

# Thank-a-Teacher

JIC Team 4115

Victor Henriksson, Dennis Austin, Jessie Rigsbee, Ryan Weng, Bradley Miller, Ami  
Doukoure

Client: Dr. Kristine Nagel

Repository: <https://github.com/CS-3311-Group-4115/Thank-A-Teacher>

# Table of Contents

List of Figures .....	2
Terminology.....	3
Introduction .....	4
Background.....	4
Document Summary .....	5
System Architecture.....	6
Introduction.....	6
Rationale.....	6
Static System Architecture .....	7
Dynamic System Architecture .....	8
Data Storage Design.....	10
Component Detailed Design.....	12
Introduction .....	13
Static .....	13
Dynamic .....	14
UI Design.....	15
Introduction .....	15
Appendix.....	20
API Calls .....	21
Introduction.....	21
Calls and Sample responses.....	21

# List of Figures

Figure Number	Figure Title	Page Number
<a href="#">Figure 1</a>	Static System Diagram	8
<a href="#">Figure 2</a>	Dynamic System Diagram demonstrating creating and sending a card	9
<a href="#">Figure 3</a>	Data Design	11
<a href="#">Figure 4</a>	Static Class Diagram	14
<a href="#">Figure 5</a>	Dynamic SSD to show sending a card	15
<a href="#">Figure 6</a>	Georgia Tech Login Page	16
<a href="#">Figure 7</a>	Search Page	17
<a href="#">Figure 8</a>	Home Page	18
<a href="#">Figure 9</a>	Card Pop-Up	18
<a href="#">Figure 10</a>	Card Design Page	19
<a href="#">Figure 11</a>	TA Inbox	20

# Terminology

Term	Definition	Context
Component	A self-contained piece of code in React that represents a part of the user interface, allowing for reusable and modular code.	Used in React to build UI elements
Middleware	A function in Node.js that processes requests between the client and server, enabling tasks such as authentication, logging, and modifying request/response objects.	Used in Node.js to handle requests
NoSQL	A database type (such as MongoDB) that stores data in a non-relational, flexible schema, often in JSON-like documents instead of tables with predefined relationships.	MongoDB is a NoSQL database
ORM	Object-Relational Mapping: A programming technique that allows developers to interact with databases using objects instead of writing raw SQL queries.	Can be used with MongoDB through libraries like Mongoose in Node.js.

# Introduction

## Background

The current system for thanking a teacher involves a writing-based form that users fill out that is then sent to a TA or professor. The new system for thanking a teacher that we are developing will be more interactive, more fun, and easier to use. Users will have a selection of cards to choose from and may select from a wide range of GIFs to accompany the thank you cards. They will get a prompt email telling them their card has been sent to the recipient, so they don't have to guess if their card reached the right person. Teaching assistants and professors will be able to view all the cards they have received in one place. They will be able to create collages of the cards to enjoy all the positive feedback they have received or print out singular cards as they see fit. We will have preventative software in place to keep hurtful or otherwise problematic material from being sent to TAs and professors. Our prototype can be interacted with through a few missions on Maze that are meant to encompass most of the user interaction with the system. A thorough Figma prototype was used to create the screens for the Maze missions. The Maze missions allowed us to perform heuristic evaluation which is detailed in the "Methods" section. In our findings and recommendations section, we discuss the results of our Maze missions and how we can improve our prototype to better suit user needs. For our revision of prototype section, we discuss how we can improve upon the prototype based on the results of the Maze missions. Finally, we include any supplementary material in our appendices section.

## Document Summary

The System Architecture section provides a high-level overview of how the major components of our system—React-based user interface, Node.js back-end, MongoDB database, and local storage—interact both statically and dynamically. It outlines the data flow from the user's interactions with the front-end, through the back-end logic, and into the database, detailing the communication paths and relationships between these components.

The Data Storage Design section outlines how we handle user-specific data. User submissions (such as thank-you cards and GIFs) are stored using MongoDB, while certain session-specific or user-preference data may be stored in the browser's local storage. It also addresses file

handling, data exchange between the front-end and back-end, and security measures such as validation and prevention of inappropriate content.

The Component Detailed Design section delves into the individual components of our application, describing both their static structure and dynamic interactions in greater detail. Specific classes and methods that make up key elements of our React-based front-end and Node.js middleware are showcased, along with how they interface with the MongoDB database to process user inputs and store or retrieve data.

The UI Design section presents and describes the primary user interface screens of our application. It focuses on the design and user flow for selecting and customizing thank-you cards, adding GIFs, and viewing or printing received cards. User interaction details and how the UI is structured to enhance the user experience are also discussed, including any findings from usability testing using Figma and Maze.

# System Architecture

## Introduction

The Thank-a-Teacher application's goal is to use the MERN (Mongo, Express, React, and Node.js) tech stack to give the user a reactive and smooth experience of thanking a teacher. In order to reach the goal, the application utilizes a layered architecture. The project requires a frontend that provides a good experience for the user, a backend that can communicate with the frontend, database and email service, and a database layer. These layers need to work together to create the Thank-a-Teacher application. Additionally, the layers need to be worked on separately of each other to gain the benefits of smooth development, scalability, and maintainability.

