Emergency Class Manager

Katie Hoskins, Tanya Peacock, Marcus Stange

Introduction

The Emergency Class Manager is a website created in order to make the process of marking students' presence in class during emergency situations or drills in Elementary Schools. This website expedites the process of checking students in class. The website also allows for events to be created, a way to add or delete students, and class information if it is needed. In the Clarksville-Montgomery County School System, students are checked in class manually by teachers through a spreadsheet. This makes the process much more difficult for both teachers and administration, as there are much more spreadsheets submitted as well as much more for teachers to write in the spreadsheet, leading to a larger chance for errors in the data. The website is also accessible by a smartphone through a web browser which gives teachers another way to send their reports to administration if there was a situation where a teacher did not have their laptop. Overall, this website will increase productivity in these situations and give teachers and administration a centralized place to submit and get information.

Technology

For the website, there were four main technologies used. These consist of VSCode, Supabase, Vue3, and Vuetify. VSCode was chosen because of the ease of use of the IDE as well as the extensions that are able to be used with the IDE. This also allows for easy usage of git and GitHub. Creating and adding things to the GitHub repository are incredibly easy to implement with VSCode. This also made sure that all programmers on the team were using the same IDE so that if one had trouble, another could help them. All programmers on the team are also very familiar with VSCode so it seemed like the obvious choice of IDE. All team members knowing how to use VSCode and its extensions already made VSCode incredibly helpful to the project development. Supabase is the database that is used to support the Emergency Class Manager. Supabase is a Postgres database that is a Firebase alternative. Supabase uses its own authentication service and its own API in order to make the database work. Originally the website was going to use Firebase as the database, but after research on Firebase compared to what the website needed to do, the team decided that Supabase would be a better option. Supabase was the right decision to make because as the team continued on the development process, there were things such as authentication that they wanted to use that worked well with Supabase, and extensive documentation and community that was willing to help when needed. Since everyone in the team worked with databases in the past, this was not that difficult to get the hang of. Supabase is very clear in viewing and creating tables and how to use them which made it very easy to learn and figure out. Even if something was difficult to understand, Supabase has a large community who could help when needed. All of this together made Supabase extremely helpful to the project and development. The Emergency Class Manager also uses Vue3 and Vuetify. Vue3 is a JavaScript framework and the main framework that the team used to build the website in frontend development. All of the components of the website were built from Vue3. Vuetify is a JavaScript framework built on top of Vue3 that allows the website to have more components and features than it would with just Vue3. Vue3 and Vuetify were the right choices when it comes to the user side of the website as the team saw while making the website. The components are easily laid out and easy to use. Because of this, it was relatively easy for the team to learn as the websites have an extensive amount of information and documentation available. When the team uses one component, it is easy to reuse it later with components and imports which makes it much easier to learn as well. At first the team talked about using React instead of Vue3, but after more research they decided that Vue3 and Vuetify would be the best option for the frontend. Together, these frameworks of development helped the team more than any other technology. It put the website together and made it incredibly easy to do what was needed.

Design

For the design of the website, Vuetify and Vue3 do not use classes as normal. For the project, we used .vue files. The files consist of a <template>, a <script>, and certain files contain a <script setup>. The template makes up the user side of the website. This is where the team created what is seen on the user side, or in the frontend. The script and script setup both are more of the backend side of the website and what is used for the template. The website also has separate folders for assets, components, layouts, router, and views that are all contained in the src folder. The assets folder contains pictures that the website uses, for example, the logo for the Emergency Class Manager is contained in the assets folder so that it can be used when needed. In this folder, the image is saved as a png and can be imported where needed. In the components folder, these are files that can be imported and used wherever needed similar to the assets folder. In this folder, there are four files that were created. These files are Confirmation.vue, EditClasses.vue, EditTeachers.vue, and snackbar.vue. Confirmation.vue is a file that the team used for confirmation on delete. For EditClasses.vue, this file is imported into the Classes.vue file. This file is used to edit specific classes in the Classes.vue file. It contains a retrieve function that has different functions inside of it such as count, fetch, and search. The count function gets all of the data that is in the Perm Roaster table and fetch does the same thing, except it creates an itemsPerPage variable for the table. Finally, the search function allows a user to search for students when adding them to a class. This file also contains nine methods: loadRows, searchRows, studentSearch, saveChanges, createClass, addStudent, and removeRow.. The loadRows method does exactly what it says; it loads the rows from the count and fetch function and puts them into the table for the user to see. The searchRows method implements the search function to search for classes in the loadRows method. The studentSearch function allows the user to search for students when adding them to a class and takes advantage of the searchStudents function and the search function to allow the user a fully functional search method. The saveChanges method checks for changes that were made to the classes and allows the user to save the changes to the database when the button gets clicked. The createClass method creates a class and saves it into the database table. The addStudent method allows the user to add a student to a class and add them to the database. Finally, removeRow completely removes a class from the table. Some of these methods are used in multiple files as well. The

EditTeachers.vue file is used to edit specific teachers in the Classes.vue file as well. This file has some of the same functions as the EditClasses.vue file does but also has an extra searchUsers function that calls a supabase function to get the users from the database table. This file uses many of the same methods as the EditClasses.vue including loadRows, searchRows, saveChanges, and removeRow. There are also some different methods though such as fetchUserData, userSearch, addTeacher, and setPrimary. One of the most important methods that is used everywhere in the project is fetchUserData. This method pulls the information from the signed in user from the database to save and use for later. For most files, this method is needed, as each person has their own information in the database. The userSearch method allows the user that is assigning teachers to classes to search for specific users to add. It uses the searchUsers and search functions to do so. The addTeacher method does the same thing that the addStudent method does in the EditClasses.vue file, but allows the user to add a teacher instead of a student. This method inserts data into the Teacher Classes database table. Finally, the setPrimary method allows a newly added teacher to be labeled as the primary teacher. There is a confirm module that opens when a teacher is added to the primary teacher and finally updates the data in the Teacher Classes table. Lastly in the components folder is the snackbar.vue file. This file is one of the more important files that is used in this project. The snackbar is used in almost every file. While it may not have been imported on every file, it was used in almost all of them. The snackbar component is used to confirm or give an error to a user if something they did was not right or if something could not be submitted to the database. This component is used almost everywhere in this project as an alert. Some users may turn off alerts on their web browser, and the snackbar component gives an alert that the user is unable to disable.

