application has a lot of different functionality. We only have three components to our application which is why our integration testing will not be long. Performing all these tests will serve to help make our application better and fully functional from all aspects. It will also help weed out any unforeseen errors that may occur when the product is handed over to our clients.

Unit Testing

Unit testing is a form of testing that focuses on each individual component of a product and only tests the functionality of that component. The purpose of unit testing is to get a detailed inspection of a small portion of your application rather than looking at the product as a whole. Most unit testing is just simple input and output verification. If a unit or specific functionality passes our unit tests this proves to the client and potential users that our product is working as intended. If a unit fails this also helps as it shows us developers where fixes and improvements need to be made.

Our goal with unit testing is to identify any components that do not work as intended and fix them as needed. Doing this will allow us to get closer to our final product while making sure each component is working properly. The components of our application that we will be unit testing are; the user login and authentication, the badge delivery system, the Kudos system, badge printing, the email signature generator, and our waterfall feed and badge library. For each of these components we will have 3 phases of our testing.

1. Front-end execution - The first phase is to perform the action on the front end and verify the results.
2. Back-end functionality - The second phase is to ensure the functionality on the backend was performed correctly, this will be done using postman.
3. Database verification - The final phase will be to verify the action performed has correctly updated the database with the new data.

For each component we will discuss the purpose of each component and provide the unit testing descriptions for each of the components functions.

User Login Authentication

The goal of our User Login Authentication system is to manage and provide a way for users to securely create and continue using our application. It also allows us a way to uniquely identify every user who signs up for our application using a token and their email address. The three functions we will test are the Register, Login, and Log out functions. Below are the unit testing descriptions for each of the functions.

Register

A. **Description:** The purpose of this function is to allow a new user to create an account on the application using their email address.

B. **Main Flow:**

    1. Go to the application Login page.
    2. Click the register button at the bottom of the modal to switch from login to register.
    3. Fill in the email field with a valid email.
    4. Fill in the password field with your new password.
    5. Click the register confirmation button at the bottom of the modal to submit your information.
    6. The user will then be directed to the application homepage.

C. **Expected Outcome:** A user will have successfully created an account with the information they have provided.

Login

A. **Description:** The purpose of this function is to allow a user who is already registered to login to the application with their account information.

B. **Main Flow:**

    1. Go to the application login page.
    2. Fill in the email field with a valid email.

3. Fill in the password field with your password.
4. Click the login button at the bottom of the modal to submit your information.
5. The user will then be directed to the application homepage.

C. **Expected Outcome:** A user will have logged in to the application with their pre-existing account.

Log Out

A. **Description:** The purpose of this function is to allow a user to log out of their account before closing the application.

B. **Main Flow:**

1. While logged in to the application, locate the logout button in the top right corner of the screen.
2. Click the logout button.
3. The user will then be directed to the login page and will no longer be logged in to their account.

C. **Expected Outcome:** A user will have successfully and securely logged out of their account on the application.

Badge Delivery System

The goal of our badge delivery system is to provide a way for our users to send and receive badges with other users. This is the heart of the application and will serve as the main rewards system for our users. The function we will be testing is the Send Badge function.

Send Badge

A. Description: The purpose of this function is to allow a user to send a badge to another user using their email address. A user may also send a personal message to the receiver with the badge.

B. Main Flow:

1. Click the “Send a Badge” tab on the left side of the homepage.
2. The Send a Badge Modal will pop up.
3. Fill in the recipient field with the recipients email address.
4. Fill in the message field with a reason as to why you are sending the badge.
5. Select a badge from the list on the right hand side of the modal.
6. Click the Send Badge button.
7. The badge will then be added to the recipients badge library.
8. The application will then create and post a message onto the waterfall feed for all users to see.

C. Expected Outcome: A badge will be sent to the recipients badge library and a waterfall feed message will be posted.

Kudos System

The goal of our Kudos System is to provide the user with a second reward type in the form of a social currency. Users will be able to send and receive Kudos from their own personal bank to other users. Users will also be able to spend their Kudos in exchange for rewards such as candy and coffee. The functions we will be testing are the Send Kudos and Spend Kudos functions.

Send Kudos

A. Description: The purpose of this function is to allow a user to send kudos to another user.

B. Main Flow:

1. Click the "Send Kudos" tab on the left side of the homepage.
2. The Send Kudos Modal will pop up.
3. Fill in the recipient field with the recipients email address.
4. Fill in the message field with a reason as to why you are sending Kudos.
5. Fill in the amount field with the amount of Kudos you wish to send.
6. Click the Send Kudos button.
7. The amount of Kudos will then be subtracted from your bank and added to the recipients bank.
8. The application will then create and post a message onto the waterfall feed for all users to see.

C. Expected Outcome: Kudos will be sent from one user's bank to another and a message will be posted in the waterfall feed.

Spend Kudos

A. Description: The purpose of this function is to allow a user to spend the kudos they have earned on an item in the reward shop.

B. Main Flow:

1. Click the "Spend Kudos" tab on the left side of the homepage.
2. The Spend Kudos Modal will pop up.
3. Select the item(s) that you wish to purchase with your Kudos
4. Click the confirm button at the bottom of the modal.

5. The amount of Kudos that the item(s) cost will then be deducted from the users Kudos bank.

C. **Expected Outcome:** An Item from the shop will have been purchased by the user and a receipt will be provided for the user to print. A message will also be posted in the waterfall feed.

