Thank-a-TA

JIA Team 5301

Daysen Gyatt, Megha Mishra, Varun Vudathu, Luke Wang

Client: Dr. Kristine Nagel

Repository: https://github.com/CS-3312-Group-1/JIA-5301-Thank-A-TA

Table of Contents

Table of Figures

Terminology

Term	Definition	Context
API	Application Programing Interface: set of rules that allows software applications to communicate, exchange data, and improve functionality	By using APIs, applications can add functionalities and features that would otherwise not be possible or take much more time to implement
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
CSS	Cascading Style Sheets: style sheet language that allows for customizable styling of a document written in HTML	Used to style user interfaces
Express.js	Framework for Node.js that allows for web applications and APIs to be developed	Used along Node.js for our project.
Fork	When the codebase from a software project is copied to then be worked on separately	MariaDB is a fork of MySQL.
GIF	A type of image file format typically used for animated images	

HTML	Hypertext Markup Language: language that defines what content is included in web applications and its structure	Used along with CSS for front-end.
HTTPS	Hypertext Transfer Protocol Secure: Allows for secure communication between a web browser and site via encryption, letting users access websites securely	Our site on Plesk uses HTTPS so users can access it securely.
JavaScript	Programming language that is used for most web development	Our project is written in JavaScript
Key-value Data	Simple method of data storage where information is stored in pairs of a key (a unique identifier) and a value (the data associated with the key)	
MariaDB	Relational database that is a fork of MySQL, allowing data to be stored in flexible ways	Used as the database for this project
MERN Tech Stack	MERN (MongoDB, Express.js, React, and Node.js): Full framework for building web applications that allows both front and back ends to be done in JavaScript.	The original team who we inherited the project from used this for their development.
MySQL	Open source relational database that allows for data to be manipulated using SQL queries	

Node.js	A runtime environment that allows developers to use JavaScript for backend development and gives access to a wide variety of libraries	Used for web development back-ends.
Plesk / Plesk Hosting	Control panel for web hosting that allows users to manage websites, domain names, and server resources	Allows project to be deployed to a @gatech.edu domain name
React	An open-source JavaScript library that allows users to create user interfaces for web applications using components	Used for web development front-ends.
Relational Database	A type of data storage where the data is arranged in rows and columns and uses relationships between the data to make connections	MariaDB, the database used for the project, is a relational database.
SQL	Structured Query Language: programming language that allows data in databases to be managed (stored, retrieved, modified, etc).	
SSH	Secure Shell: Network protocol that lets users connect to remote computers securely	We use SSH to connect to our Plesk database when running the project locally.
SSO	Single sign-on: A method of user authentication that allows one ID to be used to log into multiple different applications	Georgia Tech SSO is integrated with our Node.js application for login.

Introduction

Background

Our project is a web application that aims to make showing gratitude to teaching assistants (TAs) at Georgia Tech more interactive, fun, and intuitive by allowing students to send TAs thank you cards. The web application is built using React, Node.js, CSS, and HTML. Our final product is deployed on Georgia Tech Plesk Hosting services at the following URL: https://thankta.cc.gatech.edu/. We used MariaDB, a fork of MySQL as the database for this application, since GT Plesk requires MariaDB by default. We use GT Plesk functionality to send emails from a GT email address, access our database, and use an external API for profanity screening.

The application allows for login with GT SSO, meaning accounts are linked to students' pre-existing GT accounts. Students will be able to filter TAs by semester and class to select the TA they wish to thank and will have a selection of card backgrounds and attachable GIFs to personalize their cards. When sending a card, the student will receive a confirmation email informing them that the card has been sent and the TA who is meant to receive it will get a notification email. TAs are able to log in to the platform with SSO as well and have a page on the web application to view all the cards they've received. As TAs are also students, TAs can send cards to their own TAs as well. We have implemented preventative software to screen for profanity before cards are sent to TAs. Administrator accounts have a separate page where they will be able to update what GIFs students are able to include in their cards as well as update the list of TAs for any given semester.

Document Summary

The System Architecture section provides a high-level overview of the interactions between the major components of our system: React-based User Interface, Node.js backend, MariaDB database, and external APIs. We provide both a static diagram, which illustrates how the application is organized, and a dynamic diagram, which demonstrates an example scenario of the application completing common tasks.

The Data Storage Design section describes the database structure of our system. User submissions (such as thank-you cards and GIFs) are stored using MariaDB. 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 describes our system architecture's specific components. We provide a static and dynamic class element diagram to demonstrate important classes and methods.

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.

Lastly, our appendices section includes some relevant supplementary materials.

System Architecture

Introduction/Rationale

Our web application runs on React/Node.js, Express.js, and MariaDB. The prototype that we based our work off of used the MERN tech stack (MongoDB, Express.js, React, Node.js) so we kept most of the technologies the same. However, we had to migrate from MongoDB to MariaDB due to compatibility issues with GT Plesk.

We retain the layered architecture found in the original application. This provides flexibility and modularity in our system. One very large benefit of this modularity can be seen in our migration of databases from MongoDB to MariaDB, which was made much easier due to this layered structure. We were able to keep most of the other layers with minimal changes and only had to refactor a single section of our code. The layered architecture also allows members of the team to work on different components simultaneously, allowing for more efficient multitasking and workflow. Lastly, isolating the database layer and API calls allows for the backend to add a layer of encapsulation which enhances security, such as preventing SQL attacks. This is important as we handle some sensitive personal information within the backend and database, such as information about TAs and their personal contact info.

Below, we provide two diagrams to demonstrate our system architecture. The first is a static system architecture diagram (Figure 1), which demonstrates all the main components of our systems and their relationships/interactions. The second is a dynamic system architecture diagram (Figure 2), which walks us through a common task that the system must perform. This gives the viewer insight into how the system components work together in a specific scenario.