The layouts folder of the project contains one simple file called Default.vue. This page is incredibly important as it contains what is on every page. On every page of the website there is an appbar and a dark mode switch. In the app bar for the Default.vue file before a user has signed in, there is the homepage, sign in page, and contact page. Once a user has signed in, they have access to more pages including dashboard, roster, account, teacher events, and sign out. If the user is marked as an admin in the database table, they have more access to pages like users, events, classes, student manager, and reports. This is shown on every page as a pop out menu so that it can be used anywhere to change pages. Included in the Default.vue file is a dark theme toggle. When the toggle is clicked, it changes the color of the pages and it follows to each page so that the theme stays the same on each page that the user changes to. The last item on this page is where the team uses the assets folder to get the website logo and put it on the top of the page on the app bar. When the user clicks the image, it reroutes the user to the dashboard page. There is only one method in the Default.vue file, signOut. This method simply allows the user to be signed out. The router folder contains the paths of the website and assigns them to names so that they can be used. The Default.vue file uses the router folder to get the names and assign the menu buttons to router definitions. Each page in the views folder has a matching router path. The paths include: home, signin, contact, users, dashboard, roster, account, events, teachevent, classes, teachContact, and studentManager. There is also another file outside of the folders called supabase.js. This file is imported into every file that has a call to the database. It consists of the supabaseUrl, the supabaseAnonKey, and it creates a client via both of those. This allows the team to set up supabase API and calls to the database.

The views folder is the largest and most important part of the project. The views folder contains all of the code for the website. While the other folders and files are important, these files contain almost all of the information needed to run the website. There are two different folders inside the main views folder. There is a Base Page folder and an In Pages folder. The Base Page folder contains three different files called Contact.vue, Home.vue, and SignIn.vue. The Contact.vue and Home.vue files are relatively simple files. On both, there is only a <template> that was made for the files. The Contact.vue file contains information for the team. It provides the user the team members names and emails in case they need to be contacted. The Home.vue file shows information about the Emergency Class Manager on the main page. Both of the pages contain an app bar for pages such as the sign in page, homepage, and contact page. The SignIn.vue file is what allows users to sign into the website. Since the data must be put into the database, supabase is imported here. There is a function for handleLogin which uses Supabase to allow users to sign in. This sign in uses a magic link so that when users put in their email to sign in, they are then sent a link. They use this link to open the newly signed in page and allow them to see the new pages. The file uses one method called insertData. This data calls the Supabase table 'Users' and inserts the email address into the database. If there is a problem with inserting the data, the snackbar is used to inform the user that there was an issue inserting their data into the database.

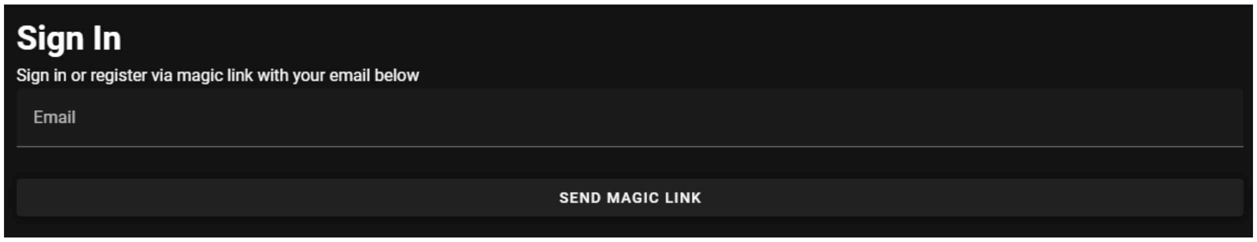
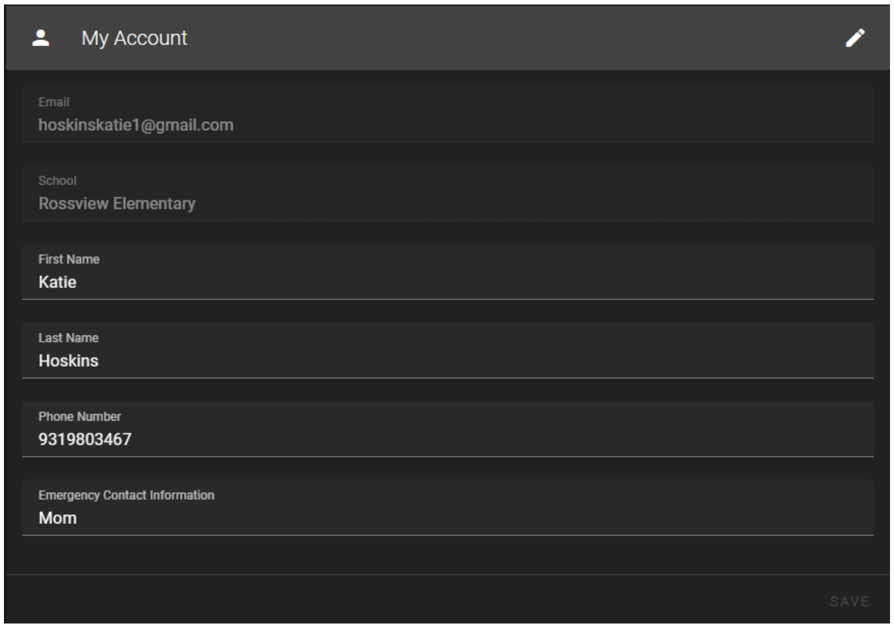


Figure 1: Sign In page of Emergency Class Manager.

The In Pages folder contains all of the files that are used once a user is signed in. This folder consists of nine files: Account.vue, AddStudents.vue, Classes.vue, Dashboard.vue, Event.vue, Roster.vue, TeachContact.vue, TeacherEvents.vue, and Users.vue. These pages completely make up the inner workings of the website. The Account.vue is the file correlated with the 'Account' page. This file shows all account information in the 'Users' table for that specific signed in user.


My Account

Email
hoskinskatie1@gmail.com

School
Rossview Elementary

First Name
Katie

Last Name
Hoskins

Phone Number
9319803467

Emergency Contact Information
Mom

Figure 2: Account page of Emergency Class Manager.