Badge Printing

The goal of the badge printing component is to allow the user to select a badge from their bade library to print and display in the real world. The user may also save the badge certificate on their machine. The function we will be testing is the Select Badge function.

Select Badge

A. **Description:** The purpose of this function is to allow the user to select a badge from their badge library for printing.

B. **Main Flow:**

1. Click the “Print Badge” tab on the left side of the homepage.
2. The Print Badge Modal will pop up.
3. Select a badge.
4. Click the confirm button.
5. The application will open the certificate PDF in another tab for the user to do with as they please.

C. **Expected Outcome:** The user will have access to the PDF of the selected badge.

Email Signature Generator

The goal of the email signature generator is to allow the user to create a custom email signature highlighting the badges and Kudos that they have earned. The user will have full customization of the signature. The function we will be testing is the Customize Signature function.

Customize Signature

A. **Description:** The purpose of this function is to allow the user to customize their email signature.

B. **Main Flow:**

    1. Click the “Email Signature Generator” tab on the left side of the homepage.
    2. The Email Signature Generator Modal will pop up.
    3. Fill out the name field with the name you would like to be on the signature.
    4. Fill out the information field with any extra information you would like to be in the signature. (email, phone number, etc.)
    5. Select a font.
    6. Select up to three badges.
    7. Click the generate button.
    8. The Modal will then preview the generated signature which the user can then copy to their clipboard.

C. **Expected Outcome:** A custom email signature will have been generated for the user to copy.

Waterfall Feed and User Badge Library

The goal of the waterfall feed is to display any action such as badges or Kudos earned on the homepage of the application for every user to see. This allows users to show off the accomplishments they have achieved. The goal of the User Badge Library is to display to the user the badges they have earned. The functions that we will be testing are the Display Feed, Update Feed, Display Library, and Update Library functions.

Display Feed (Initial)

A. **Description:** The purpose of this function is to load the most recent version of the waterfall feed when the application is launched.

B. **Main Flow:**

   1. Log into the application.
   2. The application will pull the most recent messages.
   3. The application will display the most recent version of the waterfall feed.

C. **Expected Outcome:** The most recent version of the waterfall feed will be displayed when the application is first opened.

Update Feed

A. **Description:** The purpose of this function is to update the waterfall feed for all users when one user performs an updating action.

B. **Main Flow:**

   1. A user performs an updating action.
   2. The application Updates the feed with a new message.
   3. The updated waterfall feed is displayed to the user in real time.

C. **Expected Outcome:** The most recent version of the waterfall feed will be displayed given the user was already logged in before the updating action was performed.

Display Library (Initial)

A. **Description:** The purpose of this function is to load the correct badges that the user has earned in the past on the right hand side panel(personal badge library).

B. Main Flow:

1. Log into the application.
2. The application will pull the badges the user has earned from the database.
3. The application will display the users badges in the personal badge library on the right panel.

C. Expected Outcome: All the badges the user has earned in the past will be displayed when the application is first opened.

Update Library

A. Description: The purpose of this function is to update the users personal badge library after they earn a new badge.

B. Main Flow:

1. A user sends you a new badge.
2. The application updates your badge library.
3. The application displays the updated badge library on the right side panel.

C. Expected Outcome: The updated version containing the new badge and all the previous badges earned is displayed to the user in real time.

After all these unit tests are performed the team will have a better understanding of what is working and what still needs improvements. Next we will do our integration testing to ensure that all our components are communicating and working with one another.

Integration testing

After the completion of our unit testing we will move our to our next round of testing which will be our integration testing. Integration testing is a series of tests whose purpose is to expose problems and errors that occur between two interfaces and their interactions with one another. Although each individual component may work perfectly on its own it is essential to do integration testing to ensure that the product also works perfectly together as a whole.

For our project we have three technical interfaces that are in constant communication with each other. Those interfaces are the front-end, the back-end, and the database. To test that these modules are communicating properly with one another and the data passed between them is correct we have divided our testing into three separate groups. The first group of testing will happen on the Kudos. The second group of testing will be on the badges. The last group of testing will be on the waterfall feed, this ensures that our action components are communicating with our homepage.

When performing the integration testing our team will be carrying out two checks after an action is performed to verify the transfer of data is equal on all three interfaces.

1. Front-end - The first check will be to verify the correct value/result is appearing on our applications front-end (for example if Kudos are spent, verify the Kudos bank has lost the amount of Kudos spent). If the correct value/result is showing we also know the back-end is performing properly as the majority of our functionality happens on the back-end.
2. Database - The Second check will be to ensure the database has the same value from the first check. This can be done by checking our database dashboard.

Badges

Our badges are widely used on our application and are the main form of achievement that users can earn. Every user has a badge library that is stored in our database and is a list of badge objects that is constantly being added to. We will need to ensure that these badges in the users badge library can be accessed by both the front and back ends of our system safely. When a function is performed involving a badge, this functionality happens on our back-end and the badges are displayed on our front end for the user to view. There are three different components that utilize these badges and use all three of our interfaces to do so. Those three components are receiving a badge, printing badges, and the email signature generator. Since there are multiple components that use the same badges, we must ensure that the transfer of the badge and all its data, between interfaces, is correctly passed. We can ensure this by performing our integration tests. Below we have described how our interfaces should interact and how we plan on testing the integration between each of our interfaces when a badge is received, a badge is printed, and when an email signature is generated.

Receiving a Badge

A. **Description:** This test will ensure that all our interfaces are working together correctly when a user receives a badge.

