CSCI 370 Final Report

DECTechnicians

Braden Gehr
Trevor Hartshorn
Micah Munoz
Tyler Rotello

Revised June 16, 2023

CSCI 370 Summer 2023

Prof. Rob Thompson

Table 1: Revision history

Revision	Date	Comments
New	5/15/23	Started Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements
Rev – 2	5/19/23	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done
Rev – 3	5/26/23	Completed Sections: I. System Architecture
Rev – 4	6/1/23	Updated Sections: I. System Architecture (risks/difficulties portion) Started Sections: I. Software Test and Quality II. Project Ethical Considerations
Rev – 5	6/2/23	Updated Sections: I. Non-Functional Requirements Completed Sections: I. Software Test and Quality II. Project Ethical Considerations
Rev – 6	6/9/23	Updated Sections: I. Results A. added current results B. added tests still being implemented II. Definition of Done III. System Architecture A. which design difficulties mitigated B. added section for email architecture
Rev – 7	6/12/23	Updated Sections I. Results A. Updated results table B. Modified tests/functions not implemented and summary of results II. System Architecture A. Added diagrams, email system Added/Completed Sections: I. Future Work II. Lessons Learned III. Acknowledgments

Table of Contents

I. Introduction.....2
II. Functional Requirements.....2
III. Non-Functional Requirements.....3
IV. Risks.....3
V. Definition of Done.....4
VI. System Architecture.....4
VII. Software Test and Quality.....6
VIII. Project Ethical Considerations.....9
IX. Results.....10
X. Future Work.....10
XI. Lessons Learned.....10
XII. Acknowledgments.....10
XIII. Team Profile.....10
References.....10
Appendix A – Key Terms.....10

I. Introduction

DECTech (Discover, Explore, Create with Tech) is a program founded by Dr. Tracy Camp and a group of her female Mines Computer Science students to kindle the interest of children in STEM. Initially (and predominantly) targeting girls in grades 3-6, but has since expanded into a wider range of grades and genders. The program still has the primary goal of developing underrepresented groups in STEM; however, the client, Dr. Christine Liebe, believes that an overhaul of the current registration system could greatly help the program.

The project goal is to streamline the registration service for this program, as the current registration system has high labor requirements. Not only will the product ease the process for DECTech administration, but it will hopefully also make the system easier for guardians/students to access, regardless of their registration interests. The product should be able to take in the information of registrants, sort the information into rosters for the camps/programs automatically, and be malleable, amongst the other requirements; malleable here meaning that DECTech admins should be able to change questions and information within the website without excessive struggle. The foundation from the last team does have admin capabilities; however, they must use PSQL to manually change information in the database and forms, making the overall system unfriendly for common users. The new changes aim to make registration and confirmation amongst guardians and administrators more efficient and less labor-heavy than before.

Currently, there is a GitHub repository that holds the foundation and initial steps taken by the last team who worked on the project. They have made a login system and a test database which holds the guardian login information and general guardian and child information. The team has expanded on this foundation, including additional forms to acquire other necessary information, a FERPA-compliant Mines' database, and other niceties that were requested by the client (summary information and CSV outputs).

II. Functional Requirements

The first requirement of product development is that it must take in student and guardian information and store that information into a FERPA-compliant database. Some of this was done by the previous team, which we have expanded on, mostly by modifying the Mines server/general web page and database in which this data will be held and collecting auxiliary information about the student and course they would like to register. This system must be able to handle both registration for after-school (Fall) courses and summer courses, as well as being accessible by mobile device. Along with that, the new form should collect and store the same information that is already collected. In other words, the form should match the current Google forms that the website uses for registration. The product would automatically fill rosters, preventing registration when a roster is full, as well as giving administrators copies of the rosters for running the courses.

Along with storing information and giving roster output, Christine requested that the system send automated emails to guardians to confirm their child's registration and convey other information. After registering a child for the program, an automated email would be sent using the email connected to the guardian in the database to confirm that the child has been registered for the course. This email would also contain an embedded link to pay for the course through the Mines' system. Other automated emails would include information about the course, such as the time and location of the course they would be taking, which would be sent closer to when the course is held.

The team was given some tasks that were not as high-priority by the client as 'stretch goals'. The first is a CSV output for the DocuSign process, in which guardians must electronically sign consent forms for their children. Another non-priority task was a summary dashboard accessible by administrators, which would give them statistics about the children enrolled, courses taken, and other information. The summary statistics would also be filterable by admins, depending on what data they wanted to see.

III. Non-Functional Requirements

One of the most important non-functional requirements given to the team is that the database be FERPA compliant. This is extremely necessary, as the information collected about the children enrolling in the program is protected and dictated by FERPA, and not doing so would present legal troubles for DECTech and the Colorado School of Mines itself. The database must be consistent and dependent, which the team dictated is out of our project's scope and would depend more on Mines' ITS teams. This means that the database in which DECTech stores information about registrants should be secure and reliable in the event that it goes down, and hold information about previous registrants. This relates to another non-functional requirement, which is that once in the system, both guardian and child should be able to register for future courses without having to re-enter their information.