This page allows users to view and input their personal information. The information automatically gets called from the database to the account page. They are able to view their email address and assigned school, but are not able to edit that information. They also see their first name, last name, phone number, and emergency contact information. The user must have a first and lame name, but they do not have to input a phone number or emergency contact if they do not want to. The first name must be at least a length of at least two, and the last name must have a length of at least one. All of this information is able to be changed and saved in the database when the user clicks the pencil icon in the top of the card. When the user hits the save button, their information is then updated in the database. If the user edits the page but does not save and tries to switch to a different page, an alert will show warning the user that their changes have not been saved and they can continue, or go back to editing their information. There are four different methods in the Accounts.vue file: fetchUserData, toggleEditing, handleInput, and saveChanges.The fetchUserData method calls the information from the ‘Users’ table of the database and inputs the information into the text fields on the frontend. This method must be called before rendering the page as the information would not show properly otherwise. The toggleEditing method simply checks that if the pencil icon is clicked, editing goes from false to true so that the user can input their information. The handleInput method is called whenever the user needs to change an item in the text field for their account. This is called every time the user inputs changed information. Finally, the saveChanges method grabs the information that the user has recently changed, and updates the information for the user in the ‘Users’ table. Once the user clicks ‘Save’ at the bottom of the card, the form is then marked that the information has been saved.

The AddStudents.vue file is the file correlated with the ‘Student Manager’ page and is only accessible to those labeled admin. It shows a tabbed card where an administrator can add or delete students from the database. Both add and delete student have the same input boxes with

‘Student ID’, ‘First Name’, and ‘Last Name’.

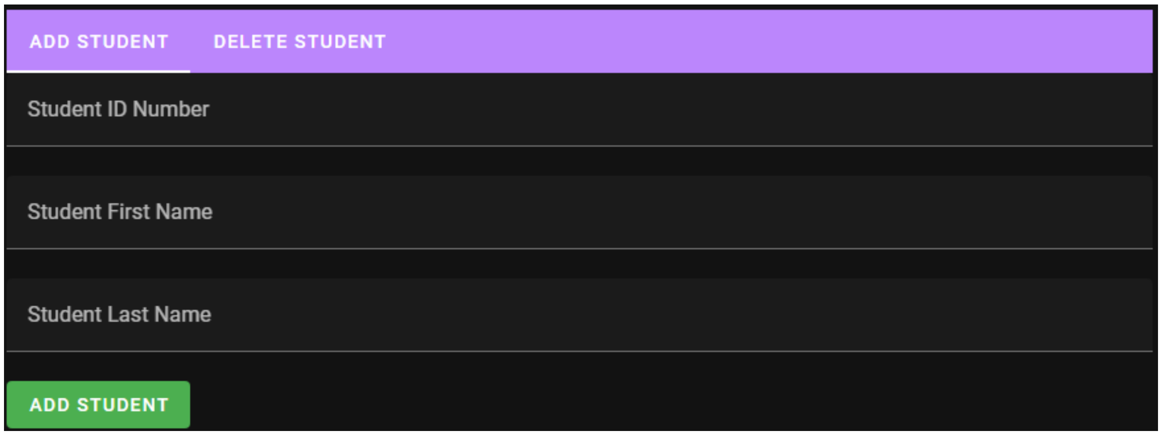
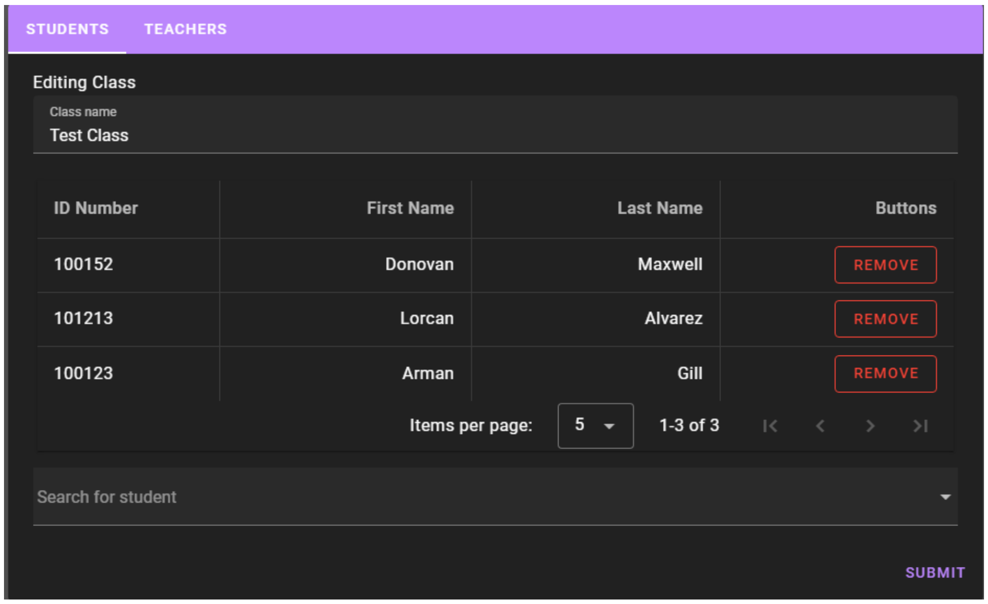


Figure 3: Student Manager page of Emergency Class Manager.

There are five methods in the AddStudents.vue file: addStudent, deleteStudent, fetchUserData, updateData, and insertData. The addStudent method first checks that all information (id_number, fName, and lName) have all been filled out. After that the method does a check that the student does not exist. If the student does not exist in the database, the data gets inserted into the ‘Students’ table using the insertData method. When the student has been inserted, the school_id that is assigned to the student is automatically set to the user’s id who added the student. If the student already exists in the database, the user is shown the snackbar saying that the student already exists in the database and does not get added. The deleteStudent method does a similar thing as it checks if a student exists or not. If the student exists, that ‘Students’ table is updated using the updateData method. The removed column is set to ‘true’ and the student is removed. The fetchUserData method does the same as all the other fetchUserData methods do. It calls the ‘Users’ table from the database and gets all user information for the user who is signed in. The updateData method calls from the ‘Users’ table and updates the data in the table when adding a new student. Finally, the insertData method calls from the ‘Students’ table and inserts data into the table when adding a new student.