Static System Architecture

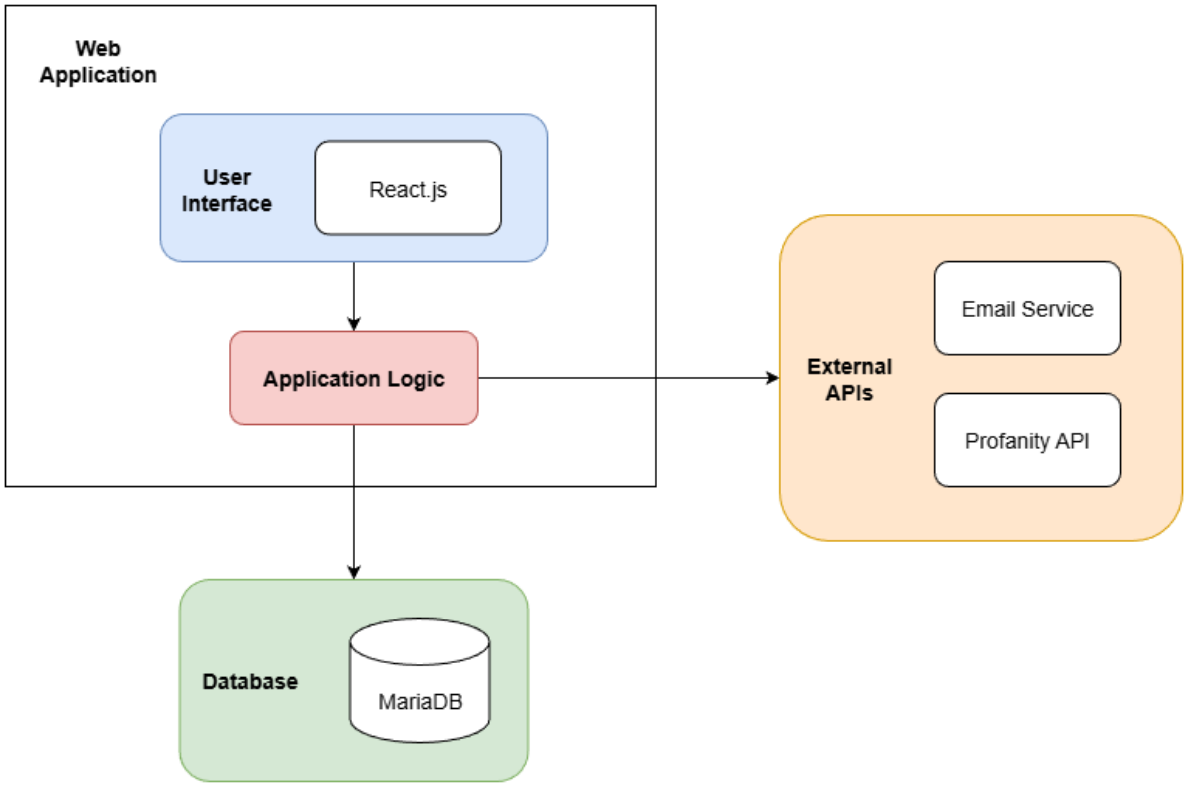

Figure 1 - Static System Diagram

Above is a Static UML diagram of our system architecture. Note that the arrows here represent dependency. For example, the User Interface displays information based on what it receives from the Application logic, which in turn receives information from the Database.

Our application consists of 4 main architectural layers: the User Interface, the Application Logic, the External APIs, and the Database. The User Interface and Application Logic together create what we call the Web Application, whereas the Database and External API layers are external inputs to the Web Application and are interacted with only through our Application Logic, or backend.

The User Interface is the only part of our web application shown to users. This layer includes all webpages, dashboards, and user interactable/viewable fields like text fields or alerts. All information about other parts of the system is shown through the User Interface. The Application

Logic is our backend system, and transfers information between the User Interface and all external inputs. Any comparison logic necessary for the application to function, such as checking for a correct password for login, happens in this layer. The Database keeps track of information necessary for the application, such as, but not limited to, student/TA profiles and created cards. Lastly, the External APIs layer consists of helpful APIs that aid us in implementing features we want in the application. The two APIs we use are an email service to help us with notifying TAs of card deliveries, and a profanity filter to prevent harmful content on the platform.