In this section we will review the architecture of the Thank-A-Teacher application. The static system and dynamic system diagrams will show a high-level architectural view as well as describe the architecture patterns being used in the application. We will also give context for all architectural decisions we have made in the development of the Thank-A-Teacher app.

## Rationale

Our team chose a layered architecture approach due to its decoupled and flexible nature. The separation allows for a disconnection between the front-end and back-end of the software, which allows the team to work on different components at once. Another important aspect of the layered architecture pattern are the security improvements. The increased security comes from the fact that backend handles all interaction to the database thus allowing a layer of security between the user and database. This consideration is especially important due to the fact that teaching assistants' emails, students' emails, and personal cards are stored within the database.

## Static System Architecture

In Figure 1, we showcase how each component of our application interacts as a system. The Thank-A-Teacher web application's core components consist of its UI and application logic. These are grouped together because they represent internal elements. Our user interface layer utilizes React, HTML, and CSS to create dynamic and detailed web pages. Following that is our backend which contains the application logic for the application. Our backend uses Node.js and communicates with different external components such as databases or APIs.

Our database of choice is MongoDB due to its ability to store structured data. The database stores the various courses in the College of Computing, the teaching assistants (TA), and their emails. The database's connection to application logic allows us to query TA information upon a student's search.

The last component is the email service. This is a REST API that allows our application to send emails from within the code. By creating an email template that contains the user's designed card we can send an email directly to the recipient's inbox. This API is called via the application logic component which contains our backend.