The Classes.vue file is the file correlated with the ‘Classes’ page and is only accessible to administration. This file shows a table with the Classes that have been created and ‘Edit’ buttons on each entry in the table. There is also an ‘Add Class’ button at the bottom where the user is able to add new classes. When adding a new class, the user must input the name of the class and then can add students and teachers to the classes. When the class is added, or the ‘Edit’ button is pressed, a modal is opened for the user. This file uses the EditClasses.vue and EditTeachers.vue components that were mentioned previously. The user is able to search for students and add them or delete already added students, and edit the class name if needed. They are also able to add and remove teachers from classes if needed.



Editing Class

Class name
Test Class

ID Number	First Name	Last Name	Buttons
100152	Donovan	Maxwell	REMOVE
101213	Lorcan	Alvarez	REMOVE
100123	Arman	Gill	REMOVE

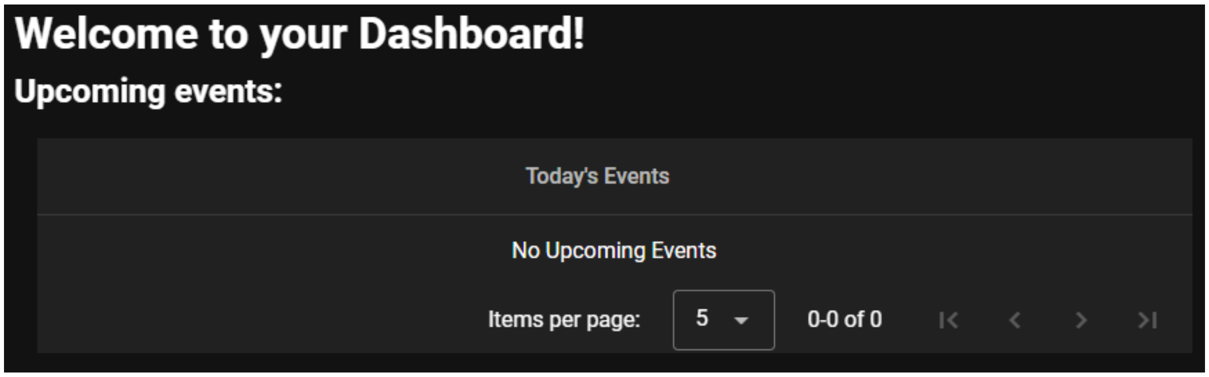
Items per page: 5   1-3 of 3

Search for student

Figure 4: Classes page to add students and edit classes.

Since this file must connect to supabase, it is imported in this file. There are three functions in this file: count, fetch, and search. These functions have been used previously and are used for all table functions in order to order the tables and get the correct number of inputs into them but with different tables. The functions in this file are used on the ‘Classes’ table in Supabase. There are five methods in this file as well: loadRows, searchRows, editRow, addRow, and refresh. The loadRow method gets the total number of rows using the ‘count’ and ‘fetch’ functions to load the rows from the ‘Classes’ table in the database. The searchRows method uses the ‘search’ function in order to search for classes. The editRows method takes the data that the user inputs when creating classes and updates it in the ‘Classes’ table. The addRow method is used whenever a new class is being added. It does a similar thing to editRows, but inserts the class into the database instead of updating it. Finally, the refresh method calls the loadRows method to update the most recent page to show the newly added class.

The Dashboard.vue file corresponds to the ‘Home’ page as well as the first page that shows when the user logs in, and the page that is loaded when the user clicks the logo on the app bar. On this page, the user is greeted with ‘Welcome to the Dashboard!’. If the user has a name loaded in the database, it will display ‘Welcome to the Dashboard, {firstName}!’. This page also shows a table for events that have been assigned to occur on that day. If no events have been scheduled for that day, it will show ‘No Upcoming Events’ on the table. If there are events happening that day, the event name will show on the table. The user can then click the event name on the table and get more information about the event from a modal including the name, date and time, and the event description.


Welcome to your Dashboard!

Upcoming events:

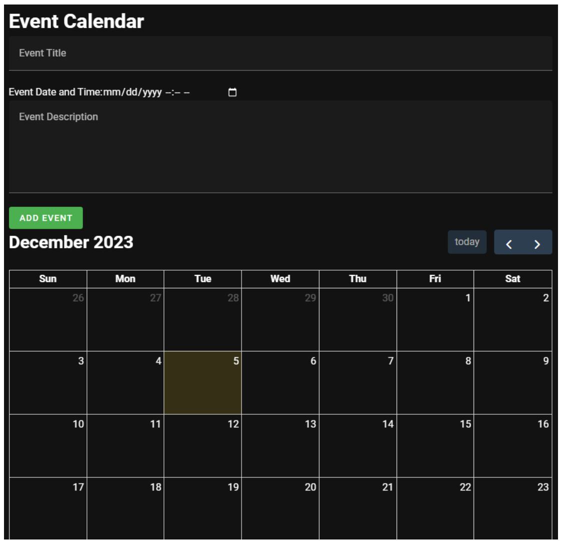
Today's Events
No Upcoming Events

Items per page: 5 0-0 of 0

Figure 5: Dashboard page of Emergency Class Manager.

Since the dashboard page connects to Supabase, it imports supabase into the file. In the Dashboard.vue file there are five different functions being used: fetchUserData, formatDate, count, and fetch. The 'fetchUserData' function gets all information from the signed in user. The 'formatDate' function simply formats the date so that it is more user friendly and readable. The 'count' function makes sure that the only events that are grabbed from the database are the ones that are occurring on that day. It pulls this information from the 'Events' table. Finally, the 'fetch' function does the same thing as the count function. There are also three methods used in this file: loadRows, showInfo, and showModal. The 'loadRows' method calls the count function and then checks the user's school_id. It then uses fetch to pull that information from the database and displays it on the table. The showInfo method just shows event information whenever the event is clicked on the table. The showModal method simply makes the modal close.

The Events.vue file is one of the larger files in the project. It correlates to the 'Event Manager' page and is only accessible by administration. This page includes a text field for 'Event Title', an input area for 'Event Date and Time', and a text box for the 'Event Description'. There is also an 'Add Event' button. The user is able to create an event and once the event is created and saved to the database, it is added to the calendar at the correct time and date. The user is also able to click on the event on the calendar and a modal will pop out with all information about the event like 'Event Name', 'Event Date and Time', and 'Event Description'.