Dynamic System Architecture

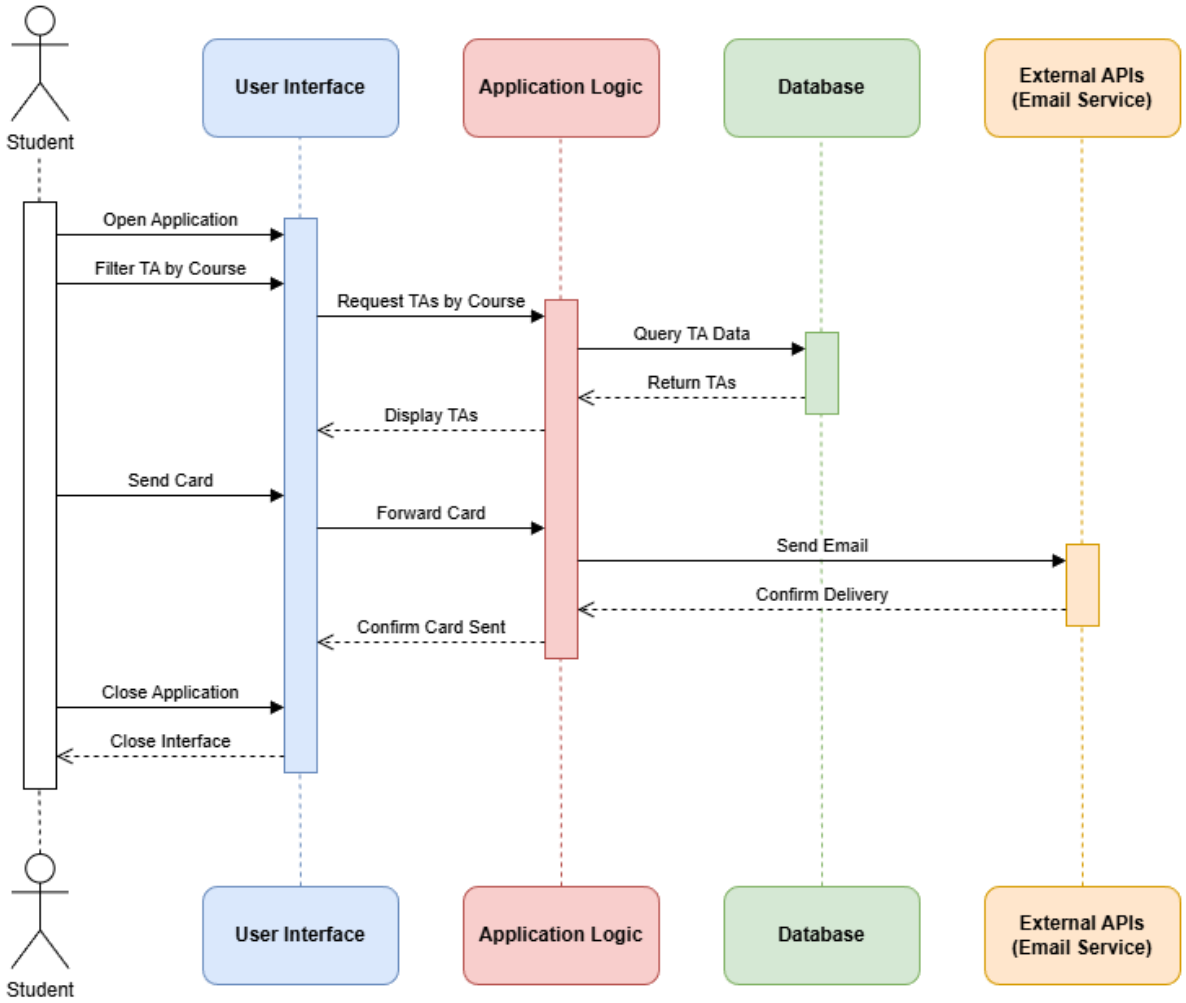

Figure 2 - Dynamic System Diagram

Figure 2 above is a system sequence diagram (SSD) that shows a simplified flow of control between the layers of our application as a common task is performed. The task we demonstrate is a student searching for and sending a card to a TA. Some steps, such as the use of the profanity filter, have been removed from this example for clarity.

The process begins when the student connects to our web application. The User Interface initializes very shortly after. Through the User Interface, the student selects a course to filter TAs by. The Application Logic receives this request from the User Interface and queries the Database, which returns a list of TAs for that course. The Application Logic then forwards this list to the User Interface, which renders the information in a presentable way to the student.

The student then creates a card for a TA on the User Interface, which sends the completed card to the Application Logic. The Application Logic adds the card to the correct TA's inbox and then sends a request to the External API layer, in this case the email service, to send a notification email to the TA. The email API confirms to the Application Logic that the email has been sent, which in turn confirms to the User Interface that the card has been sent. The student then receives a confirmation message on their interface, and if the student then closes the application, the process is complete.

Data Storage Design

Introduction

Our data storage technology choice is MariaDB because this is the only database compatible with GT Plesk Hosting. The figure below (Figure 3) is an Entity Relationship Diagram (ERD). Because the previous prototype used MongoDB, the backend code was designed to handle NoSQL key-value data. Thus, it was much easier to implement the database in MariaDB as 5 separate tables without foreign keys. We decided that the redundancies/inefficiencies caused by this were negligible in our use case. However, we set up the tables in such a way that foreign key relationships can be added in the future with relative ease.

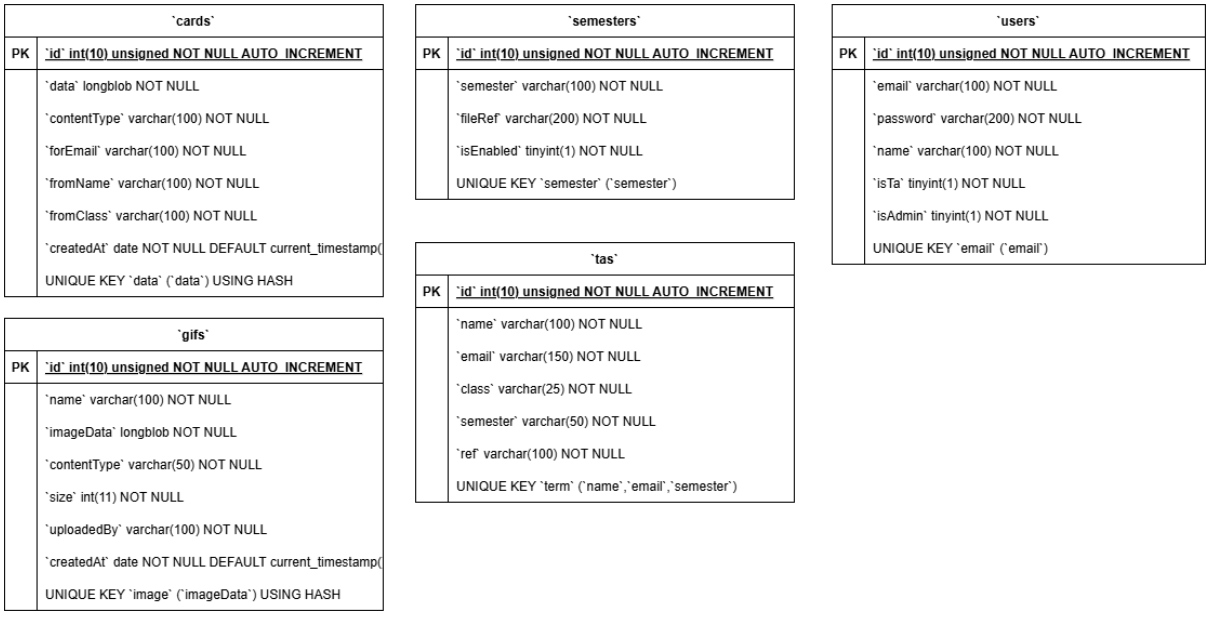

Figure 3: Data Design

Our database consists of 5 main tables.