B. **Main Flow:**

   1. Create and login to a new account (this is so the users badge library is clear and it is easier to tell if receiving a badge has worked).
   2. Have another user send you a badge and tell you which badge was sent and the message that was sent with it.
   3. **VERIFY** that the badge and message sent by the other user is appearing in your badge library on the right panel.
   4. **VERIFY** in the database dashboard that the badge and message sent are the same badge and message that are showing in your badge library.

C. **Expected Outcome:** The badge and message sent by another user are consistent between interfaces.

Badge Printing

A. **Description:** This test will ensure that all our interfaces are working together correctly when a user prints a badge.

B. **Main Flow:**

1. Login to an account that has preexisting badges (but not too many as it will be hard to tell if all badges are appearing in further steps).
2. Click the "Print a Badge" tab on the left panel.
3. Once the modal pops up, **VERIFY** that all the badges available to print match those that are in your badge library.
4. Select any badge and click submit.
5. **VERIFY** that the PDF the application sent you to is the same badge as the one selected in the above step.

C. **Expected Outcome:** The badges available to print are consistent with the badges in the badge library and the badge selected matches the badge in the PDF that the application sent the user to.

Email Signature Generator

A. **Description:** This test will ensure that all our interfaces are working together correctly when a user generates an email signature.

B. **Main Flow:**

1. Login to an account that has preexisting badges (but not too many as it will be hard to tell if all badges are appearing in further steps).

2. Click the “Email Signature Generator” tab on the left panel.
3. Once the modal pops up, **VERIFY** that all the badges available to select match those that are in your badge library.
4. Fill out the rest of the information fields.
5. Select up to three badges and click submit.
6. **VERIFY** that the email signature generated contains the same badge(s) as the one(s) selected in the above step.

C. **Expected Outcome:** The badges available to include in the email signature are consistent with the badges in the badge library and the badges selected match the badges in the generated email signature that the application has created for the user.

After performing all of these integration tests we can confirm that the data for our badges are correctly being passed between our interfaces.

Kudos

Similar to our badges, the Kudos system will be widely used on our application so we must ensure that the amount of Kudos a user has is safely stored and altered as needed. We will store the amount of Kudos a user has in our database. Any alteration on a user's Kudos, such as sending Kudos to another user, receiving Kudos, or spending Kudos will happen on our back end. The visual representation of a user’s Kudos will happen on the front-end. As you can see we use the same Kudos value on all three of our interfaces so it is vital that we ensure the communication when passing the Kudos data between our interfaces is working properly. Below we have described how our interfaces should interact and how we plan on testing the integration between each of our interfaces when Kudos are sent, received or spent.

Send Kudos

A. **Description:** This test will ensure that all our interfaces are working together correctly when a user sends Kudos.

B. **Main Flow:**

  1. Click the “Send Kudos” tab on the left panel.
  2. Once the modal pops up, **VERIFY** the amount of Kudos available to send matches your Kudos bank on the homepage.
  3. Fill out the recipient, message, and amount fields. Then click submit.
  4. Next, **VERIFY** the amount sent has been subtracted from your Kudos bank.
  5. Lastly, **VERIFY** the amount of Kudos showing in our database matches the Kudos bank amount on the homepage.

C. **Expected Outcome:** All values through the test are consistent between interfaces.

Receiving Kudos

A. **Description:** This test will ensure that all our interfaces are working together correctly when a user Receives Kudos.

B. **Main Flow:**

  1. Have another user send you a specific amount of Kudos.
  2. **VERIFY** the amount sent has been added to your Kudos bank.
  3. Lastly, **VERIFY** the amount of Kudos showing in our database matches the Kudos bank amount on the homepage.

C. **Expected Outcome:** All values through the test are consistent between interfaces.

Spend Kudos

A. **Description:** This test will ensure that all our interfaces are working together correctly when a user Spends Kudos.

B. **Main Flow:**

1. Click the "Spend Kudos" tab on the left panel.
2. Once the modal pops up, **VERIFY** the amount of Kudos available to spend matches your Kudos bank on the homepage.
3. Select any amount of items whose total is less than your Kudos bank amount. Then click submit.
4. Next, **VERIFY** the amount spent has been subtracted from your Kudos bank.
5. Lastly, **VERIFY** the amount of Kudos showing in our database matches the Kudos bank amount on the homepage.

C. **Expected Outcome:** All values through the test are consistent between interfaces.

After performing all of these integration tests we can confirm that the data for our Kudos System is correctly being passed between our interfaces.

Waterfall Feed

Our application utilizes a waterfall feed on the homepage to allow users to see the accomplishments and achievements of their peers. This waterfall feed will be similar to the ones used by twitter and facebook. The waterfall feed will contain messages that show the user when another user has earned a new badge or some Kudos in the form of a formatted message. We must ensure that when a user sends a badge or Kudos that the formatted message is correctly stored and displayed to every user in the application. Similar to our badges and kudos, the messages will be stored in the database in a list that is ordered from newest to oldest and will only contain 50 of the most recent messages (any more would be unnecessary). Below we have described how our interfaces should interact and how we plan on testing the integration between each of our interfaces for the waterfall feed on start up, when a badge is sent, and when Kudos are sent.