Another non-functional requirement is that the overhauled web app should be administrator-alterable, without the use of raw code. To make changes in the current foundation, an administrator would have to access GitHub or the database and change the source code or use PSQL to manually alter information. However, Christine has requested that the product be more malleable, such as being able to change questions and student information through the website itself, not the source code. To go along with this, the altered codebase should be well-documented enough such that an inexperienced or new team can continue the project without excessive difficulties, such as in the case that the team does manually access the database or codebase.

Lastly, the web app should be somewhat similar to the current website design, if not more aesthetically pleasing. This is extremely low on the team's priority, as the team is more concerned with fulfilling requirements and getting a working product that is as close to being able to roll out as possible.

IV. Risks

The risks that the team was most concerned about were the data privacy and compliance of the database. Since the team was dealing with information about children, as well as login information about the guardians wanting to register those children, the database and login system needs to be secure, safe from attacks, and FERPA compliant. The greatest risk would've been the involuntary release of a child's information due to an attack, such as a man-in-the-middle or SQL injection attack. Another risk the team identified relating to the database was the general loss of data. The DECTech team wanted this web app overhaul because the current system is inefficient for both guardians and administrators. A general loss of data would be very detrimental to the efficiency of registration, as well as to running the program overall. Administrators would then have to try to reform rosters and acquire information from the guardians again.

Another set of risks the team identified related to the codebase and running the existing code itself. Since the team was mostly unfamiliar with web development, as well as JavaScript, there were a few concerns that arose. Firstly, being able to familiarize ourselves with and run the existing code generated by the previous team was imperative to the project. If the team was unable to do this for any reason, such as poor organization or documentation of the existing code left by the last team, the team would be unable to complete the project in any timely manner, if at all. Another risk of being unfamiliar with the existing code and JavaScript included involuntary software degradation. It was a possibility that the team could cause the existing code to fail, meaning that not only was progress on the project not made but there were actually steps taken in the reverse.

V. Definition of Done

A minimal goal for this project is to flesh out the existing foundation formed by the previous team and get the website as close to rollout as possible, if not ready. To be more specific, the minimum the team would consider 'done' is new/modified forms that would take in the same information as the existing Google forms (information that is needed by the DECTech team). These forms would have to input this data into a database that the team has made. This database will be FERPA compliant and be run through Mines itself since DECTech is a Mines program. The database and information should be administrator modifiable through the website, instead of the current manual access system. The database will then be automated in a few ways: emailing and roster formation. The automated emailing system will both: email guardians confirming their child's registration (and sending them the link for payment through Mines after filling out the forms on the website, and email guardians course information, including time and location, closer to the course start date. The automated roster formation will be performed within the database, which is how the seat cap will be applied and will be able to be output in CSV form. The administrator page should also be mostly functional, which ties back to the database/information being administrator modifiable in the website.

VI. System Architecture

General Architecture/Process Diagram

The general system architecture for the project involved the web page for users to interact with, which interfaces with the database to store their information and register them for classes. The web page allows users to create their accounts, add their children to their accounts, register them for after-school or summer classes, or update any of their information. This web page uses ExpressJS to update the database, as well as display the information the user has in the database already. As stated previously, the foundation of this project was laid out by the previous team, which was mostly login and registration capability. This means that the architecture specifically built by our group is adding to this system, including modifying the registration forms and making them modifiable by admins, adding features, such as when users forget their passwords and an auto-emailing confirmation system, adding an admin page, in which admins can edit information currently in the database (guardians, children, classes, etc.), view the info in the database with settable filters, and download the CSV files needed for rosters and DocuSign or view summary statistics.

The process diagram in Figure 1 below illustrates the general process that a user would experience when entering the web page. This process diagram is a modified version of the last team's [1], as they were mainly focused on the login and general information section. Our team added the admin split from the guardian login, as well as some other minor changes, like info/registration changes and the email system. Now, when logging in, the back end checks whether the user logging in is an admin or instructor, in which case it transitions them to the admin page instead. This page includes options for administration, such as changing information within the database or accessing the statistics and CSV downloads. In the case that the user is a guardian, from the work done by the previous team, they would be prompted to sign in/create an account (prompting a verification email) and then add or change their child's info. They would then also be able to access the registration form, leading to a confirmation email as well.