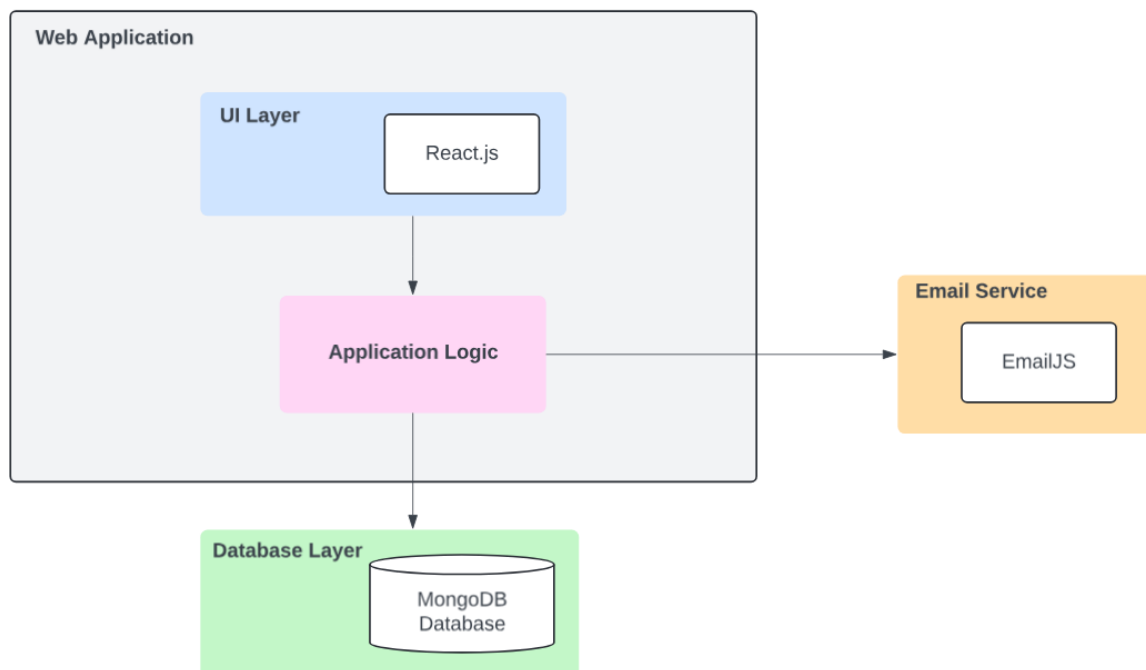


Figure 1 - Static System Diagram



## Dynamic System Architecture

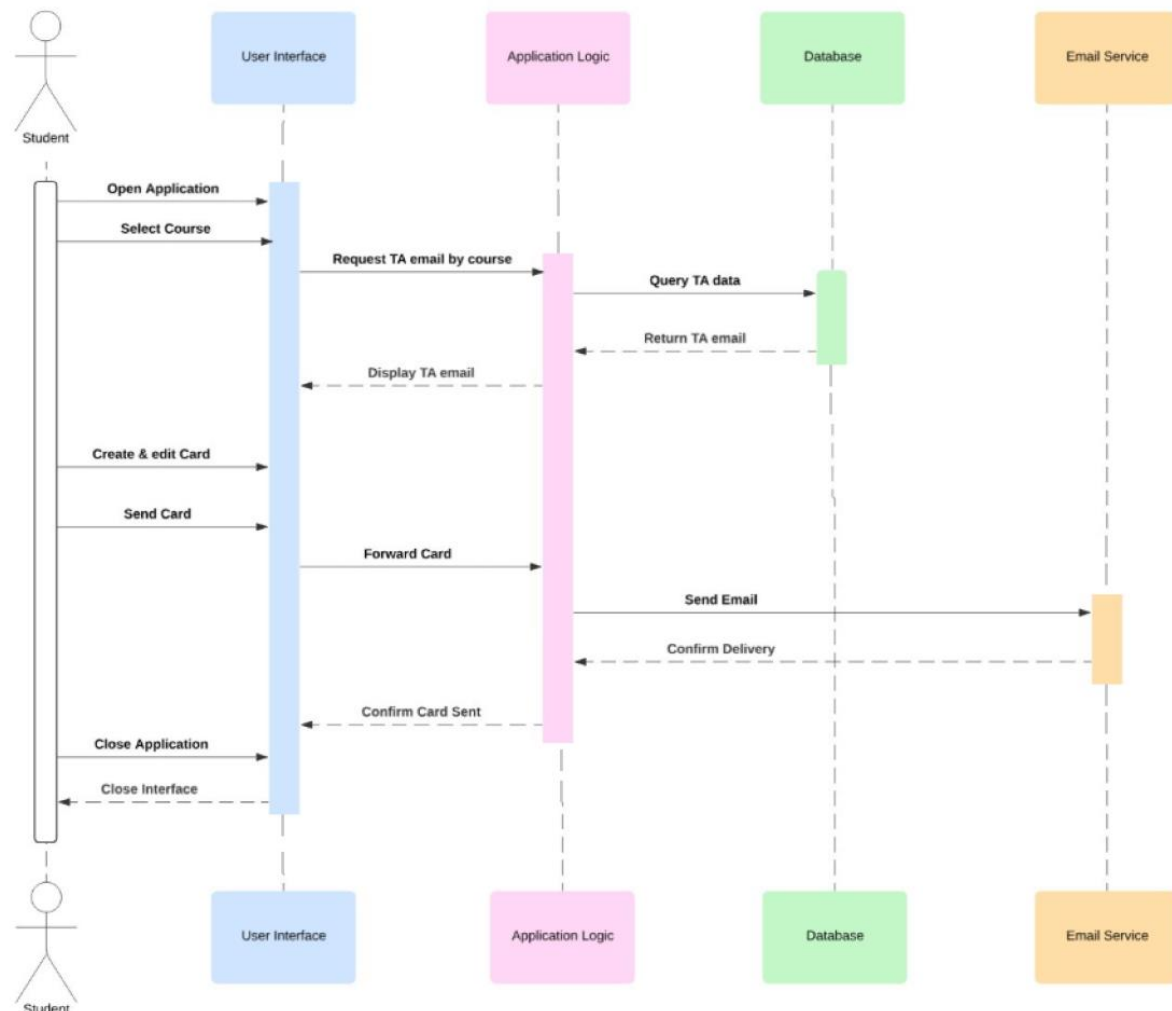


Figure 2 - Dynamic System Diagram demonstrating creating and sending a card

In Figure 2, we can see a system sequence diagram (SSD) that illustrates the dynamic flow of the Thank-a-Teacher application, detailing the student's interaction with the system to create and send a thank-you card. The process starts as the student accesses the application through the User Interface and selects a course prompting a request to the Application Logic for the email address of the teaching assistant (TA) associated with the selected course. Then, the Application Logic engages with the Database to retrieve the TA's email address, which it subsequently presents in the User Interface for the student's use.

Once the student creates and edits the card in the interface, the system advances to the next step, which is sending the card. Upon hitting send, the Application Logic forwards the card to the Email Service, which then processes it and transmits it to the TA's email. Following a

successful delivery, the Email Service communicates confirmation back to the Application Logic, which then subsequently informs the User Interface of the card's successful delivery. And finally, the student then receives this information, and they are given the option to close the application, thus completing the process.

# Data Storage Design

## Introduction

Our data storage technology choice is MongoDB. We chose this database specifically because it allows us to rapidly prototype our design. Our databases has two main classes: 'Users' and 'Cards'. The 'Users' class stores information about regular users and teaching assistants such as userId, name, email, and isTA. If isTA is true, we also track what class that students assist with for our search function.

The next class is our 'Card' class which stores information about the cards created within the application and sent to the TAs. It has the cardId, the userId (from the user who sent the card), and the message attached to the thank you.

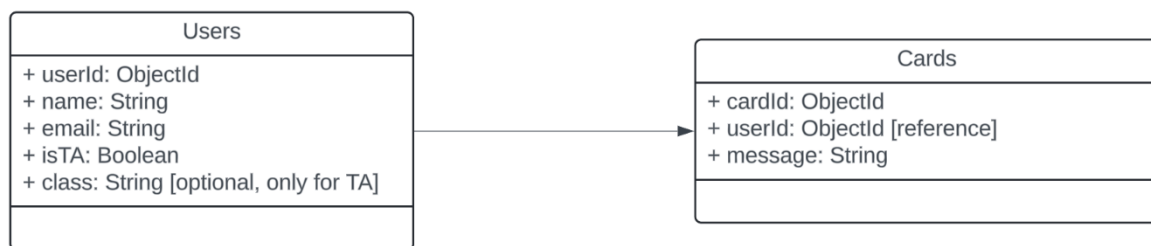


Figure 3: Data Design

The 'Users' and 'Cards' class have a one-to-many relationship where one regular user can send any number of cards. By using MongoDB, we can store our applications information in a JSON like format which is efficient for querying.