Start up

A. **Description:** This test will ensure that all our interfaces are working together correctly on the waterfall feed on startup.
B. **Main Flow:**
    1. Login to the application.
    2. **VERIFY** that the messages in the waterfall feed match the messages in the database dashboard in the correct order.
C. **Expected Outcome:** All messages displaying in the waterfall feed are consistent throughout the interfaces and are appearing in the correct order.

Send a Badge

A. **Description:** This test will ensure that all our interfaces are working together correctly on the waterfall feed when a user sends a badge.
B. **Main Flow:**
    1. Click the “Send a Badge” tab on the left panel.
    2. Send a specific user a badge with a message.
    3. Next, **VERIFY** the message formatted is displaying at the top of the waterfall feed with all the correct information.
    4. Next, have another user **VERIFY** the message formatted is displaying at the top of the waterfall feed with all the correct information on their homepage.

5. Lastly, **VERIFY** the message formatted displaying is consistent in the database dashboard.

C. **Expected Outcome:** The message formatted by the Send a Badge modal is displaying in the correct location for all users on the waterfall feed and is consistent in the database.

Send Kudos

A. **Description:** This test will ensure that all our interfaces are working together correctly on the waterfall feed when a user sends Kudos.

B. **Main Flow:**

1. Click the “Send Kudos” tab on the left panel.
2. Send a specific user Kudos with a message.
3. Next, **VERIFY** the message formatted is displaying at the top of the waterfall feed with all the correct information.
4. Next, have another user **VERIFY** the message formatted is displaying at the top of the waterfall feed with all the correct information on their homepage.
5. Lastly, **VERIFY** the message formatted displaying is consistent in the database dashboard.

C. **Expected Outcome:** The message formatted by the Send Kudos modal is displaying in the correct location for all users on the waterfall feed and is consistent in the database.

After all the integration tests are performed the team will have a better understanding of which interfaces are working and communicating correctly between each other and what still needs improvements. Lastly we will do our usability testing to ensure that our product as a whole is easy to use and is simple to learn.

Usability Testing

Usability testing is different from the other types of testing previously mentioned as it does not test the code but instead assesses how easy the product as a whole is to use. This testing section is critical to our overall product as it determines the quality of it. If the product is easy to use, learn, and navigate then the application passes the usability testing. On the other hand if the product is not easy to use and users struggle with it, the product fails.

When we deliver our product to our clients we want to make sure that it is easy to learn and use so that they do not become frustrated with it. Along with our usability testing, we will be providing our clients with a user manual that will teach them how to navigate through our application. This manual will focus on each individual component of our product and will have a very detailed description on how to use them.

For our usability testing we will have two separate phases or test groups to ensure the products ease of use. The first test group will be the team as we know the product best and can figure out any major issues or errors before we allow other users to use the product. The second testing group will be our clients as they are the users who will be using the product when it is finished. The reason we want an outside group to test our product usability is due to the fact that it is difficult to assess the usability of the product as the developer because we know the ins and out of the code. Having users who have never seen the product perform testing will allow us to fully understand the product's ease of use and its shortcomings.

The plan for the first phase, team testing, of our application will go as followers. We will start by clearing the database so that our application is at a fresh start. Next we will all sign up and login. Then, for the next few days we will be using the product as if we were state farm employees. We will make a list of things that work perfectly, things that work but not as intended or are hard to use, and things that do not work at all or are extremely difficult to use. After we have finished our first round of testing we will go through our list and with the items in the "working but not as intended" group we will fix and find a way to make them easier to use. With the items that do not work at all or are difficult to use we will also fix our errors and work to enhance these items so that they are easy to use and can become a functional component of our product. After our fixes and improvements we will do one more round of testing with our team to make sure the product is working as intended and is easy to use for us.