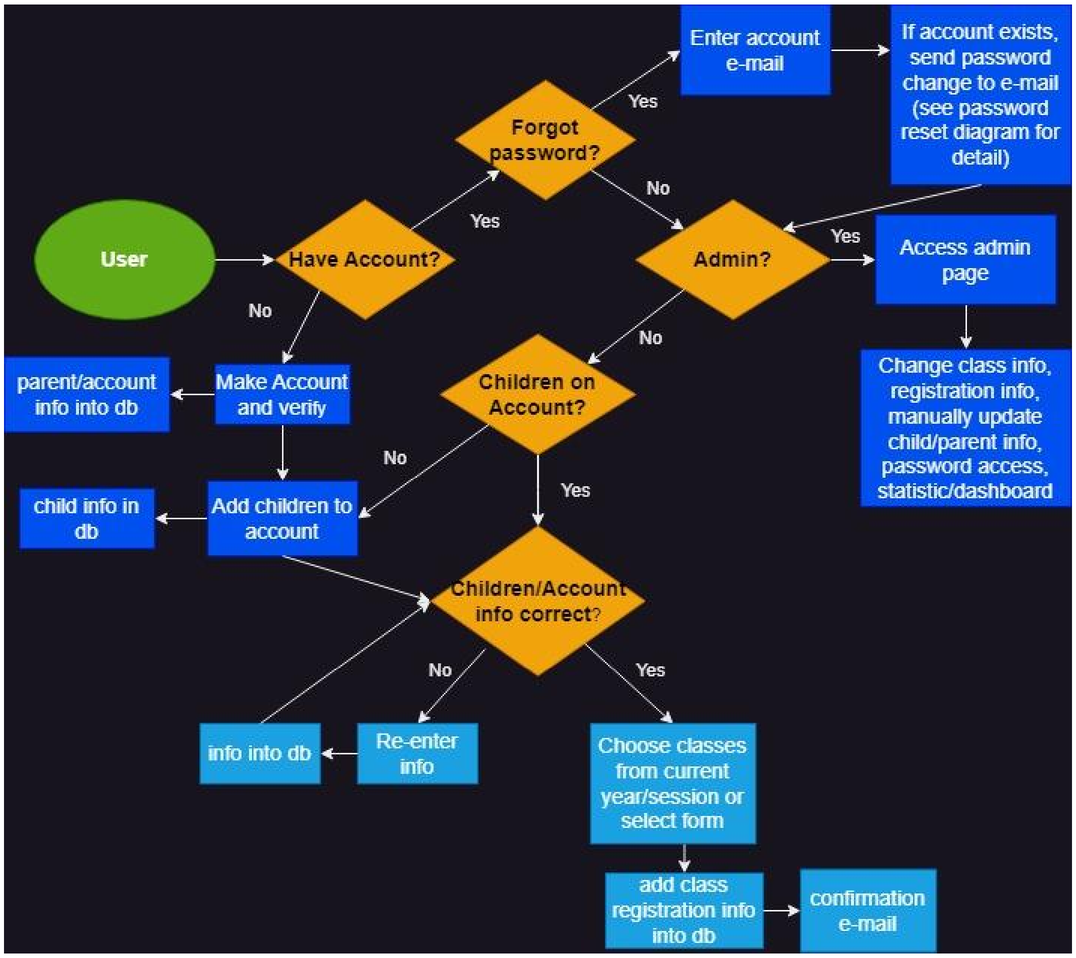
User
Have Account?
Yes
No
Forgot password?
Yes
No
Enter account e-mail
If account exists, send password change to e-mail (see password reset diagram for detail)
Admin?
Yes
No
Access admin page
Change class info, registration info, manually update child/parent info, password access, statistic/dashboard
Make Account and verify
parent/account info into db
Children on Account?
No
Yes
Add children to account
child info in db
Children/Account info correct?
No
Yes
Re-enter info
info into db
Choose classes from current year/session or select form
add class registration info into db
confirmation e-mail

Figure 1: Process Diagram [1]

Database Architecture

Figure 2 below visualizes the current and modified version of the database. The previous team left us with some PSQL that establishes entities within the database: child, parent, and class, as well as a portion of the codebase that deals with editing those tables [1]; however, our team has decided to add 3 flags: Guardian, Admin, and Instructor. As it stands, the team is only adding those 3 flags, though we did consider modifying the database in different ways, such as breaking down/normalizing the entities, or adding two separate entities for Admin and Instructor; however, after consulting with the previous team's proxy, Emily Hime [1], she advised that we keep the database as simple as possible, as to avoid confusion for future management or team members; therefore, we only added flags to user (renamed from parent) to differentiate between guardian, instructor, and admin users.