1. Cards: Contains created cards, as well as metadata about the sender and recipient
2. GIFs: Contains all pre-uploaded images/GIFs that students can use to customize cards
3. Semesters: Contains information about semesters and TAs. Allows admin to easily modify semesters through their dashboard.
4. TAs: Contains information about TAs like email, class, and semester
5. Users: Contains user profiles and login information

File Usage

The MariaDB database contains all the information about the cards created. The contents of the card table contain HTML fragments containing information about the design and structure of the card within the ‘data’ column. The GIFs table also contains the PNGs of the blank cards that are used as templates by users, as well as the GIFs that the users can add to their cards.

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 private student information we store. To ensure user verification, we confirm that all users are Georgia Tech students by utilizing GT SSO. This ensures that only authorized users can access information. Using SSO also ensures password/user profile security.

Our entire web application runs on SSH, meaning computers will access it on HTTPS, handling most web security concerns. Additionally, React has many defensive tools built, such as those to counter XSS attacks.

All sensitive information in MariaDB is encrypted. Additionally, our database queries are sanitized and encapsulated with an execute message, protecting the web application from SQL injections.

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 encompasses everything within the application. There are various pages to be navigated between within 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.

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 MariaDB database for data storage and retrieval. External services, such as the email service, 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 Class Diagram