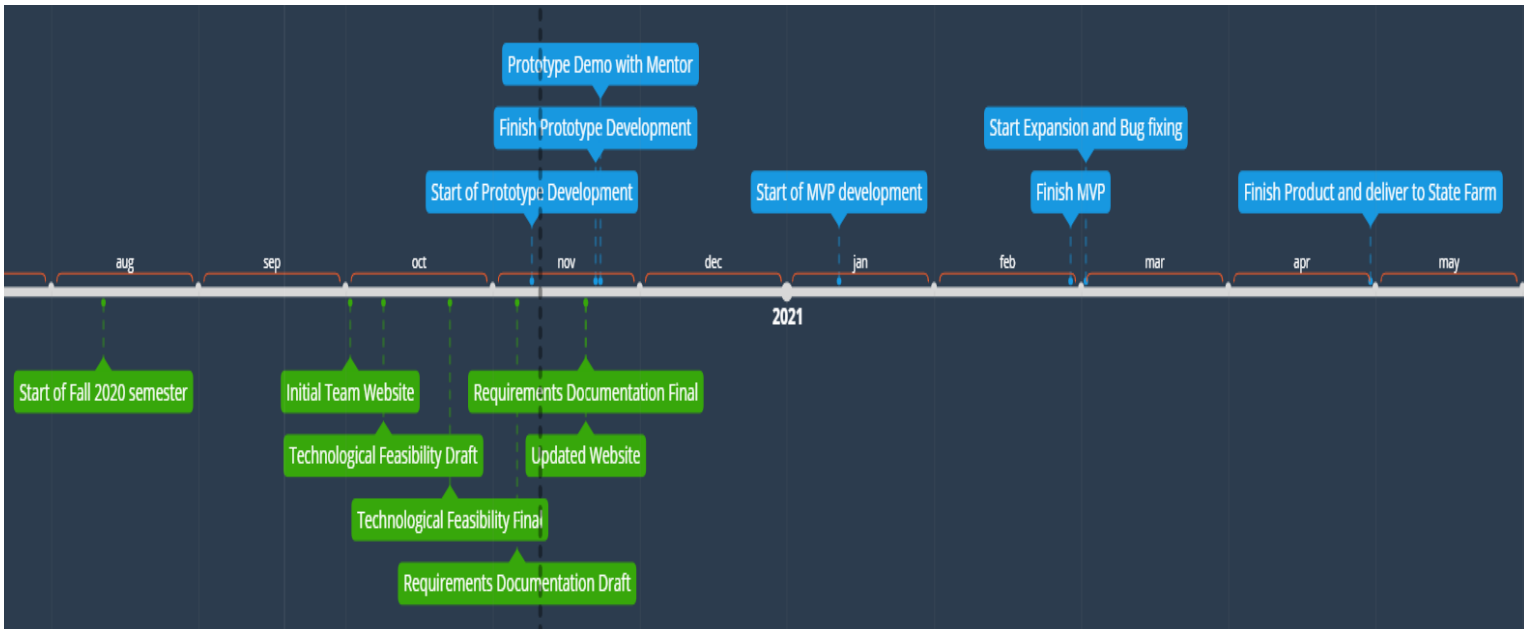
For the second phase of testing, the client testing, we will deliver our product to our clients with the user manual that teaches them how to use the product. The clients will use the product for a day or two to make sure they use every aspect of the application. For each user who tests the product we will ask them to make two lists for us. The first list ranking each function's ease of use. The client will rank the function 1-10, 1 being impossible to use and 10 being very easy to use/near perfect. We would also ask that they provided a reason as to why they ranked the function the score they did. The second list that we will ask the clients to make is a list of things they do not enjoy or things they wish were better. This could be things from the GUI to performance based criteria such as speed. This list will also help the team understand how the product looks from the user perspective.

After the clients finish their testing they will send us their list and the team will analyze each of the lists carefully. We will fix any errors reported and enhance any component that was reported hard to use. We will also weigh the opinions of our clients on any non-functional suggestion from the second list, and make changes where we see fit.

Performing these two phases of testing will allow the team to better understand and address the usability of our product. This is one of the most important testing sections as it determines if the application is usable and likeable.