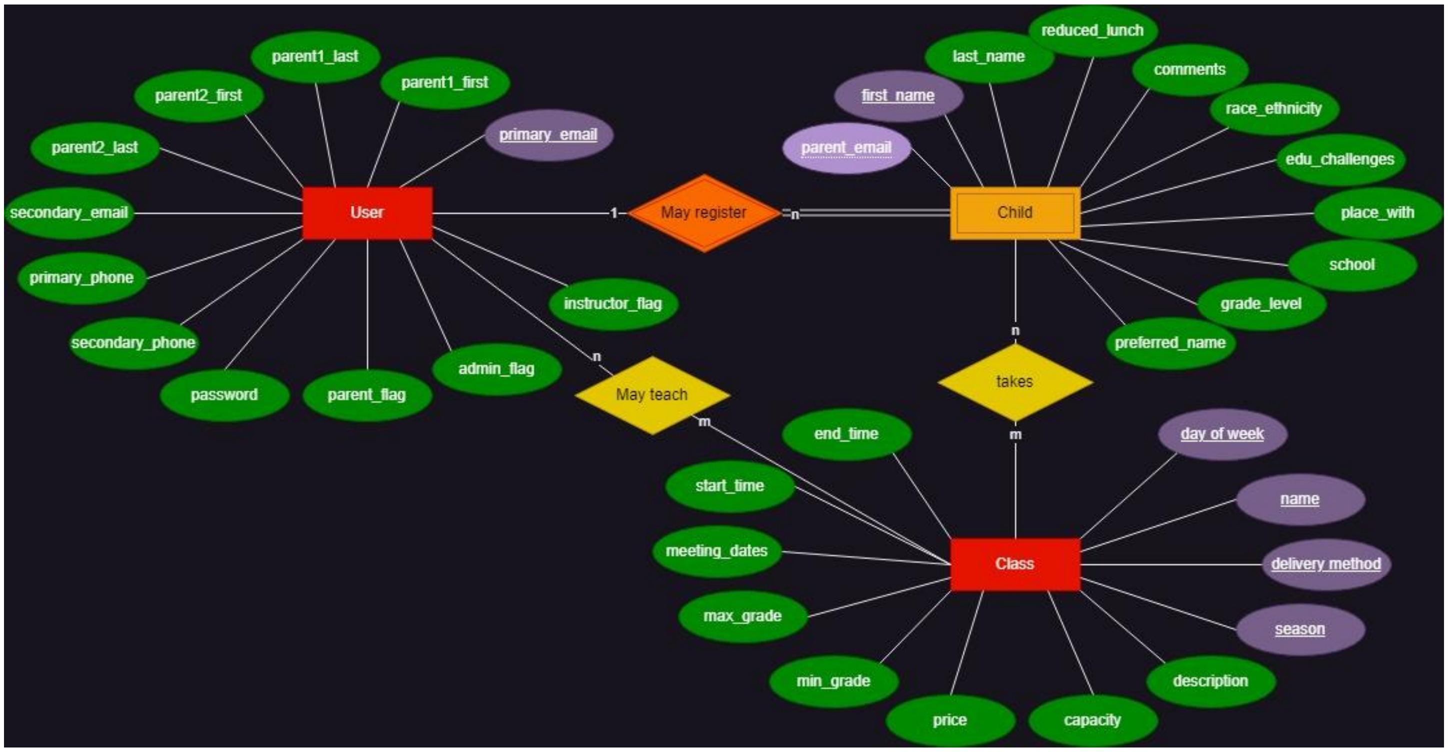

Figure 2. ERD of the DECTech Schema

Emailing Architecture

Figure 3 below illustrates the flow of system calls in order to automate the emailing system, more specifically for resetting passwords. This system promptly sends emails when a user requests/performs certain actions. For example, requesting to reset their password (if they've forgotten it) or after registering their child for a DECTech course. It was requested that these emails be sent through the stem-tech@mines.edu distribution group for consistency and professionalism.

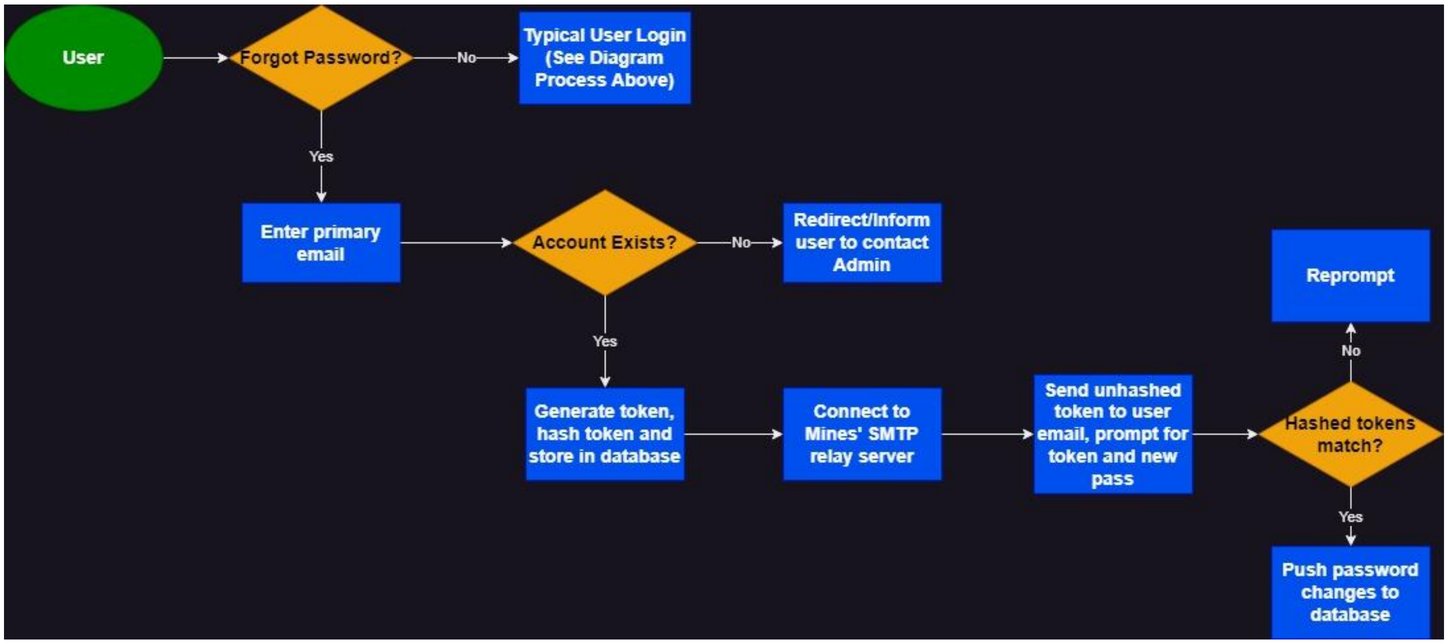
User
Forgot Password?
No
Typical User Login (See Diagram Process Above)
Yes
Enter primary email
Account Exists?
No
Redirect/Inform user to contact Admin
Yes
Generate token, hash token and store in database
Connect to Mines' SMTP relay server
Send unhashed token to user email, prompt for token and new pass
Hashed tokens match?
No
Reprompt
Yes
Push password changes to database