Event Calendar

Event Title

Event Date and Time:mm/dd/yyyy --:-- --

Event Description

ADD EVENT

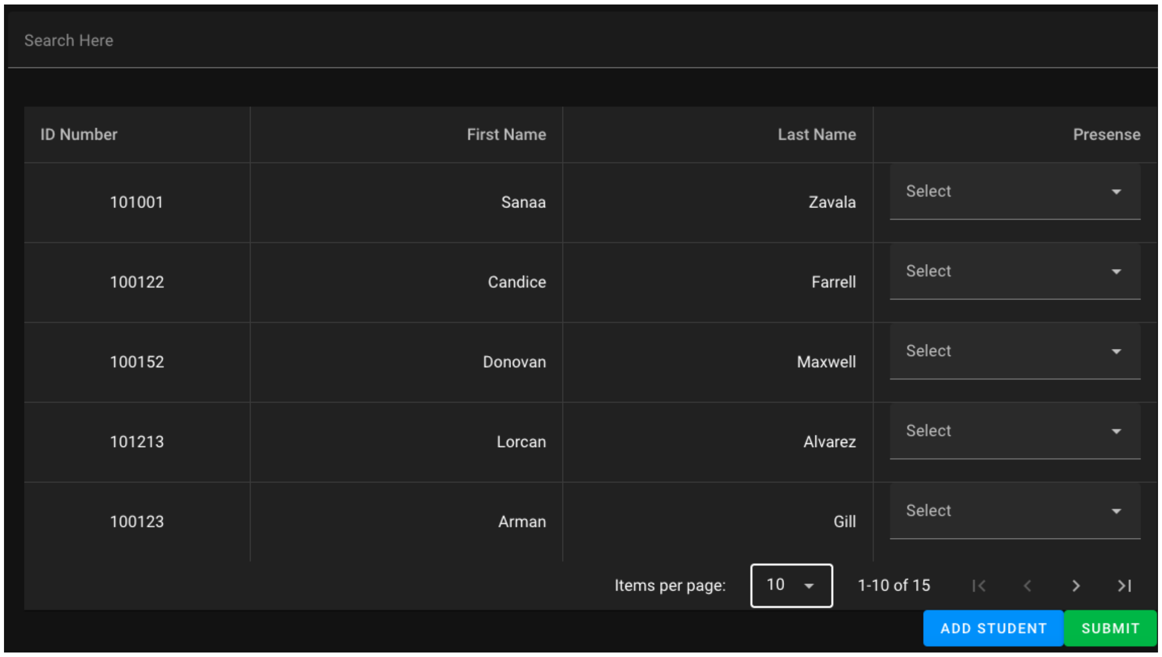
December 2023

Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23

Figure 6: Events page of Emergency Class Manager.

This file uses a third party library called 'FullCalendar' and 'dayGridPlugin'. This allows the frontend side of the website to show a calendar for the user as well as using its own API to add events to the calendar. Because of this, the file imports FullCalendar from GitHub as well as dayGirlPlugin from GitHub. It also imports supabase as the file uses information from the database. There are five methods used in this file: handleEventClick, addEvent, fetchUserData, insertData, and deleteEvent. The handleEventClick method is the method that is used whenever the user clicks on an event in the calendar and a modal appears with event information. Whenever this method is called, the modal appears and gets automatically filled with the information from the database. The addEvent method checks the user's school_id and assigns the event to the user's school_id automatically. If the user is not assigned to a school, they are unable to make events as they do not have a school_id. Then, a new event is created with a title and a start date that the user inputs into the text fields. The event name and event date and time are required if the user is creating an event, but the description is not, so the method checks that the event name and date/time are filled in. It then uses the API from the FullCalendar implementation to add the event to the calendar itself on the frontend. Then, the insertData method is called here to insert all the event data into the database in the 'Events' table. The fetchUserData is the same as the rest of the fetchUserData methods in previous files where it pulls information from the 'Users' table and gets all information for the signed in user. The insertData method pulls from the 'Events' table and adds the information, event name, event date and time, and event description, to the database table. Finally, the deleteEvent method checks when the event is scheduled to occur. If the event has already occurred, the event is not able to be deleted and the snackbar shows that the event is unable to be deleted. If the event has not already occurred, the event_id is checked and that specific event_id and all correlated information gets deleted from the table. The calendar API is then called and the event_id is checked. The event then gets removed from the calendar on the frontend.

The Roster.vue file correlates to the 'Roster' page of the website and is accessible by all users. The Roster.vue file outputs a table of students that are in the user's assigned class. The table shows the student's id_number, first name, last name, and a dropdown for student presence. In the dropdown, there are four selections: Present, Absent, Missing, and Visiting. The user will assign students based on these four selections and they are able to add a student temporarily to their list using the 'Add Student' button. This student's presence status is automatically set to 'Visiting' if the student is added. The information from this table then gets added to the 'Event Roaster' table for administration to view on the 'Reports' page using the 'Submit' button.


Search Here

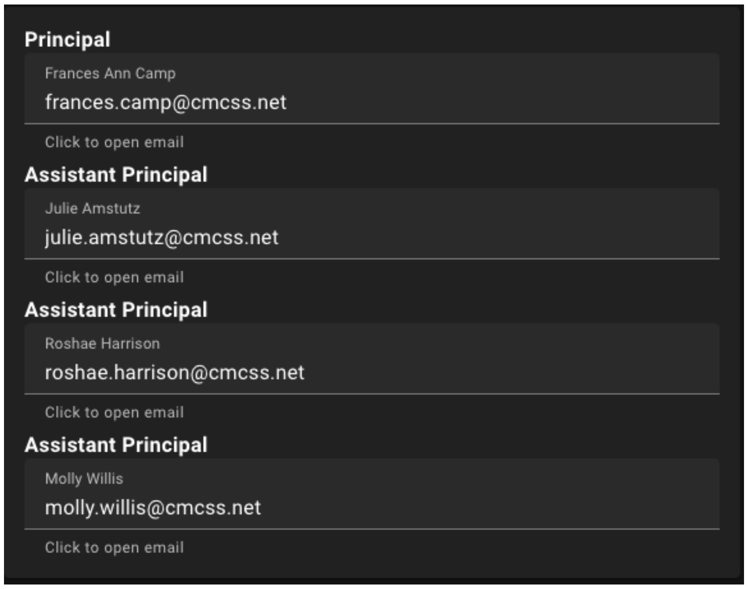
ID Number	First Name	Last Name	Presense
101001	Sanaa	Zavala	Select
100122	Candice	Farrell	Select
100152	Donovan	Maxwell	Select
101213	Lorcan	Alvarez	Select
100123	Arman	Gill	Select