## File Usage

The MongoDB storage contains all information about the created cards. Therefore, the contents of the card class contain HTML fragments containing information about the design and structure of the card. The database also contains the PNGs of the blank cards that are used as a template by users.

## Data Exchange and Security

Our application has no direct data exchange between physical devices; however, we do have critical security concerns that arise from the student information we store. To ensure user verification, we confirm that all users are Georgia Tech students by verifying their @gatech.edu email addresses. Since this email is considered sensitive information, we ensure that only secure users can access it. We store this email data in our MongoDB database with strict

access control measures in place to ensure that only authorized individuals and services can access this sensitive information.

Additionally, our application allows users to send personalized messages to teaching assistants via the "Cards" feature. All messages are treated with the same security measures as user emails. These messages are stored in MongoDB, where access is limited by role-based permissions to prevent unauthorized access. In terms of data transmission, our application relies on a React front-end, with a Node.js/Express backend communicating with MongoDB. All communication between the front-end, back-end, and database occurs over HTTPS, ensuring that data is encrypted during transmission and cannot be intercepted or tampered with. Since we handle user authentication via email, we prioritize the security of the entire login and verification process.

# Component Detailed Design

## Introduction

The figures shown below will display the static and dynamic structures of our Thank-A-TA application. We have decided to use a class diagram to display our static behavior which will encompass everything within the application. There are various different pages to be navigated with our application, all serving different purposes, and the static diagram will display the functionalities of these different pages. These include pages such as the Login page, where students can log into the system, the Home page for user interactions like viewing TA cards, and the Base Page. Each class defines attributes and methods relevant to its role, such as `fetchData()` in `TaSearch` for retrieving TA information or `handleAddTextBox()` in `BasePage` for creating dynamic content. To maintain conceptual integrity and align with the overall system architecture, we adjusted the color scheme in the Class Diagram to correspond with the layers depicted in the System Architecture Diagram, creating a unified visual representation of the system.

The dynamic interactions within the application are depicted using a System Sequence Diagram (SSD), which highlights the processes that occur behind the scenes as users interact with the system. The SSD reflects how the application's UI layer, built with React.js, communicates with the Application Logic layer to handle business operations, which in turn interacts with the MongoDB database for data storage and retrieval. External services, such as EmailUS, are also integrated to send email notifications, and the triggers for these workflows are represented within the SSD. By maintaining consistent color alignment between the layers of the System Architecture Diagram and the corresponding classes in the Class Diagram, we ensured a cohesive connection between static and dynamic elements, reinforcing the conceptual integrity of the design. Together, these diagrams provide a detailed understanding of the relationships, workflows, and visual consistency within the Thank-A-TA application.