Figure 3. Flow Diagram for Resetting Password

A similar process can be followed for sending the confirmation emails, in which after a user registers their child, a connection to the smtp.mines.edu relay server is established and a NodeMailer (see Appendix A) transporter sends the email telling a user that registration was successful. This email is also sent through the stem-tech@mines.edu distribution group handle.

Design Difficulties

One of the design difficulties the team faced was getting access to a Mines' server in order to establish a database and test remote connections, as we were only running locally and testing everything on local databases, without knowledge of if remote connections worked. This was resolved by contacting the previous team's proxy (and former DECTech member) Emily Hime, as we did not know that they had already established a PSQL schema in the Mines' codd.mines.edu(See Appendix A) server. The team was also able to get a VM allocated by the Mines' HPC (See Appendix A) Server Team, which allowed for separate remote access.

The team also faced difficulties regarding the emailing system. In order to perform tasks like automated email confirmation when registering a child or password reset, we thought that we needed access to SMTP(See Appendix A) emailing systems like MailChimp (see Appendix A) or SMTPjs since JavaScript cannot interface with SMTP ports directly. The DECTech team uses a paid MailChimp subscription for mass-distributed emails but wanted to reserve that system for bulk emails and marketing. Through contacting ITS, the team learned that using the stem-tech@mines.edu distribution group is possible by routing emails through the smtp.mines.edu relay server. This meant that using NodeMailer (without authorization) and no third-party system was possible as long as the machine sending the email was connected to the Mines' network.

Another significant, and arguably more urgent, difficulty that the team faced relates to the existing codebase left to us by the previous team. A large portion of the tasks requested of us could utilize existing framework modules to simplify, such as password reset; however, some of the codebase is either written in a way that does not support the framework capability or uses different (typically manual) methods. The team chose to continue the existing codebase design, following similar methods to implement our functionality manually.

Component List

The entire application is highly interactive, but at a high level, a developer can think of the VUE (see Appendix A) files being the frontend implementation, and the JavaScript files being the backend, with serve.js containing all of the Axios (see Appendix A) server calls that make the requests from the frontend.

Pre-Existing/High-Level Overview (**See previous team's report for more details on existing, if needed*)

The main.js file creates the web app using App.vue and mounts it in the register.html file. The App.vue file is what manages the .vue files, receiving the calls to switch VUEs and some other minor functionality. First, database.js is the file that establishes the database, creates the user, child, and class tables as well as their cross-references, and holds the functions that query the database and add/change information within the database. userAccounts.js handles most of the data and functionality relating to the users' accounts, their children, and encryption. The .vue files interact with these javascript files as follows:

- registerHome.vue
    - Page that the user sees upon clicking 'register' from the homepage. This page prompts the user to either login or signup, depending on whether they are a new or existing user.
- signupForm.vue
    - This is the page that the user sees when they choose to signup if they are a new user. This page only prompts them for the name, primary email, and password to create a basic account.
- loginPage.vue
    - This is the page that the user sees when they choose to log in, prompting them for primary email and password to sign in. There is also a forgot password function (see next section)
- loggedInPage.vue
    - After signing up or logging in, the user is brought to this page. Depending on the permissions, a user will have 4 or 5 different buttons: updating user info, updating child info, adding a child, registering a child, and, if the user is an admin, a button to access the admin page.
- updateChildren.vue
    - This page has all of the child informational fields, filled with the information already within the database. The information is alterable and after submission, will call the server to update the database.
- updateUser.vue
    - Similarly to the updateChildren.vue, this is where a user may update the information they have in the database.
- registerForm.vue
    - This is the form that a user will select which child is being registered, the class type and class itself, input any educational/additional information, and sign to submit and register their child.
- classRegistrationForm.vue
    - (Left by the previous team; however, we do not see any interactions) It seems that this file is a file that the previous team created as a separate form. It collects the same information as registerForm.
- childInfoInput.vue
    - This is the form a user sees when adding a child. It prompts the user for the child's name, educational challenges, etc., and calls the server to push the information to the database.
- fullRegistrationForm.vue
    - This file exists only in the mono repository (not in the previous team's repository), which handles a few of the Mines' CS websites; however, we may decide to add it to our repository as well. This is a full user form which is essentially a mirror of the Google form that the DECTech team is currently using. This is a

bypass of the account creation system made by the former lead intern, Emily, to see if rollout was minimally possible.

Note: The vue files may have been minorly altered; however, the main changes were made to database.js, userAccounts.js, App.vue, and the files added, which are detailed in the next section.