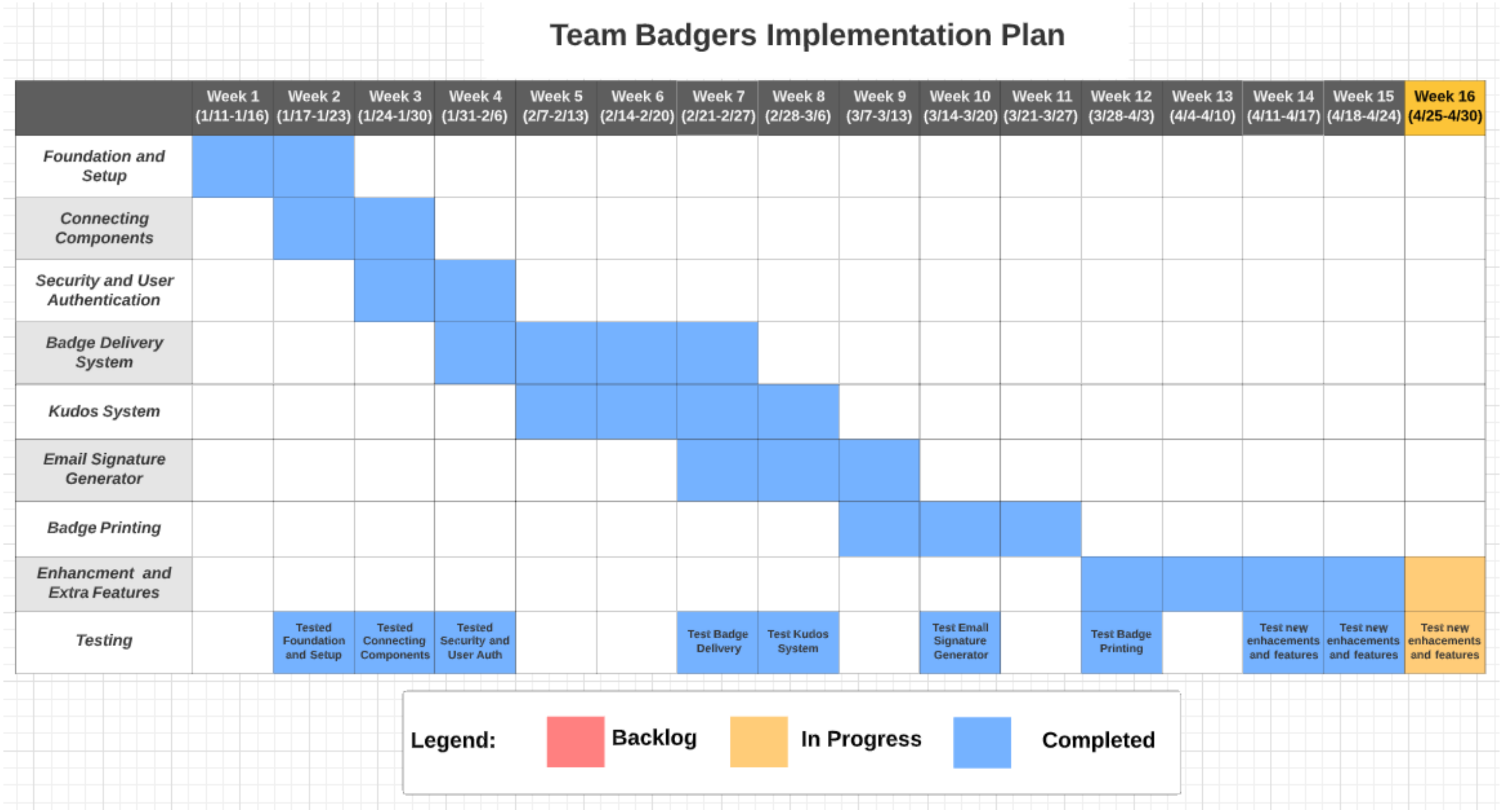
6. Project Timeline

Team Badgers has been developing the Skills/Knowledge Badging System over the past two semesters, Fall 2020 and Spring 2021. Over the past nine months our team has been using many tools to stay on track and on pace with our product and this course. Team Badgers had two kanban boards and two gantt charts to keep organized. Figure 8 below shows the tasks that were completed over the course of the Fall semester.



(**Figure 8 -** Tasks completed in the Fall 2020 semester. Green tasks are documents, Blue tasks are code related.)

The tasks completed in the Fall 2020 semester were primarily documents that set up the foundation of our product. These documents include our Team Standards, Team Inventory, Technological Feasibility, Requirements Specifications, and many more. All of our teams documents and presentations can be found on our team website (https://ceias.nau.edu/capstone/projects/CS/2021/Badger-F20/ProjectDocuments.html). Towards the end of the fall semester our team developed the prototype for our product. Figure 9 shows the tasks involved in developing our product.



Team Badgers Implementation Plan

	Week 1 (1/11-1/16)	Week 2 (1/17-1/23)	Week 3 (1/24-1/30)	Week 4 (1/31-2/6)	Week 5 (2/7-2/13)	Week 6 (2/14-2/20)	Week 7 (2/21-2/27)	Week 8 (2/28-3/6)	Week 9 (3/7-3/13)	Week 10 (3/14-3/20)	Week 11 (3/21-3/27)	Week 12 (3/28-4/3)	Week 13 (4/4-4/10)	Week 14 (4/11-4/17)	Week 15 (4/18-4/24)	Week 16 (4/25-4/30)
Foundation and Setup	🟦	🟦														
Connecting Components		🟦	🟦													
Security and User Authentication			🟦	🟦												
Badge Delivery System				🟦	🟦	🟦	🟦									
Kudos System					🟦	🟦	🟦	🟦								
Email Signature Generator							🟦	🟦	🟦							
Badge Printing									🟦	🟦	🟦					
Enhancment and Extra Features												🟦	🟦	🟦	🟦	🟨
Testing		🟦 Tested Foundation and Setup	🟦 Tested Connecting Components	🟦 Tested Security and User Auth			🟦 Test Badge Delivery	🟦 Test Kudos System		🟦 Test Email Signature Generator		🟦 Test Badge Printing		🟦 Test new enhacements and features	🟦 Test new enhacements and features	🟨 Test new enhacements and features

Legend: 🟥 **Backlog** 🟨 **In Progress** 🟦 **Completed**

(**Figure 9 -** Development plan for our product during the Spring 2021 semester.)

Through the Spring 2021 semester our team worked on documentation alongside the development of our product. Some of those documents include the Software Design Document, Software Testing Plan, Capstone Poster and Presentation, which can all be found on our team's website (https://ceias.nau.edu/capstone/projects/CS/2021/Badger-F20/ProjectDocuments.html). For our products development the team started by setting up the foundation and connecting all our components. Next we made sure our system was secure and then we started developing our major components. Throughout development our team was constantly testing each and every aspect of our product to ensure its functionality is working as intended. When our product was finished we went through a large testing plan as a last measure of security to our product. Finally our team has been adding, testing, and enhancing the product as much as time allows. Following our plans the team was successfully able to develop a successful web application.

7. Future Work

For future work and development on this product there are many routes the clients can take in enhancing the Skills/Knowledge Badging System. After many discussions with our mentor and the clients our team came up with a few possible enhancements that can be made on the product. The possible enhancements our team has thought of are as follows:

- **Multiple Access Levels -** In our product there is room to add a few levels of permissions to create a system hierarchy. The first level, that is already implemented, is the general user level. The next level that could be added is the manager level. This would be above the general user level within the hierarchy of permissions. Some of the permissions that the manager level would have include creating teams or groups, customizing their team's Kudos store, rewarding special badges, and many more. The next level that could be added is the administrator level. This would be the highest level of access to the system and they would have full access to everything on the application and have the capability to add and remove components, edit badge libraries of users, and make internal application changes.
- **Team/Group Creation -** One feature that would enhance the social media aspect of our application would be allowing users to create groups or department teams, similar to the real world. A team would have to be created under a user with manager access. Some benefits to having groups include a custom Kudos store, custom team badges, and a custom application style.
- **Badge Idea Submission Form -** Our application will need to constantly have new badges for users to earn or else it will become stale and users will stop using the product.

One way to add badges would be to create a badge idea submission form where a manager can submit their ideas for badges. These badges with all their detail would be sent to the system admin for review and approval. If the Admin agrees the badge belongs in the application, they will add it to the system. This will allow our application to always have new badge ideas that users actually want to earn as they are created by the users themselves.