## Static

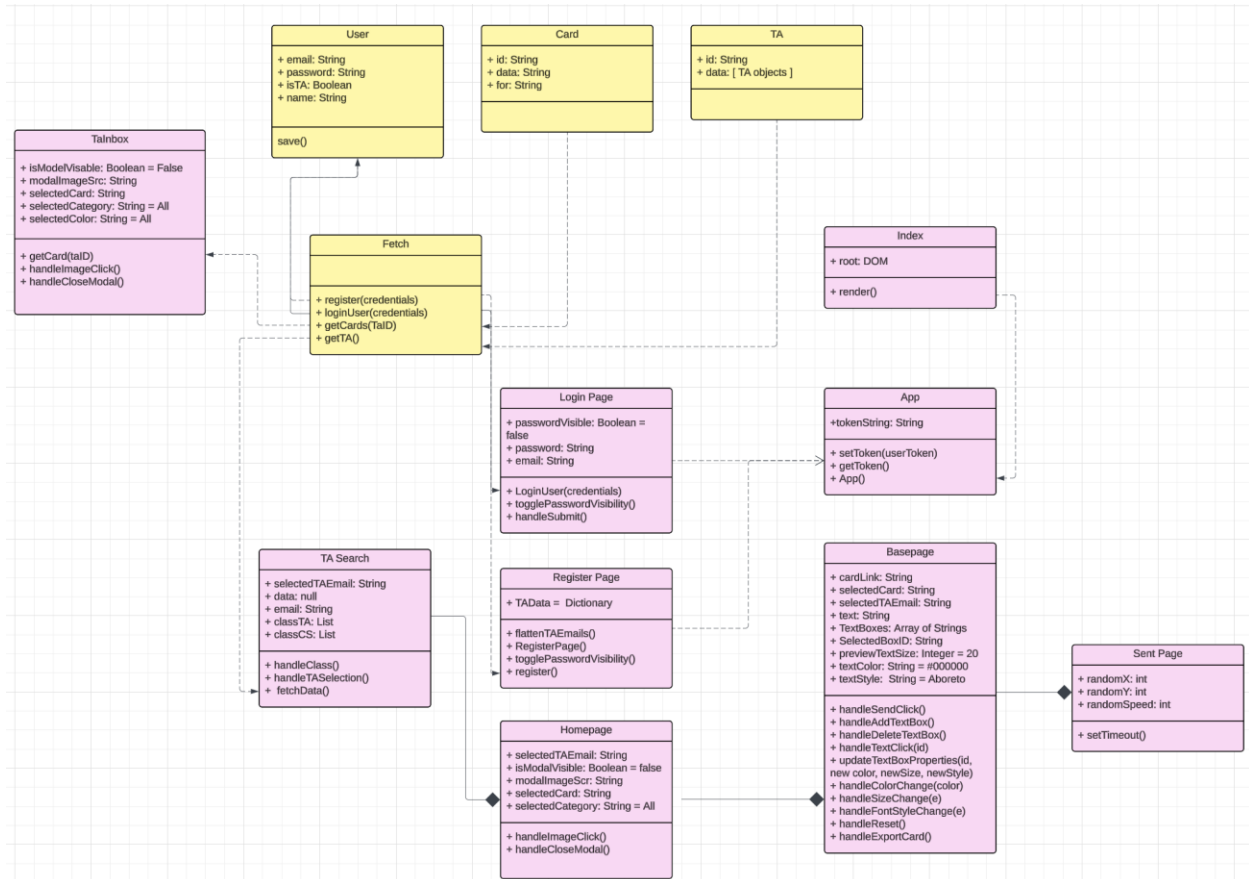


Figure 4. Static Class Diagram

The class diagram above depicts the relationships between classes in our application. It is fairly simple as the classes represent the three pages which navigate to one another (the dotted arrows). There is also an inheritance with BasePage and Index.js. There is a clear indication of the attributes and methods of each class, where one can see how each page operates.

## Dynamic

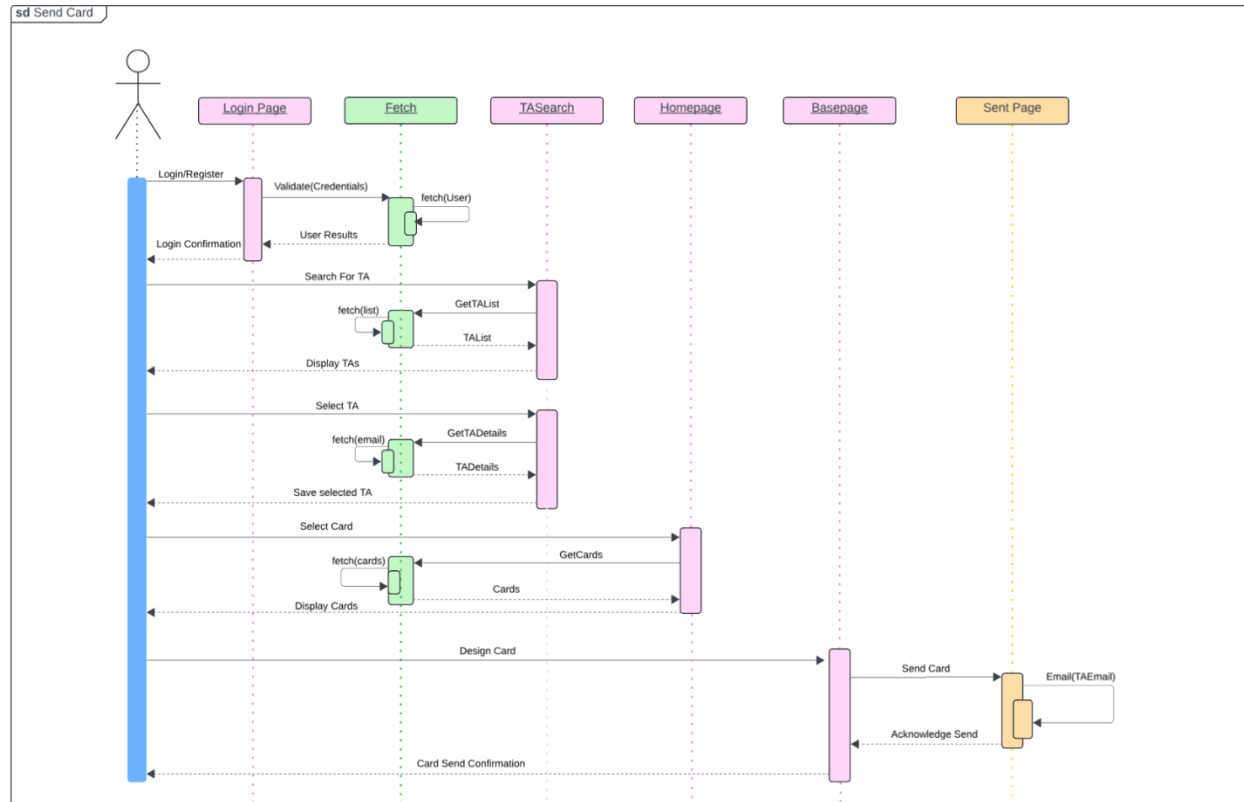


Figure 5. Dynamic SSD to show sending a card

The sequence diagram above shows the sequence of events the user would follow in order to create and send a card to the TA. This involves the main pages: Login Page, TAsSearch, Homepage, and Basepage. The Fetch portion is in association with our database to show retrieval of information about stored cards and TAs. The diagram also shows how the card interacts with the email sender and then notifies the sender. This is done in the Sent page, where an email is sent to the recipient and acknowledgement is sent to the user. The components all interact with each other to create a seamless user experience.