Altered/Added Components

The added the system calls and functions to handle the functionality we were asked to implement in the JavaScript files detailed above, as well as serve.js. In database.js, we implemented the changes that we made to the schema, including changing the name of the parent entity to users (and refactoring all of the related calls/queries), and adding attributes to users, such as the flags for admin and instructor. We also added functions for querying/updating the database from the admin's submissions on the admin page, as well as the functions to query/update the database for the reset password functionality, notably token storage and password update on the database side. userAccounts.js has added functionality for password reset/encryption, which calls the functions previously referenced in database.js, as well as the functionality for sending the confirmation email after a user has registered their child. We've also added resetPass.js, which mostly handles the functionality for the forgotten password system, like checking if a user is valid and generating the token for validation. The added VUE files interacting with the existing and altered JavaScript files are as follows:

- admin.vue
    - This was technically existing when the team joined the project; however, only had "This is the admin page!" This page displays the options for an admin when joining, which the team implemented the user, child, and class manual information access and update with filtering.
- adminClasses.vue
    - This is the page where the admin may filter, view, and update class information manually.
- adminChild.vue
    - This is the page where the admin may filter, view, and update child information manually.
- adminUser.vue
    - This is the page where the admin may filter, view, and update user information manually.
- forgotPassword.vue
    - This page is linked to the login page. It prompts a user for the primary email of their account and validates the email provided. If the user exists, it will make the server calls to begin token generation.
- resetPassword.vue
    - This page appears after a user has provided an email to forgotPassword.vue. It prompts them for the token emailed to them, as well as a new password (two fields for matching password). This page makes the server calls to match the token and push password changes if validated.

Note: Again, the VUE files are mostly just frontend, whereas the functionality was implemented through the Axios calls in serve.js and the new implementation in the different JavaScript files.

VII. Software Test and Quality

Note: The team is also using the unit tests(See Appendix A) the previous team implemented that are still in the codebase.

For example, the general registration test and login test were slightly expanded upon by our team. They also had other unit/component tests (See Appendix A): registration test, edit account information, class list, downloading spreadsheet, login test, encryption test, and API test. These tests are fairly self explanatory by name, but their descriptions are as follows[1]:

Registration Test	“Check that parent and child information can be queried in the database after registering”
Edit Account Information	“Check that the database gets updated when a user changes the parent or child information associated with their account”
Class List	“Checks if the registration form can correctly display the available classes read in from a file”
Downloading Spreadsheet	“This will test if information from the database can be extracted and put into a readable form”
Login Test	“This will test that users will get their information back when they login”
Encryption Test	“This will test that passwords are stored in encrypted formats”
API Test	“This will test if the database is updated upon requests and that Http requests are handled”

Our tests are as follows:

Name	Purpose	Description	Tools Used/Type of Test	Desired/ Acceptable Result	Edge Cases	Result
Admin Creation Test	Test admin creation and population	When an admin is created, it should be stored with correct flags. Additionally, an admin should have all of the correct information pushed into the database.	Unit	Admin is pushed into the database, and the flag is set correctly.	An admin may be a guardian as well, in which case both flags should be set	Pass, Admins are stored in the database correctly, when the flag is set manually

General Registration Test	Test child creation and form input	When a child is created and form filled out, the child should be pushed to the database and cross-referenced with a guardian (and class, if appropriate). The database should have this information inserted.	Unit, Component	The child and cross references should exist in the database, matching the info that was inserted.	A guardian could attempt to register a child twice, in which case a second cross reference should not be made	Pass, Child creation pushes data into the database, registration also pushes to the database. Child cross references guardian and class (if registered).
Form / Information Alteration Test	Test the functionality of form/question changing by admins	When an admin changes a question or information on the form or for a class, the form should update and those updates pushed to the database. This should appear for all users on the refreshed page.	Component	The form should update on all ends, including any users who view the form after the change. This should result from the question being changed in the database, or in the VUE file itself.	The question or class on the form could be altered while someone is viewing the form	(Partial Pass) Admins are able to alter the information of guardians, children, and classes; however, the filter functionality should be further tested. The team was unable to develop manual form question changes without changing the code in the repo.

GUI Test	Test the GUI acceptability and functionality	Make sure that the Admin GUI is acceptable for the client, and that the buttons on the Admin page lead where they are supposed to.	Component, Client Review	The client should accept the GUI and the correct VUEs loaded when prompted	Buttons can be spammed, dragged, etc. in which case the VUEs should not break	Pass, Client has accepted GUI for general login page, and admin page. Buttons on both pages route to correct VUE files.
Login Test	Test user login, both admin and guardian	Ensure that logging into accounts gives access to correct pages/buttons (admin-flagged users have access to admin pages, guardian flagged users have access to guardian pages).	Component	The guardian flagged buttons should only appear for guardians, and admin flagged buttons for admins	Again, a user can be both guardian and user, in which case all buttons should appear and be accessible	Pass, general login page shows options to add child, register, etc. but only admins see the button to switch to admin VUE.

Forgot Password Test	Test that the forgot password feature is functional	The forgot password feature should only work for emails registered to existing users. Additionally, the emails should be sent automatically, with the new password being inserted into the database correctly and the new login working afterward.	Component	The forgot password pages should load, and have email/new password entry, which should result in a correct password reset.	Non-registered emails may be entered, and new passwords entered may not match, neither case should allow password reset.	Pass, once users are verified as existing in the system, automatic emails are sent with token information. The tokens are hashed and stored in the database and when a user re-enters the correct token, their password is reset.