Items per page: 10 1-10 of 15

Figure 7: Roster page of Emergency Class Manager

Since the database is used in this file, supabase must be imported. The file also uses the VDataTableServer import to set up the table in the frontend. In the Roster.vue file, there are four functions being used: fetchStudentIds, count, fetch, and search. The 'fetchStudentIds' function calls the 'Students' table and only gets the id_number from the table. The 'count' and 'fetch' functions both pull information from the 'Students' table to output on the frontend side of the website. The 'fetch' function also calls from the 'Teacher Classes' table in order to match students to their classes. The 'search' function allows the user to search for students in the class if needed. There are also four methods in the Roster.vue file: fetchUserData, addStudent, loadRows, and searchRows. The 'fetchUserData' calls the table and pulls all information for the signed in user. The addStudent method is used when adding a new student temporarily to the table. The user adds information to the modal that appears when the 'Add Student' button is clicked, and on addition, the updateStudentInfo is called to save the student information to the table, but not to the database. The 'loadRows' method is used whenever the page is loaded, in order to print all called information to the table in the frontend. This method uses the 'count' and 'fetch' functions in order to call the information from the 'Students' table and display it on the table. The 'searchRows' method calls the 'search' function in order to use the search bar to search for students when needed.

The TeachContact.vue file correlates to the 'Contact Info' page of the website and any user is able to view this page. This page consists of all of the names and emails of administration at the user's assigned school. Depending on how many principals and assistant principals the school has, only that many names and emails will show on the page using v-if statements. Each email on the page is able to be clicked and opens a new page based on what email the user is using. It will autofill the email address to the administrator that the user has clicked to make email access very easy.


Principal

Frances Ann Camp
frances.camp@cmcss.net
Click to open email

Assistant Principal

Julie Amstutz
julie.amstutz@cmcss.net
Click to open email

Assistant Principal

Roshae Harrison
roshae.harrison@cmcss.net
Click to open email

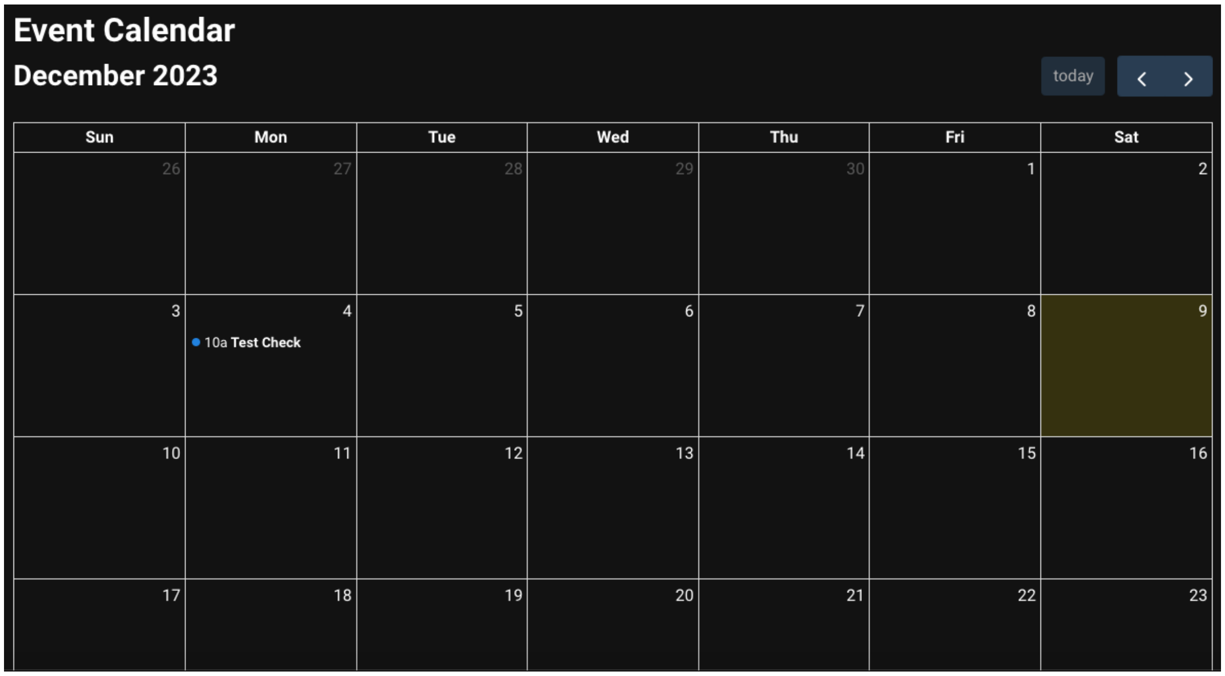
Assistant Principal

Molly Willis
molly.willis@cmcss.net
Click to open email

Figure 8: Contact Info page of Emergency Class Manager

Since Supabase is used to get the information from the ‘Schools’ table, it is imported into this file. There are only two methods in this file: openEmail, and fetchUserData. The ‘openEmail’ method is used each time the ‘Click to open email’ hint is clicked and it simply opens the email in a new tab. The ‘fetchUserData’ method calls the data from the ‘Users’ table and matches the ‘school_id’ field to the ‘school_id’ field in the ‘Schools’ table to output the respective emails.

The TeacherEvents.vue file correlates to the ‘Events’ page that teachers are able to view. This file does the same thing as the Events.vue table except the user is not able to edit the ‘Events’ table in the database. There is no place for the user to put in information to add to the table or the calendar.