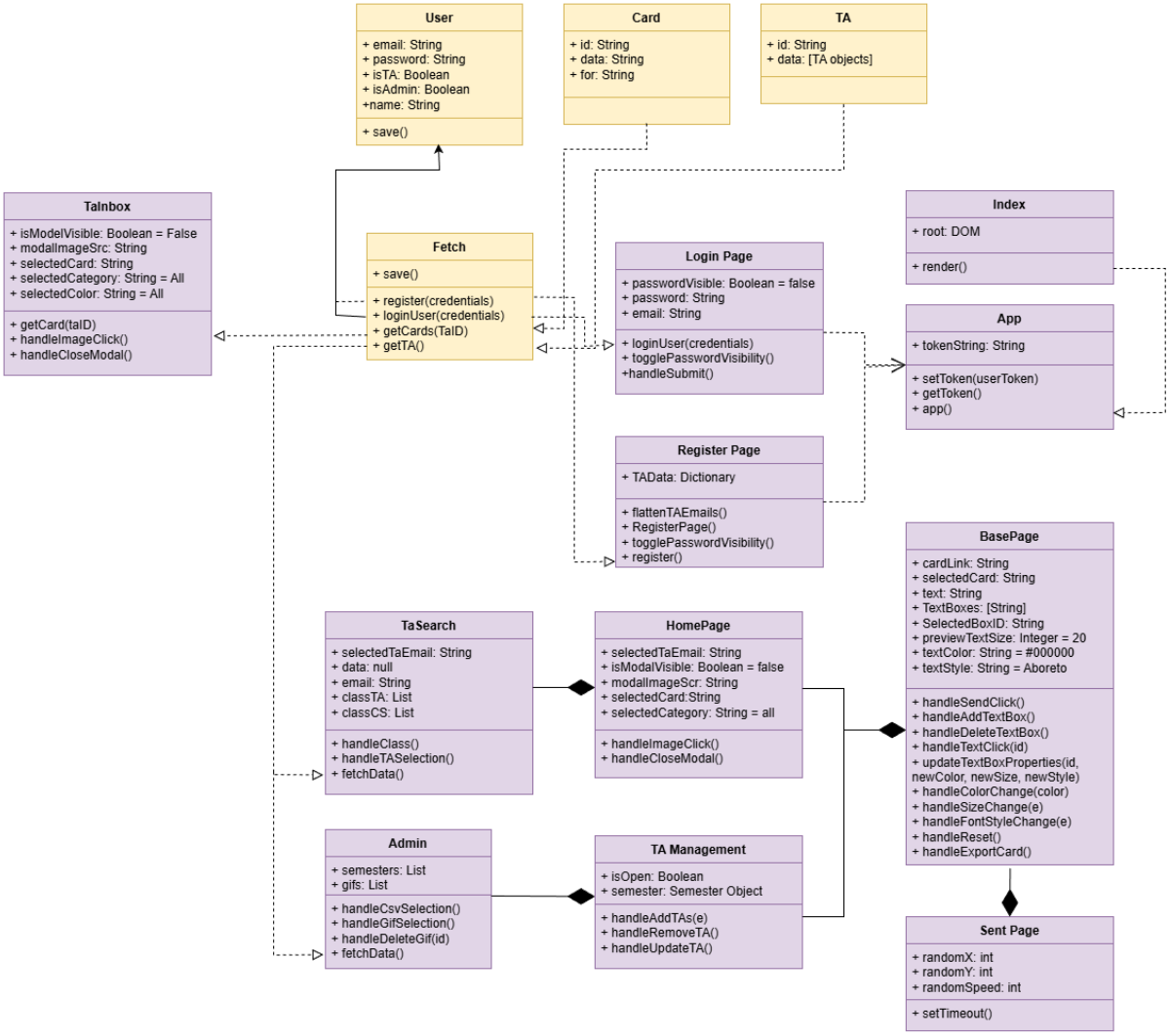

Figure 4. Static Class Diagram

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 Class Diagram

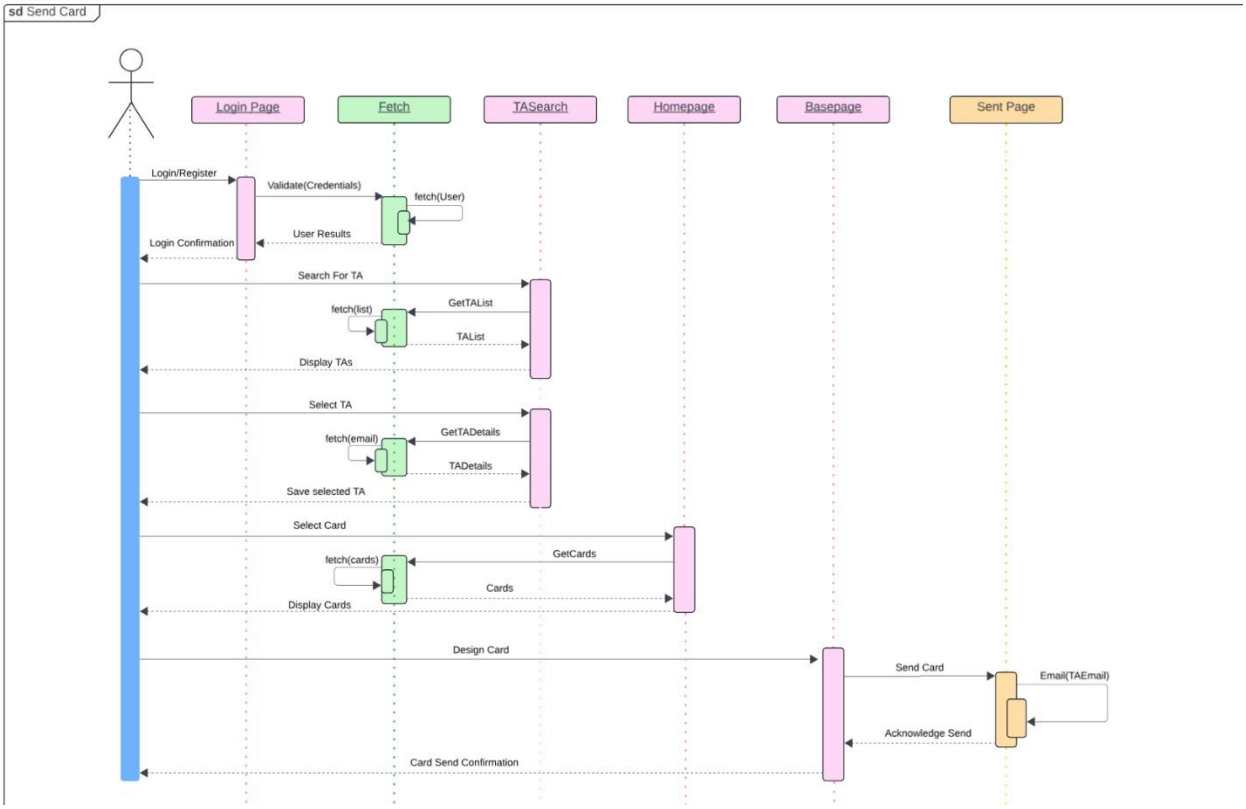

Figure 5. Dynamic SSD to show sending a card

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, TASeach, 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 the 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.

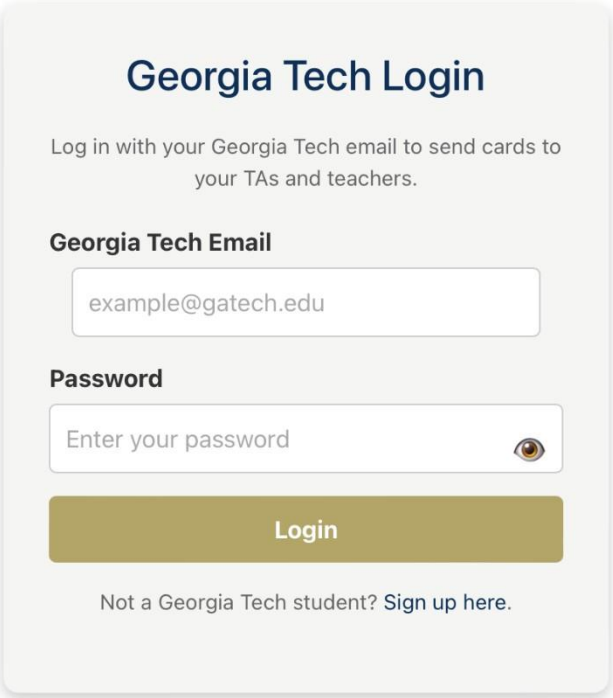

Figure 6: Georgia Tech Login Page

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**.

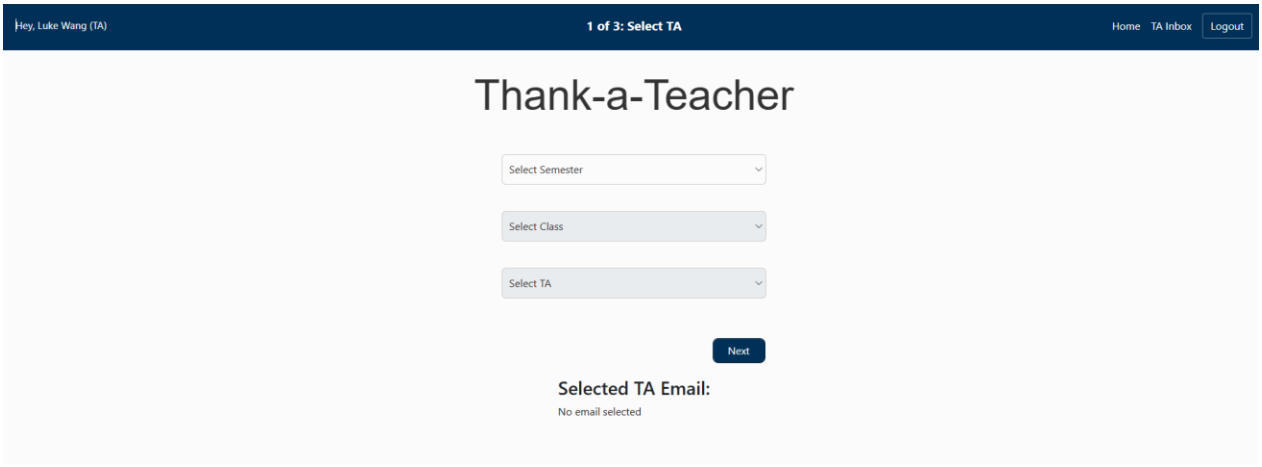

Figure 7. Search Page

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.