Automated email test	Test that the automated email system is functioning	The email system should automatically send emails to users after they register a child. It should also send emails closer to the time at which the class occurs with relevant information.	Unit, Component	The emails should be sent to registered users after child registration, close to class times in which children are registered, and when requesting a password reset.	A guardian could have multiple children registered for the same course	(Partial Pass) Emails are sent to Guardians after registering their child. This includes which child was registered and for which class. We were unable to implement the reminder email system that sends emails to guardians closer to the time of the class
CSV Download Test	Test the download links for CSVs	The links to download the CSVs on the admin page should download the stated CSV, with the pertinent information. This should also be possible for previous years.	Component	The CSVs should download correctly, with each CSV being linked to the correct information	A user could request to download multiple CSVs at once	Pass, links on admin page allow for CSV downloads of rosters, full guardian list, etc.

VIII. Project Ethical Considerations

The project must align with the IEEE Code of Ethics, the ACM Code of Ethics and Professional Conduct, and FERPA, and the appropriate ethical considerations are listed below.

1. Storing the information of minors/general information
    a. FERPA, IEEE 2.05/3.12. (Keep confidential information private in accordance with public good and law), ACM 1.6 (Be private with sensitive information and keep information usage legitimate).
    b. Since the system is storing sensitive information in guardian accounts, it is crucial for the team to keep this information secure and private, as with any good database practice. This is especially true since it

also stores the information of their children in this educational program, which makes it important that the educational and personal records of these children are stored securely by the Family Educational Rights and Privacy Act (FERPA).

2. Ensure password/account security
    a. IEE 3.12/3.14 (Data privacy and integrity), ACM 2.9 (Developing robust and secure systems)
    b. Given that these accounts are used to register children, store both guardian and child information, and view information about classes (in general, and which classes a child is registered in), it is important that accounts stay secure, especially through the encryption of the account login information.
3. Consistency with the current DECTech team and system
    a. IEEE 3.06 (Follow professional standards), ACM 2.3 (Respect existing rules of professional work)
    b. Since the DECTech team is established and has its own method of conduct, the product must align with its methods and procedures. As in, the DECTech team should not have to greatly alter their standards or procedures in order to use the product.
4. Proper documentation and readability of code (for future teams/management)
    a. IEEE 3.11 (Adequate documentation), ACM 2.1 (High work quality)
    b. Since we are an inheriting team, we especially know the importance of well-documented code. This codebase may again be passed on to a future field session team, or a progressing DECTech team, and therefore should be well documented in order for that team to produce good quality and timely code. If our code is not well-documented, this will produce delays, confusion, and possibly even code degradation.

IX. Results

Summary

**See Section VII. Software Test and Quality for full results*

The team was asked to continue the work of a previous team to make the website and new registration system for DECTech as close to rollout as possible. Specifically, we were asked to make admin accessibility, password management, and automatic emailing a priority. The website allows guardians to create accounts, add their children to their accounts, update child and guardian information, and register children for specific courses. The tests for populating these entities and their cross-references, both automatic and manual, have passed such that the account creation system and registration system are essentially complete, including both admins and regular users. After discussions with the client and her proxies, it was determined that the instructor population is a nicety, but is not necessary for functionality.

The automated emailing system, including the functionality for resetting user passwords, has been completed and tests passed. Emails are automatically sent when a user registers their child and when they request a password reset. The emails are successfully sent through the stem-tech@mines.edu distribution group handle, since we used the smtp.mines.edu relay server, as detailed in the System Architecture section.

We have also further developed the admin page, which was left mostly barebones by the last team. The VUEs are developed; however, further testing must be done to ensure that the filters work such that information access is completely functional (some of the filters have some bugs, but are mainly functional). The admin page allows the manual information change of guardians, children, and classes; however, due to time limitations, we were unable to achieve manual registration, form question alteration, and the summary dashboard.

Essentially, the account creation system and registration are complete. The tests for GUI and VUE switching passed, as well as the tests relating to accounts and registration. The front-end VUEs successfully interact with the back-end VUEs through the Axios calls interacting with NodeJS (see Appendix A). This means that the data is correctly pushed to the database (passwords and tokens being hashed first). General information (such as rosters, full information, etc.) access and alteration is successful; however, more development for the admin page should be done, not necessarily for functionality, but for admin convenience. The new system works, but some niceties could be implemented for admin and guardian convenience. Therefore, the minimal implementation needed is the SSL certification for the web page, which is the only item preventing the new system from going live. The next section details the partial pass or no result tests.

Partial Pass/Did Not Get To

The admin page filter system needs to be further developed and tested, as the filters will occasionally not work quite as intended, but do produce the correct (filtered) results a majority of the time. For example, the filters will occasionally switch what tags the information is being filtered on, if the tags have the same data type. Additionally, the time filter is not properly comparing time formats. The admin form alteration (question-wording, not data intake), as well as manual registration, should also be further developed as the team was unable to get to that functionality. Lastly, the team was unable to implement the automated reminder emails closer to the time in which courses are being held.