- **User Profiles -** Although we do have user profiles currently, they are not customizable. The next step would be to allow users to customize their profiles with information they choose as well as user photos. This would enhance the social media aspect of our application and make it similar to other popular social media platforms.

8. Conclusion

Many companies, since the beginning of March, were forced to switch to a virtual workspace due to the covid-19 pandemic. With this new workspace a new set of problems have been introduced. Those include a lack of recognition and thus their enthusiasm and incentive at work drops sharply. Although some companies have tried mechanisms to provide the missing acknowledgement and feedback, they were only partially effective. State Farm on the other hand has challenged our team to build a gamified web based application to award and incentivise their employees in the form of badges and Kudos.

Team Badgers has developed a web application social media called the *Skills/Knowledge Badging System.* Key features of the system include:

- Secure login / User Authentication
- Badge delivery system (send/receive)
- Badge printing
- Internal social currency (Kudos)
- Kudos Store to spend Kudos
- Email signature integration (Via custom generator)

Our product will be able to solve the majority of the social reward issues State Farm employees are facing in this virtual nature. Our application provides a social media that makes State Farm employees feel like they are still being recognized and appreciated for all their great work even in an online work environment.

Team Badgers is honored and grateful to have been able to work on this product over the past two semesters. We have learned a variety of new skills that we will be able to carry with us in our future endeavors. We would like to thank all our clients from State Farm as well as our mentor and professor for giving us the chance to show off our technical skills on this project.

9. Appendix: Development Environment and Toolchain

This section will go over how our team prepared our environments and configured our machines for development. For this project our members used Windows and Google Chrome/Mozilla Firefox as our browsers. However the Skills/Knowledge Badging System should work on most up to date web browsers.

The following steps will go over how to setup development for the Skills/Knowledge Badging System:

1. Install Git

To begin, it is necessary to install Git. Go to the website https://git-scm.com/downloads and select your operating system, then follow the directions to get Git properly installed on your machine. If you are using a Windows machine, the following commands will be run in Git Bash.

2. Install Node.js

To install Node.js, go to the website http://nodejs.org/ and install the version for your machine.

3. Install npm

To install npm visit and follow the instructions at: https://www.npmjs.com/get-npm

4. Download code repository

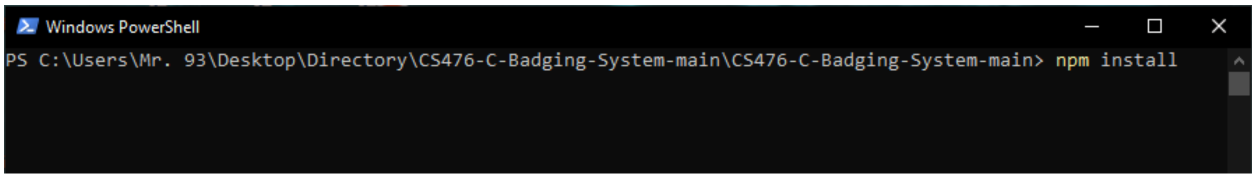
Once given access to the private repository, the source code can be found here:

https://github.com/RobelDev/CS476-C-Badging-System.git

Upon obtaining the source file (CS476-C-Badging-System.zip) through git or simply downloading the source code from github's website, extract the file to your desired directory to begin installation.

5. Install backend dependencies

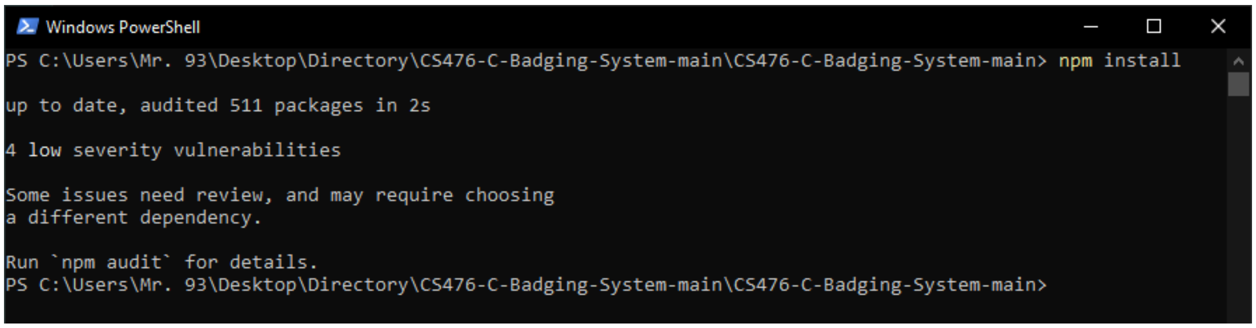
Once extracted, open a terminal and navigate to the "CS476-C-Badging-System-main" directory. First install the dependencies for the back-end by typing 'npm install'



Windows PowerShell

```
PS C:\Users\Mr. 93\Desktop\Directory\CS476-C-Badging-System-main\CS476-C-Badging-System-main> npm install
```

Upon successful installation of node modules, your console should display a similar message to the following:



Windows PowerShell

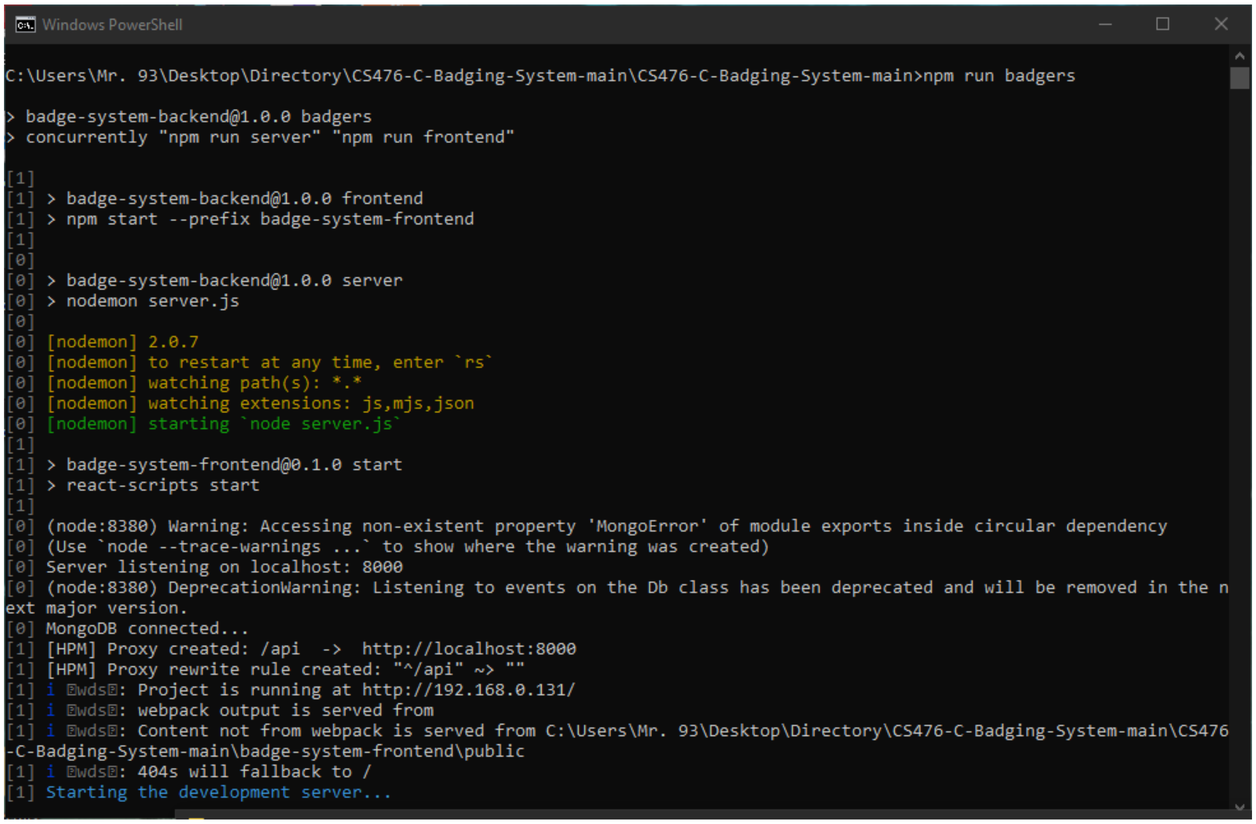
```
PS C:\Users\Mr. 93\Desktop\Directory\CS476-C-Badging-System-main\CS476-C-Badging-System-main> npm install

up to date, audited 511 packages in 2s

4 low severity vulnerabilities

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
PS C:\Users\Mr. 93\Desktop\Directory\CS476-C-Badging-System-main\CS476-C-Badging-System-main>
```

6. Install frontend dependencies

Now install the front-end dependencies by navigating into the 'badge-system-frontend' directory and type 'npm install' once again.

7. Run the server

To start the server navigate back into the "CS476-C-Badging-System-main" directory and run the type the following command 'npm run badgers'.



```
C:\Users\Mr. 93\Desktop\Directory\CS476-C-Badging-System-main\CS476-C-Badging-System-main>npm run badgers

> badge-system-backend@1.0.0 badgers
> concurrently "npm run server" "npm run frontend"

[1]
[1] > badge-system-backend@1.0.0 frontend
[1] > npm start --prefix badge-system-frontend
[1]
[0]
[0] > badge-system-backend@1.0.0 server
[0] > nodemon server.js
[0]
[0] [nodemon] 2.0.7
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching path(s): *.*
[0] [nodemon] watching extensions: js,mjs,json
[0] [nodemon] starting `node server.js`
[1]
[1] > badge-system-frontend@0.1.0 start
[1] > react-scripts start
[1]
[0] (node:8380) Warning: Accessing non-existent property 'MongoError' of module exports inside circular dependency
[0] (Use `node --trace-warnings ...` to show where the warning was created)
[0] Server listening on localhost: 8000
[0] (node:8380) DeprecationWarning: Listening to events on the Db class has been deprecated and will be removed in the n
ext major version.
[0] MongoDB connected...
[1] [HPM] Proxy created: /api  ->  http://localhost:8000
[1] [HPM] Proxy rewrite rule created: "^/api" ~> ""
[1] i ｢wds｣: Project is running at http://192.168.0.131/
[1] i ｢wds｣: webpack output is served from
[1] i ｢wds｣: Content not from webpack is served from C:\Users\Mr. 93\Desktop\Directory\CS476-C-Badging-System-main\CS476
-C-Badging-System-main\badge-system-frontend\public
[1] i ｢wds｣: 404s will fallback to /
[1] Starting the development server...
```

The server is now running and alongside this a new tab in your browser should open with the application. Installation is now complete.