## UI Design

### Introduction

This section will provide images of our application's user interface (UI). While designing our app, we considered the 10 usability heuristics to **guide** our layout. The colors and arrangement of our webpages are meant to be accessible and minimalistic.

The first page the user is presented with is the Georgia Tech Login page (Figure 5 below). It is color coded according to the Georgia Tech colors which allows for a pleasing aesthetic while creating a familiar and recognizable environment for Georgia Tech users, meeting the heuristic of **Match Between System and the Real World**. If users are not affiliated with Georgia Tech, they can select “Sign up here” and register for the application. This provides **User Control and Freedom** by allowing alternative pathways. The page also adheres to **Consistency and Standards** by following familiar login conventions, ensuring that it is intuitive.

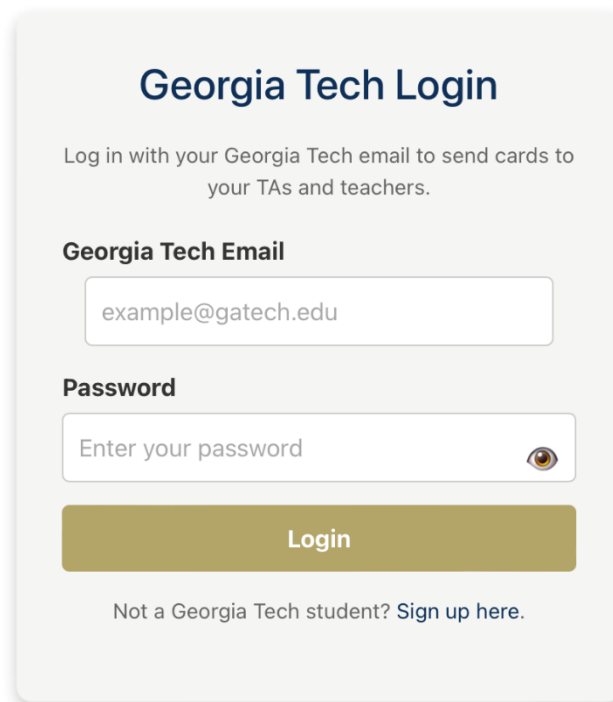
A mockup of the Georgia Tech Login page. The page has a light gray background. At the top, the title "Georgia Tech Login" is displayed in a dark blue font. Below the title, a subtitle reads "Log in with your Georgia Tech email to send cards to your TAs and teachers." in a smaller, gray font. The form consists of two main sections: "Georgia Tech Email" and "Password". The "Georgia Tech Email" section has a text input field containing the placeholder text "example@gatech.edu". The "Password" section has a text input field with the placeholder text "Enter your password" and a small eye icon to the right of the field. Below the password field is a large, olive-green button with the text "Login" in white. At the bottom of the form, there is a link that says "Not a Georgia Tech student? Sign up here." in a small, blue font.

Figure 6: Georgia Tech Login Page

Figure 6, as shown below, is the page where students will search for a specific TA to send a card to. This is the first step in sending a card to a TA. The step indicator helps highlight the current action that needs to be carried out, supporting the heuristic of **Help and Documentation**. The presentation of this page is kept simple with dropdown menus which reduce ambiguity for the user, fulfilling the heuristic of **Error Prevention** by showing only relevant choices for the specific class selected. The dropdowns will populate with only relevant teaching assistants for each class selected to reduce visual clutter, which addresses **Recognition Rather Than Recall**. The design is simple, and each element serves a clear purpose, supporting **Aesthetic and Minimalist Design**.



# Thank-a-Teacher

Select Class ▼

Select TA ▼

Your Email

Next

Figure 7. Search Page

Figure 7 shows the homepage of the application and is the second step in sending a card to a TA. This page hosts the different card designs that a student can choose from. The left portion of the page offers options to filter through the designs by color or category, which helps the user efficiently narrow their choices, meeting the heuristic of **Flexibility and Efficiency of Use**. There is also a reset filters option to remove any applied filters. The banner at the top of the screen allows students to confirm what stage of the process they are at and the ability to return to the search screen. Selecting a card will enlarge it to full screen so that the user can confirm their choice as shown in figure 8.