X. Future Work

To get the website fully functional, the team believes that more development must happen, specifically regarding website security. For example, we believe that more testing, and likely even consulting with those more experienced with web security, such as the Mines' data privacy team, should be conducted to make sure the website is as secure as possible. Once again, since the educational information of minors is being handled, this website must be FERPA compliant and extremely secure. Another aspect of the security that the team was unable to resolve is the signed SSL of the implemented web pages. Currently, when the developed VUE files are pushed to the live website server, users are prevented from interacting with the login and registration system, as it lacks a certified SSL certificate. The SSL certificate is what identifies the connection as secure and encrypts the data that is passed between the website and the server it is being hosted on. The live website has an SSL certificate which we initially believed would also cover our developed VUE files which live within one of the website's HTML files. The team also explored self-signing an SSL certificate, though neither method worked. We also think that further consultation with more experienced security teams could resolve this.

Another aspect of the website that we believe could be further developed, while not technically necessary for the registration system to function as intended, is the form modification system. The admin page that we developed mostly supports general information retrieval and modification; however, it cannot change questions on the form (wording, dates, etc., not what data is collected). A manual registration system could also be fleshed out, such that admin can manually assign children to classes if their guardian cannot do so, for any reason.

The team, along with Emily, also thought of a few functionalities that could be implemented as niceties, or just to make the system a bit more efficient. First, implement a tab for users to see what classes their child is enrolled in, along with the confirmation email. Relationally, having an automated email for summarizing the enrollments, either daily or weekly, could be implemented. We also believe a back button on a lot of the pages would probably be nice to have.

**Future teams: see the website code review document, as well as the other documentation included in the Google folder, for more detail and functionality on future work/bugs/unimplemented features.*

XI. Lessons Learned

One of the most significant lessons we learned as a team is that familiarity with your codebase is **vital**. There were a few problems that we encountered that could've likely been avoided had we been more familiar with the existing workflow, and how the previous team connected all of the files. We also spent a lot more time than we would have liked trying to understand the previous codebase and some of the systems they had set up since they had only provided us with a general overview of the files' functionalities, and minimal commenting. Therefore, we tried to apply this by including more commenting, and hopefully more documentation than we were provided, such that an inheriting team should be able to work a bit more efficiently.

The team also learned just how important structure and management are for high-quality development. There were a few times in which the team failed to follow the AGILE/Scrum process, and we were both confused and disorganized when trying to proceed with the project. This resulted in time wasted and poor quality code, since we didn't know who was working on what, nor what was supposed to be prioritized.

XII. Acknowledgments

The team would like to thank Emily Hime, a former lead intern for DECTech, as well as the previous team's proxy for Dr. Liebe, as she helped us tremendously throughout the whole process and continuously encouraged us with check-ins and suggestions of areas to focus on. We would also like to thank Dr. Liebe herself, as she was wonderful to work with, helping us with connections (like with Emily and the previous team) and being extremely communicative.

Lastly, the team would also like to acknowledge how helpful our advisor, Rob Thompson, and Tree Lindemann-Michael were during this whole process. Through giving us advice on connections, and general process advice, they were a great help overall.

XIII. Team Profile

Braden Gehr

- *Computer Science major at Colorado School of Mines*
- *Born in Modesto, California, and grew up in Louisville, Colorado*
- *Work Experience pertaining to Computer Sciences: None*
- *Hobbies: Reading fantasy novels, going on a run*

Trevor Hartshorn

- *Computer Science Major*
- *Grew up in Bethlehem, PA*
- *No relevant work experience*
- *Hobbies: Reading, creative writing, drawing, video games*

Micah Munoz

- *Computer Science Major, Minoring in Computational and Applied Mathematics*
- *Hometown: Sacramento, CA*
- *Work Experience: N/A*
- *Hobbies: Powerlifting, Cooking*

Tyler Rotello

- *Computer Science Major*
- *Hometown: Lone Tree, CO*
- *Work Experience: Mobile App Dev., IT DB work*
- *Hobbies: Climbing, Reading, Math, Basketball*

Previous team: Anna Pitcock, Audric Wang, Audrey Powers, Natasha Bailey,

Working alongside previous team: Emily Hime

References

[1] E. Hime, A. Wang, A. Pitcock, N. Bailey, and A. Powers, DECTechTives, rep.

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations, and acronyms

Term	**Definition**
HPC	*The High Performance Computing department at Colorado School of Mines*
SMTP	*Simple Mail Transfer Protocol, TCP/ICP protocol for sending/receiving mail*
codd.mines.edu	*Mines' PSQL host.*
Unit Test	*Test implemented in the codebase to automatically test a feature of the project, typically back-end.*
Component or Procedural Test	*Test that is manually performed to ensure that features work as intended, regardless of back-end implementation.*
NodeJS	*Event-driven JavaScript runtime for building/expanding network applications. This is the engine for the web page, allowing for interaction on Mines' servers.*
Axios	*HTML Client for NodeJS, driving server calls*
VUE	*JavaScript interface for user interaction. (i.e. lives within the registration HTML, allows users to submit information, etc.)*
MailChimp	*Bulk emailing system that follows SMTP, paid service*
NodeMailer	*SMTP transfer service for NodeJS, takes host server, port, and authorization (optional) to send mail on a transporter object.*