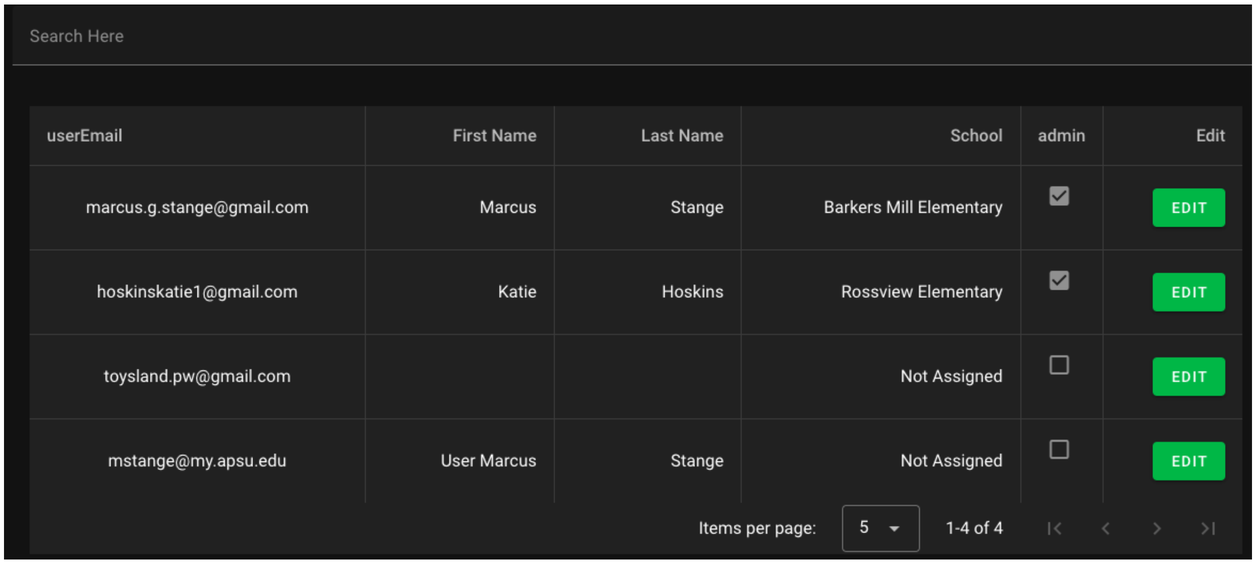


Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	1	2
3	4 ● 10a Test Check	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23

Figure 9: Teacher Events page of Emergency Class Manager

This file imports the same third party libraries as the Events.vue file, which is FullCalendar and dayGridPlugin. It also imports Supabase as it is pulling information from the 'Events' table. There are two methods in this file: handleEventClick, and fetchUserData.The 'handleEventClick' method is what allows the user to click on the item in the calendar and view the information about the specific event. When the event is clicked, information like the event name, date and time, and description show in the modal. The 'fetchUserData' method gets the information from the signed in user in the 'Users' table and compares the 'school_id' in the 'Users' table to the 'school_id' in the 'Events' table to only output events that are happening at the user's assigned school.

The Users.vue file correlates to the 'Users' page of the website. This page shows a table of users with the information from the 'Users' table in the database. It shows the userEmail, First Name, Last Name, School, admin status, and an edit button for each user. At the moment, this page shows all users since the admin should be able to change schools for a user if needed, but in the future may get updated. On click of the 'edit' button a modal will appear with user information. The userEmail, First Name, Last Name, School, and admin status will be on the page. The admin can update any of this information if needed and save it to the database for later use. The email, first name, and last name must be filled in. Email and first name must have at least four letters, and the last name must have at least two letters. The school can be changed and is changed through an autocomplete where it searches through the schools and allows the user to pick one.


Search Here

userEmail	First Name	Last Name	School	admin	Edit
marcus.g.stange@gmail.com	Marcus	Stange	Barkers Mill Elementary	✓	EDIT
hoskinskatie1@gmail.com	Katie	Hoskins	Rossview Elementary	✓	EDIT
toysland.pw@gmail.com			Not Assigned	☐	EDIT
mstange@my.apsu.edu	User Marcus	Stange	Not Assigned	☐	EDIT

Items per page: 5 1-4 of 4

Figure 10: Users page of Emergency Class Manager

Since this file uses tables in the database, it imports Supabase. There are five functions in this file: count, fetch, search, getSuperAdmin, and searchSchools. The 'count' and 'fetch' function both get the users from the database, but 'fetch' allows them to be output onto the table and orders them as necessary. The 'search' function allows the user to search for a user in the table. The 'getSuperAdmin' function checks the database entry to see if the user is assigned as a super admin. Finally, the 'searchSchools' function works with the autocomplete use in the 'Edit' module to let the user search for a specific school. In this file, there are seven methods that are used: getSuperAdmin, loadRows, searchRows, schoolSearch, editRow, updateSchool, and submit. The ‘getSuperAdmin’ method uses the ‘getSuperAdmin’ function in order to check if the user is a super admin in the ‘Users’ table in the database. The ‘loadRows’ method uses the ‘count’ function to check how many users are in the table and then it uses the ‘fetch’ function to load those users into the table that they can see. The ‘searchRows’ method uses the ‘search’ function to allow the user to search for users in the table. The ‘schoolSearch’ method uses the ‘searchSchools’ function to allow the user to autocomplete schools when filling in the table. The ‘editRow’ method is called whenever the user opens the ‘Edit’ menu. All information that the user fills in for another user calls this method to update the table as needed. The ‘updateSchool’ method is called when the user tries to update the ‘school_id’ of the user. Since the school field uses autocomplete, it is updated separately from the other information that can be updated. This method calls on the ‘Users’ table to update information in the database. Finally, the ‘submit’ function updates all other information in the database. When the user edits information in the ‘Edit’ modal, this method gets called when the user saves. It calls on the ‘Users’ table of the database and updates the userEmail, fName, lName, and admin status in the database.

The last page file that is used is App.vue. This file contains information that can be reused later such as fetchUserData and the snackbar. All this file does is hold methods and functions that need to be used many times. This file contains the fetchUserData method and the snackbar file, but that is it.

The database uses nine different tables for all the information saved from the website. These tables are: Classes, Event Roaster, Events, Perm Roaster, Schools, Students, Teacher Classes, Users, and allowed_domains.