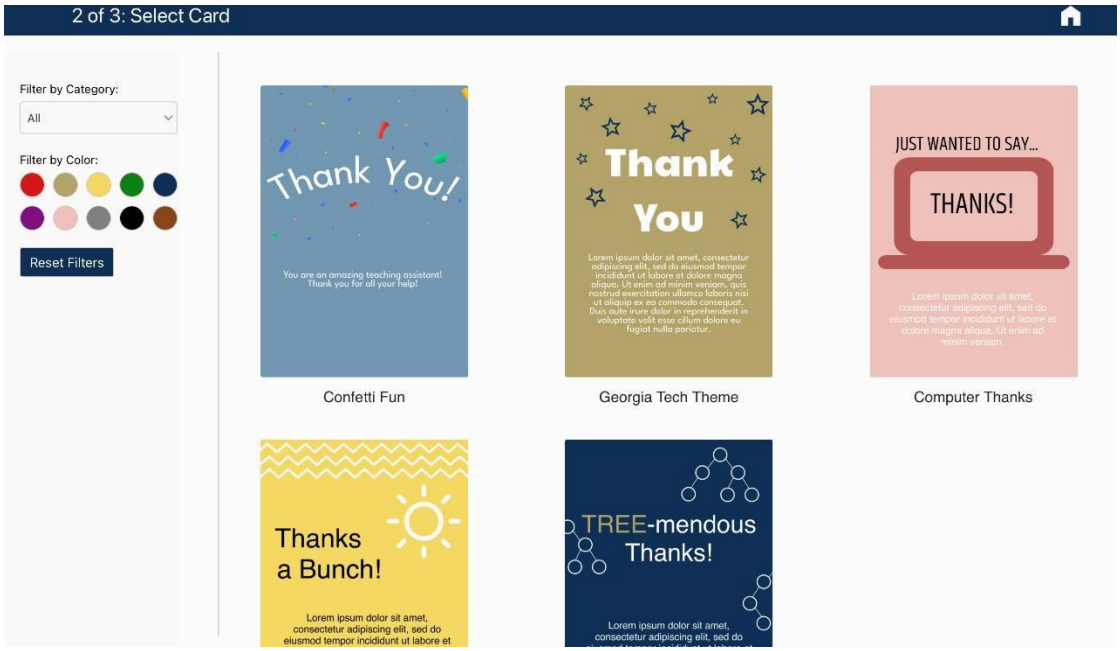

Figure 8. Home Page

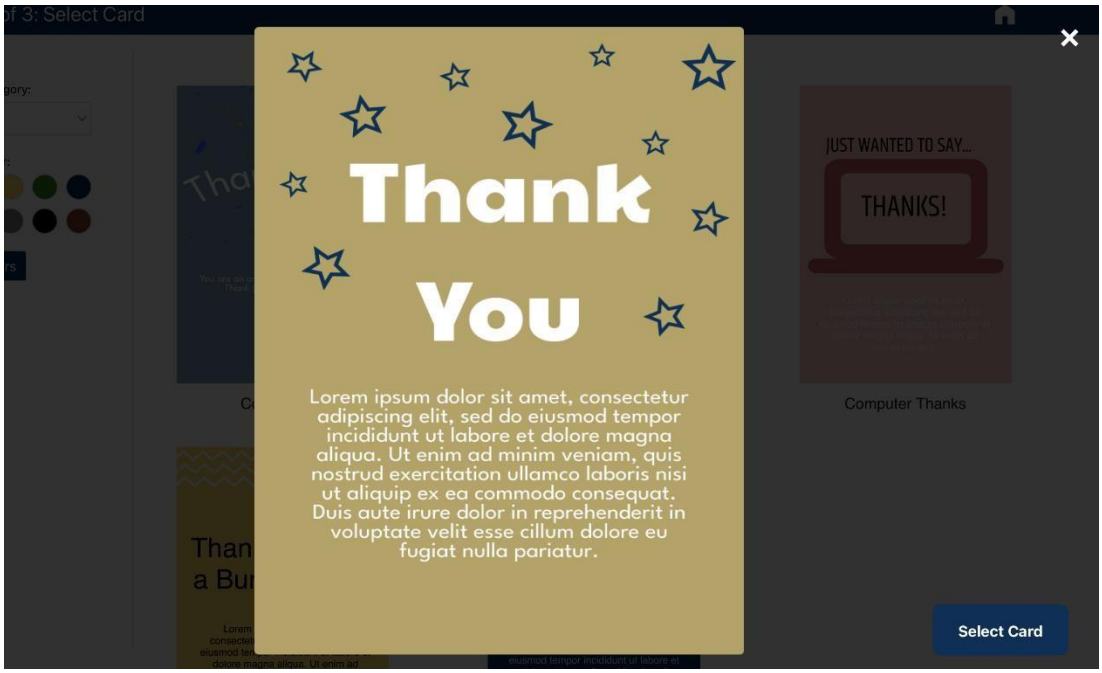

Figure 9. Card Pop-Up

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.