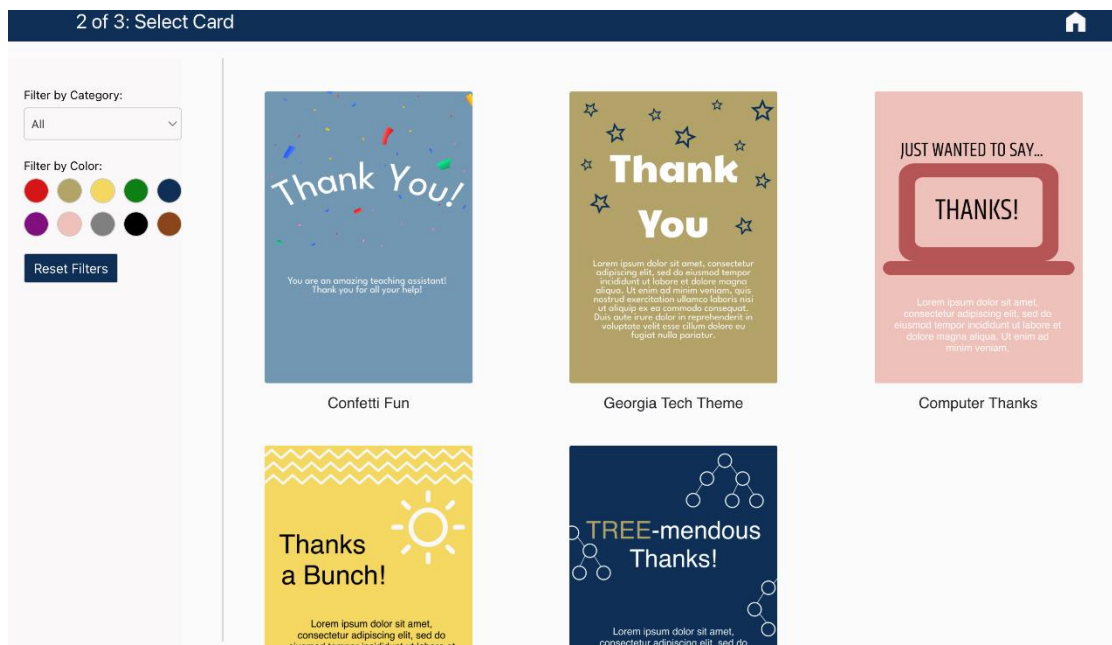


Figure 8. Home Page

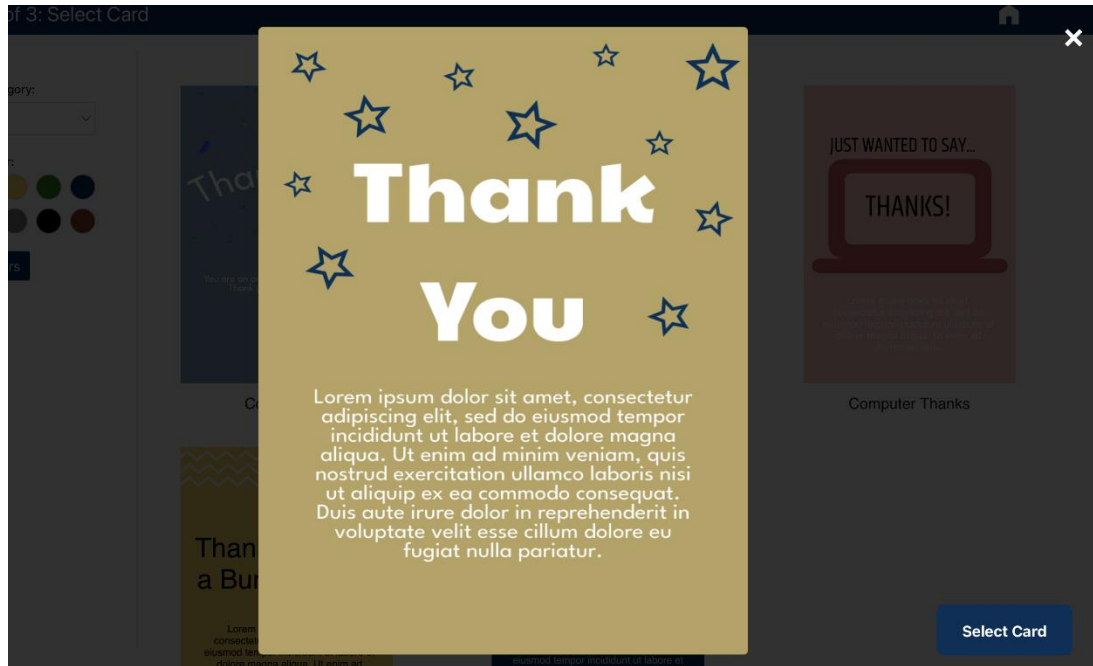


Figure 9. Card Pop-Up

The card design page is depicted in figure 9 and features the last step in sending a card to a TA. This page is where students will customize their selected card. The page is interactive and will update live as text is added to the screen and changed, fulfilling **Visibility of System Status** by showing users real-time feedback. Inputting a different text size, font, or color to the selected text box will reflect on the card. There is a drop-down menu for text font, a slider for text size, and a grid of color choices for text color. Upon sending the card, the system will notify the user if it has been successfully sent, or if an error has occurred, meeting the heuristic of **Help Users Recognize, Diagnose, and Recover From Errors**. The main goal for this webpage was to keep it as functional and intuitive as possible to increase user participation.