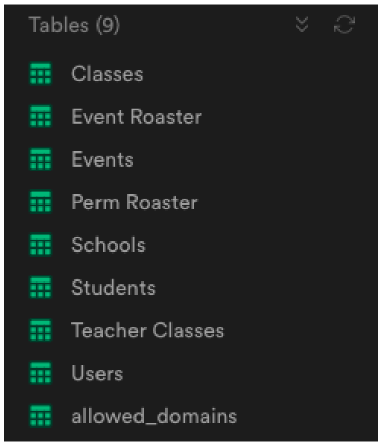


Figure 11: Tables in the database

The ‘Classes’ table stores the information for each of the classes that exist and is used in the Classes.vue file and the EditClasses.vue file. It contains a class_id of type int8 as the primary key, the className of type varchar, removed of type boolean, and school_id of type int8 as a foreign key that references the school_id column of the ‘Schools’ table. The className column cannot be null, but the rest of the columns can be. The ‘Event Roaster’ table stores information from a roster that has been submitted during an event. It contains a class_id of type int8 as the primary key but also as a foreign key referencing the ‘Classes’ table class_id, a student_id of type int8 as a foreign key that references the ‘Students’ table student_id, an event_id of type int8 as a foreign key that references the ‘Events’ table event_id, a status of type varchar, and a date of type timestamptz. The ‘Events’ table stores information that an admin puts in on the events page of the website. This table is used in TeacherEvents.vue, Events.vue, and the Dashboard.vue files. This table contains an event_id of type int8 as a primary key, date of type timestamptz, eventName of type varchar, description of type varchar, and school_id of type int8 as a foreign key that references the school_id column of the ‘Schools’ table. The ‘Perm Roaster’ table stores information of each student that is assigned to a class. This table is used in the EditClasses.vue file to get information about the students that are assigned to classes. It contains a class_id of type int8 as a foreign key that references the class_id column of the ‘Classes’ table, and a school_id of type int8 as a primary key and a foreign key that references the school_id column of the ‘Schools’ table. The ‘Schools’ table stores all relevant information about the schools in the Clarksville-Montgomery County School System elementary schools. This table is used in the TeachContact.vue file in order to get all information needed. The table contains a school column of type varchar, a school_id of type int8 as a primary key, a principal as type varchar, a pEmail of type varchar, an assistantPrincipal of type varchar, an aEmail as type varchar, an assistantPrincipalTwo as type varchar, an aEmailTwo as type varchar, an assistantPrincipalThree as type varchar, and an aEmailThree as type varchar. The principal column must have an entry, but the schools are not required to have any assistant principals if there are not any. The ‘Students’ table contains all information about students who are part of the school system and have been added through the student manager. This table is used by files including the EditClasses.vue, AddStudents.vue, and Roster.vue files. This file contains student_id of type int8 as the primary key, an fName of type varchar, a lName of type varchar, removed of type boolean, and a school_id of type int8 as a foreign key that references the school_id column of the ‘Schools’ table. The student_id and id_number are not able to be null, but the other columns are. The ‘Teacher Classes’ table stores the information for teachers that are in charge of classes and have been put at the primary teacher of a class and is used in the EditTeachers.vue file. This table contains a userEmail of type varchar as the primary key and as a foreign key referencing the userEmail column of the ‘Users’ table, a class_id of type int8 as a foreign key referencing the class_id column of the ‘Classes’ table, and isPrimary of type boolean. All of the columns are unable to be null in this table. The ‘Users’ table contains information about all users who are registered to the website. When a user registers, their information gets sent from the built in ‘Auth’ table to the ‘Users’ table. It contains a userEmail of type varchar as a primary key, fName of type varchar, lName of type varchar, phoneNum of type numeric, emergcyInfo of type varchar, admin of type boolean, superAdmin of type boolean, removed of type boolean, school_id of type int8 as a foreign key referencing the school_id column of the ‘Schools’ table, and class_id of type int8 as a foreign key referencing the class_id column of the ‘Classes’ table. The userEmail, admin, superAdmin, and removed columns are unable to be null, but the fName, lName, phoneNum, emergcyInfo, school_id, and class_id are able to be null. The final table in the database is the allowed_domains table. This table simply puts in email domains that are allowed to be used to sign up for an account. It contains one column called domain_name of type text as the primary key.

Deploying and Building the Application

In order to deploy and build the application, navigate to https://github.com/Toys0125/Emergency-Class-Manager and clone the repository. From there, open the repository in an IDE or terminal and input 'npm install' into the terminal that corresponds to the directory that holds the repository download. There is also a third party library that needs to be installed before the program can be run: https://github.com/fullcalendar/fullcalendar-vue. This is the library used to get the calendar for events. The user can then run 'npm run dev' in the terminal and wait for the local host to boot up. Once it has, the user can navigate to localhost:3000 and start working on the application.

For website hosting any website hosting provider or your own by running 'npm run build' will build a static website that can be hosted by reverse proxy and similar software.

To create the database, the user should navigate to https://supabase.com/ and start a new project. The user can create the tables from the design section of the report which includes the Classes, Event Roaster, Events, Perm Roaster, Schools, Students, Teacher Classes, Users, and allowed_domains tables. The user can then create their own information in the database. To use the database in the code, the .env file must be updated. Instead of the information the team used for the supabaseUrl and supabaseAnonKey, the user should input their own database information into this file or else they will be using data that has been used in the teams project and not their own. In the database, we used a trigger to set up the sending of a user from the built in the 'Auth' table to our own 'Users' table for easier use:

```
begin
  insert into public."Users"("userEmail")
  values (new.email);
  return new;
end;
```

This is needed so that the team can gathering their own information on users while being unable to edit the 'Auth' table.

Known Bugs

Most of the website works completely, but there are a few bugs that are currently being addressed. In the AddStudents.vue file, the user is unable to add a new student with the same id_number if it already exists, even if the removed column has been set to 'true'. We need the students to be able to be removed instead of deleted for submission of the roster and archives. For the next release, the team will try to check if removed is false in the 'Students' table and allow the user to put multiple id_numbers of the same number in the table if removed is true. Another bug is that when an admin goes to view and edit classes, all classes appear no matter the school. For the next release, the team will compare school_id in the 'Classes' file to the school_id in the 'Users' file to only get classes for the assigned school.

Future Work

Some features that the team wants to add in the future is: an archives page, usage for more schools including middle and high school in CMCSS, a chat implementation for teachers and administration to easily communicate, an implementation of a roster submission to automatically add students from an account like PowerSchool, and an easier to use calendar where the user can click on the date and automatically add an event. The team also wants to make a nicer implementation of the event reports as well. These items can be easily added in four months in order to have an updated release by then.