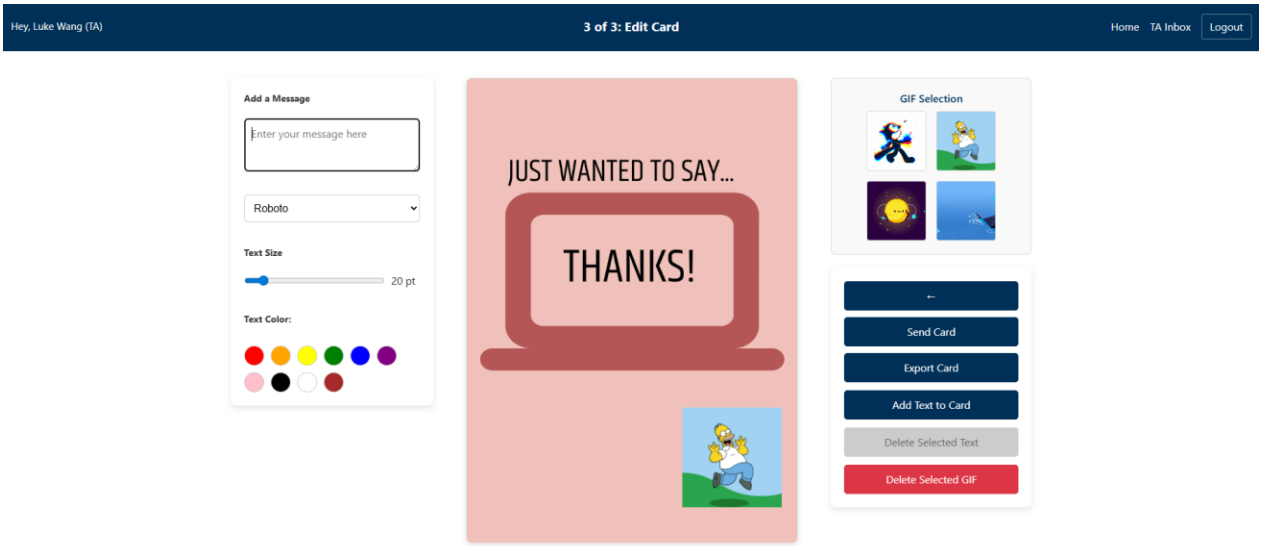

Figure 10. Card Design Page

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.

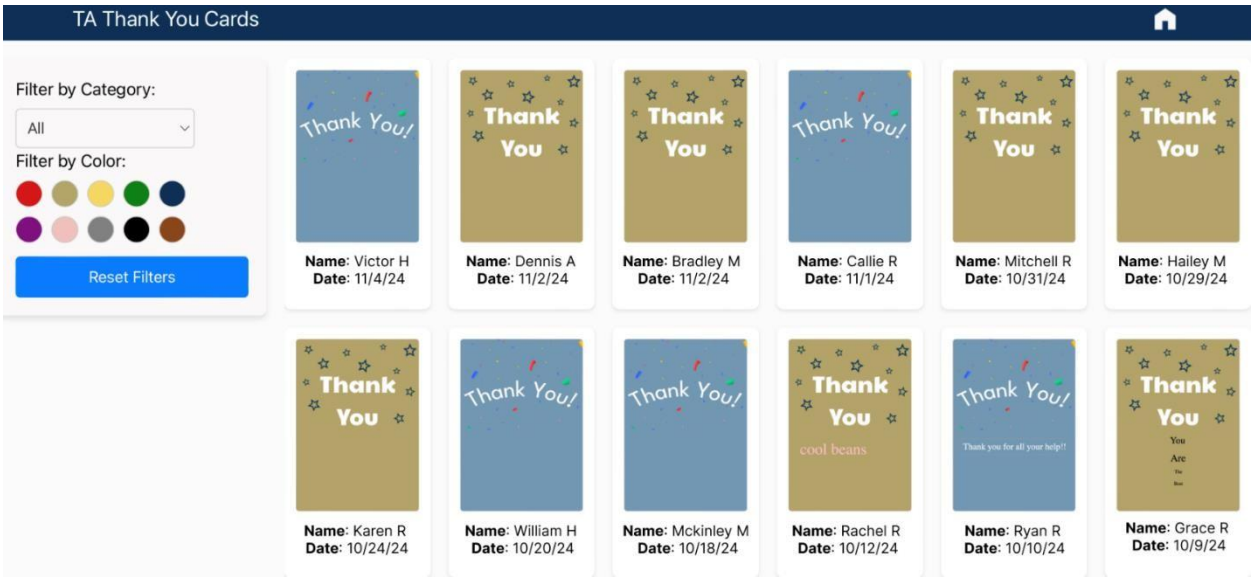

Figure 11. TA Inbox

Appendix

API Calls

The Thank-A-Teacher system utilizes a Node.js back-end that exposes an API. The API is used to fetch and store data to the database from the front-end. 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

Authentication Endpoints

/login (SSO)
Request Type: GET
This endpoint initiates the Georgia Tech CAS/SSO authentication flow. When accessed, it redirects the user to the Georgia Tech SSO login page. After successful authentication, the user is redirected back to the home page.
Response: Redirects to /

/logout
Request Type: GET
This endpoint logs the user out of the CAS/SSO session and clears their authentication.
Response: Handles CAS logout

/whoami
Request Type: GET
This endpoint returns the current authenticated user's information from the CAS session.
Response:

```
{
  "user": {
    "mail": "gburdell3@gatech.edu",
    "uid": "gburdell3",
    "displayName": "George P Burdell"
  }
}
```

Semester Management Endpoints

/semesters
Request Type: GET

This endpoint retrieves all semesters from the database, including both enabled and disabled semesters.
Response:

```
 [
  {
   "_id": "673125de0adb444d0e6f1cb8",
   "semester": "Fall 2024",
   "fileRef": "fall2024_tas.csv",
   "isEnabled": true
  },
  {
   "_id": "673125de0adb444d0e6f1cb9",
   "semester": "Spring 2024",
   "fileRef": "spring2024_tas.csv",
   "isEnabled": false
  }
 ]
```

/semesters/enabled
Request Type: GET
This endpoint retrieves only the enabled semesters. Students can only see and select TAs from enabled semesters.
Response:

```
 [
  {
   "_id": "673125de0adb444d0e6f1cb8",
   "semester": "Fall 2024",
   "fileRef": "fall2024_tas.csv",
   "isEnabled": true
  }
 ]
```

/semesters/:id
Request Type: DELETE
This endpoint deletes a semester and all associated TAs. The :id parameter is the ObjectId of the semester.
Parameters:
- id - Semester ID (e.g., "673125de0adb444d0e6f1cb8")

Response:
"Semester and associated TAs deleted successfully!"