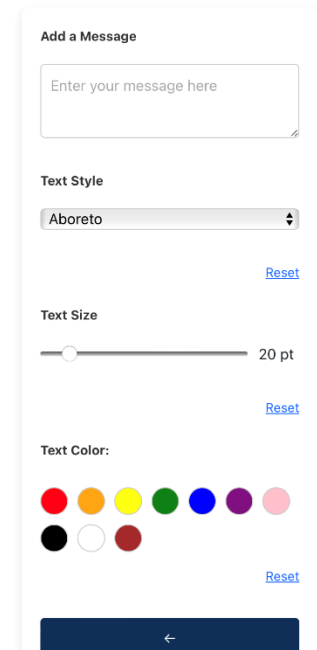
A vertical panel for editing a card. At the top, it says "Add a Message" above a text input field with the placeholder "Enter your message here". Below this is a "Text Style" section with a dropdown menu showing "Aboreto" and a "Reset" link. The "Text Size" section has a slider set to "20 pt" and another "Reset" link. The "Text Color" section shows a row of color swatches (red, orange, yellow, green, blue, purple, pink, black, white, dark red) and a "Reset" link. At the bottom is a dark blue button with a white left-pointing arrow.

Figure 10. Card Design Page

Figure 10 depicts the inbox of thank you cards for a TA. This page is rendered after a TA logs into their account. It is separated into 3 parts, a header, a left sidebar with filters, and a right display area for the thank you cards. Each card is selectable for a larger view if desired by the user. The left sidebar features sorting by category and sorting by color. There is also a large “Reset Filters” button to reset the filters applied to the cards.

Finally, figure 11 shows the TA inbox. This is what is displayed to TAs upon logging in with their credentials. On this page the TA can filter through their received cards using the filters on the left side. Each card on the page shows a preview of the design as well as who sent it and the date. This allows the TA to see all of the thank you cards that have received in one place.

TA Thank You Cards

Filter by Category:

All

Filter by Color:

Reset Filters

Thank You!

Name: Victor H  
Date: 11/4/24

Thank You

Name: Dennis A  
Date: 11/2/24

Thank You

Name: Bradley M  
Date: 11/2/24

Thank You!

Name: Callie R  
Date: 11/1/24

Thank You

Name: Mitchell R  
Date: 10/31/24

Thank You

Name: Hailey M  
Date: 10/29/24

Thank You

Name: Karen R  
Date: 10/24/24

Thank You!

Name: William H  
Date: 10/20/24

Thank You!

Name: Mckinley M  
Date: 10/18/24

Thank You  
cool beans

Name: Rachel R  
Date: 10/12/24

Thank You!  
Thank you for all your help!!

Name: Ryan R  
Date: 10/10/24

Thank You  
You  
Are  
The  
Best

Name: Grace R  
Date: 10/9/24

Figure 11. TA Inbox

# Appendix

## API Calls

### Introduction

The thank a teacher system utilizes a Nodejs backend that exposes an API. The API is used to fetch and store data to the database from the frontend. In this section, the API calls of the thank a teacher system will be demonstrated with the endpoint/URL that is used to interact with the database and the fields of the call with examples.

### Calls and Sample responses

/cards/{taID}

Request Type: GET

This call requests for the cards that are associated with the TaID. The taID is the email of the teaching assistant (TA) that the frontend wants to display to the TA Inbox. Below is an example of a singular card, but if the TA has more than one card then the request will be returned as an array of cards

```
{  
  "id": "673125de0adb444d0e6f1cb8"  
  "data": "data:image/png;base64,iVBORw0KGgoAyAAARgCAYAAARjzKKAAAA..."  
  "for": exampleTA@gatech.edu  
}
```

/card

Request Type: POST

```
{  
  "id": "673125de0adb444d0e6f1cb8"  
  "data": "data:image/png;base64,iVBORw0KGgoAyAAARgCAYAAARjzKKAAAA..."  
  "for": exampleTA@gatech.edu  
}
```

/register

Request Type: POST

The password is salted and hashed and not stored in clear text in the database.

```
{  
  "id": "6734a62784d018440a79e238"  
  "email": testUser@gatech.edu  
  "password": "2b$10$jsTb320JJ4aPodKeEe4n5xDFgMLHMZZvsPUHIAUejp8gTcFqy"  
  "isTa": True  
}
```

The /register endpoint after successfully creating a user account will respond with the users JSON Web Token(JWT) in the response as seen below.

```
{  
  "JWT": "fdsgjd46235jnn43jbn231j2523kn3dij3k1nxo235jhfk23nd23k5n11h35k3"  
  "email": testUser@gatech.edu  
  "isTa": True  
}
```

/login

Request Type: POST

On the request, the password is sent in clear text but then compared to the salted and hashed password in the database.

```
{  
  "email": testUser@gatech.edu  
  "password": myPassword  
}
```

Upon a successful match of the password, the client will be sent the JSON Web Token in the form below.

```
{  
  "JWT": "fdsgjd46235jnn43jbn231j2523kn3dij3k1nxo235jhfk23nd23k5n11h35k3"  
  "email": testUser@gatech.edu  
  "isTa": True  
}
```