/semesters/:id/toggle
Request Type: PATCH
This endpoint toggles the isEnabled status of a semester. If enabled, it becomes disabled, and vice versa.
Parameters:
- id - Semester ID (e.g., "673125de0adb444d0e6f1cb8")

Response:
```
 {
   "message": "Semester enabled successfully!",
   "semester": {
     "_id": "673125de0adb444d0e6f1cb8",
     "semester": "Fall 2024",
     "isEnabled": 1
   }
 }
```

TA Management Endpoints

/upload-tas
Request Type: POST
This endpoint uploads a CSV file containing TA information. The CSV must include columns: Semester, FirstName, LastName, Email, Class. It creates or updates a semester and replaces all TAs for that semester with the new data.
Request:
- Content-Type: multipart/form-data
- Field name: csv
- File type: .csv

CSV Format:
```
Semester,FirstName,LastName,Email,Class
Fall 2024,John,Doe,jdoe@gatech.edu,CS 1332
Fall 2024,Jane,Smith,jsmith@gatech.edu,CS 1331
```

Response:
```
"TA data uploaded successfully!"
```

/ta/id/:email
Request Type: GET
This endpoint retrieves the TA ID for a given email address.
Parameters:
- email - TA email address (e.g., "jdoe@gatech.edu")

Response:
```
 {
   "_id": "673125de0adb444d0e6f1cba"
 }
```

/tas/:semester
Request Type: GET
This endpoint retrieves all TAs for a specific semester.
Parameters:
- semester - Semester name (e.g., "Fall 2024")

Response:
```
 [
   {
```

```
    "_id": "673125de0adb444d0e6f1cba",
    "name": "John Doe",
    "email": "jdoe@gatech.edu",
    "class": "CS 1332",
    "semester": "Fall 2024",
    "ref": "fall2024_tas.csv"
  },
  {
    "_id": "673125de0adb444d0e6f1cbb",
    "name": "Jane Smith",
    "email": "jsmith@gatech.edu",
    "class": "CS 1331",
    "semester": "Fall 2024",
    "ref": "fall2024_tas.csv"
  }
]
```


/tas/:semester
Request Type: POST
This endpoint adds a single TA to a semester manually (personal add).
Parameters:
- semester - Semester name (e.g., "Fall 2024")
Request Body:


```
{
  "name": "John Doe",
  "email": "jdoe@gatech.edu",
  "class": "CS 1332"
}
```


Response:


```
{
  "_id": "673125de0adb444d0e6f1cba",
  "name": "John Doe",
  "email": "jdoe@gatech.edu",
  "class": "CS 1332",
  "semester": "Fall 2024",
  "ref": "personal add"
}
```


/tas/:id
Request Type: PUT
This endpoint updates a TA's information (name, email, or class).
Parameters:
- id - TA ID (e.g., "673125de0adb444d0e6f1cba")
Request Body:


```
{
  "name": "John Doe Updated",
  "email": "jdoe@gatech.edu",
```

```
  "class": "CS 1332"
 }
```

Response:

```
 {
  "_id": "673125de0adb444d0e6f1cba",
  "name": "John Doe Updated",
  "email": "jdoe@gatech.edu",
  "class": "CS 1332",
  "semester": "Fall 2024",
  "ref": "fall2024_tas.csv"
 }
```


/tas/:id
Request Type: DELETE
This endpoint deletes a specific TA from the database.
Parameters:
 - id - TA ID (e.g., "673125de0adb444d0e6f1cba")
Response:
 "TA deleted successfully!"

Card Management Endpoints

/card
Request Type: POST
This endpoint creates and sends a card to a TA. It performs profanity filtering on the text content, stores the card in the database as a base64-encoded image, and sends an email notification to the TA.
Request Body:

```
 {
  "data": "...",
  "forEmail": "jdoe@gatech.edu",
  "fromName": "Student Name",
  "fromClass": "CS 1332",
  "text_content": ["Thank you for being an amazing TA!", "You helped me so much!"]
 }
```

Response:

```
 {
  "ok": true
 }
```

Error Response (Profanity Detected):

```
 {
  "message": "Your message contains inappropriate language and could not be sent."
 }
```


/cards/:taId
Request Type: GET

This endpoint requests all cards associated with a specific TA. The taId is the email of the teaching assistant (TA) that the frontend wants to display in 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.
Parameters:
- taId - TA email address (e.g., "jdoe@gatech.edu")
Response:

```
[
  {
    "_id": "673125de0adb444d0e6f1cb8",
    "data": "...",
    "forEmail": "jdoe@gatech.edu",
    "fromName": "Student Name",
    "fromClass": "CS 1332",
    "contentType": "image/gif"
  }
]
```

GIF Management Endpoints

/upload-gif
Request Type: POST
This endpoint uploads a GIF file to the database for use in card creation. Only GIF files are allowed, with a maximum size of 5MB.
Request:
- Content-Type: multipart/form-data
- Field name: gif
- File type: image/gif
- Max size: 5MB
Response:
"GIF uploaded and saved successfully!"
Error Response:
"Only GIF files are allowed."

/get-gifs
Request Type: GET
This endpoint retrieves a list of all GIFs stored in the database. Returns only metadata (name and ID), not the actual GIF data.
Response:

```
[
  {
    "name": "celebration.gif",
    "_id": "673125de0adb444d0e6f1cbd"
  },
  {
    "name": "thank-you.gif",
```

```
    "_id": "673125de0adb444d0e6f1cbe"
  }
]
```


/get-gif/:id
Request Type: GET
This endpoint retrieves the actual GIF file data for a specific GIF ID. Returns the raw binary data with the appropriate content-type header.
Parameters:
- id - GIF ID (e.g., "673125de0adb444d0e6f1cbd")
Response:
- Content-Type: image/gif
- Body: Raw GIF binary data

/delete-gif/:id
Request Type: DELETE
This endpoint deletes a specific GIF from the database.
Parameters:
- id - GIF ID (e.g., "673125de0adb444d0e6f1cbd")
Response:


```
{
  "message": "GIF deleted successfully"
}
```


Error Response:


```
{
  "message": "GIF with ID 673125de0adb444d0e6f1cbd not found"
}
```


Health Check Endpoint

/health
Request Type: GET
This endpoint is a simple health check to verify the server is running.
Response:


```
{
  "ok": true
}
```


Previous Team's Work

The work of the previous team which this team's work was based on can be found here: https://github.com/CS-3311-Group-4115/Thank-A-Teacher