COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

TechDECs

John Jagger
Curiosity Steen
Nathaniel Garner
William Wheaton

Revised May 13, 2024

CSCI 370 Summer 2024

Dr. Iris Bahar

Table 1: Revision history

Revision	Date	Comments
New (initialize)	5/13/24	Added group member names, and group name to title page.
Rev – 2	5/14/24	Added logo Work on requirements: ● High-level description/vision of the product. ● A list of the functional requirements. ● A list of the non-functional requirements. ● Potential project risks. ● Definition of done.
Rev – 3	5/16/24	Continued work on: ● Introduction ● Functional and Non-Functional Requirements ● Potential risks ● Definition of done
Rev – 4	5/20/24	Started: ● System Architecture
Rev - 5	5/30/24	Started: ● Software Test and Quality
Rev - 6	6/5/24	Started: ● Project Ethical Considerations
Rev - 7	6/7/24	Started: ● Results/Project Completion Status
Rev - 8	6/10/24	Continued work on: ● Minor edits ● Introduction Started: ● Future work ● Lessons learned ● Acknowledgements ● Key Terms
Rev - 9	6/11/2024	Continued work on: ● All Added: ● Appendix B
Rev - 10	6/13/2024	Started: ● Integrating peer feedback ● General edits
Rev - 11	6/14/2024	Continued work on: ● Integrating peer feedback ● General edits

Table of Contents

I. Introduction..... 3
II. Functional Requirements..... 3
III. Non-Functional Requirements..... 4
IV. Risks..... 4
V. Definition of Done..... 5
VI. System Architecture..... 5
VII. Technical Design..... 7
VIII. Non-Technical Design..... 11
IX. Software Test and Quality..... 11
X. Project Ethical Considerations..... 14
XI. Project Completion Status & Results..... 15
XII. Future Work..... 15
XIII. Lessons Learned..... 16
XIV. Acknowledgments..... 16
XV. Team Profile..... 17
References..... 18
Appendix A – Key Terms..... 18
Appendix B - Components/File List..... 19

I. Introduction

DECTech (Discover, Explore, Create with Tech) is a STEM outreach program ran through Mines with a focus on girls in STEM. The program provides unique experiences for 3rd-10th grade students through afterschool and summer camps. These camps involve hands-on activities that introduce the students to a variety of STEM topics such as biology and robotics. The program was founded by Dr. Tracy Camp back in 2012, initially accepting girls in grades 3-6, but then expanded to all genders. Dr. Christine Liebe now works as the head of DECTech, having worked with two previous teams on the website.

The first field session team (2022 Fall) worked on the overall website. The website uses HTML, Javascript, CSS, and Node.js with Express apps for most of its API calls and back-end, as well as FormKit for user input and Axios for HTTP posts. They created the initial database, front-end design, and initial form fields meant for user accounts and registration. Their main focus was to include a registration feature into the website for parents to register their children for different afterschool or summer programs. The front-end pages they made include various about or informational pages about DECTech and includes login/user pages. With user pages, the first team also implemented a token for the accounts, meaning users could stay logged in as they went between pages. Besides this, the team also implemented ways to download roster information as a .csv file, allow users to edit their own information and update in the database, and encrypt passwords.

The second field session team (2023 Summer) worked primarily on administrative pages and updates. The team reformatted the database and added new flags for users to allow for admins, instructors, and regular users to be separated with different permissions. The instructor flag for a user isn't implemented, however, the admin and user flags are, allowing an administrator user to visit admin pages with filters and class/children management. A regular user can only edit their own data and register their own children. The admin pages for class/children management was implemented by this group, allowing the admin to access .csv files of rosters, make and manage classes, and manage user data. The team also implemented a system to email users if they requested a password change.

Being the third team to work on this project, many areas of the website need updates and changes. Recently the website was migrated to a different Mines website hosting server with Vercel for security testing. This testing expanded upon various security concerns with the website in its in-progress state. The team has done its best to address these security concerns and implement new protection where possible. Otherwise, documentation has been created to highlight still present gaps in the security as well as other new security concerns. Because the website's visuals were outdated, meanwhile, new front-end visuals were added and have been integrated properly with the website. Overall, further improvements and documentation have been a big focus for this project.

II. Functional Requirements

The main functional requirement Dr. Liebe gave us was to ensure that the current iteration of the website had good front- and back-end functionality. This would require us to go through the website and test all the features it had available to find any issues such may have. In particular, looking for any bad links, security risks, and anything else that does not work as intended. As such, our primary minimum goals are as follows—

1. ensuring registration features are correctly and securely integrated with the database.
2. running a variety of rigorous cybersecurity tests.
3. updating the front-end on the test deployment to match the currently published website.

These are the goals that we, for certain, want to complete within these five weeks, and are considered our minimum functional requirements. If there is enough time, there are also a set of secondary goals and stretch goals we seek to complete as well. The secondary objectives are as follows—

4. adding phone number collection to registration.
5. implementing a secure mailer that can automatically send emails to intended recipients.
6. ensuring no duplicate emails when creating accounts.
7. implementing a verification process, including a way to generate secure links.
8. implementing a method to reset a user's password.
9. and resolving a database issue regarding the inability for two guardians to be linked to the same child(s).

The stretch goals are also as follows—

10. making the administrator page more user-friendly.
11. researching a way to make registering a child for a class seamless with payment registration.
12. and developing training materials for future teams, developers, and/or administrators.

All of these above requirements are ranked in order by their level of priority. It is unlikely all of them will be fully completed by the end of this field session, but again, we expect the minimum requirements to be entirely done. The secondary goals come next, of which we'll complete as many as we can, and if we still have time, we will begin tackling one of the stretch goals and likely leave whatever work remains for the next team.

III. Non-Functional Requirements

Our main non-functional requirements center around user-friendly design and security alongside the basics of building our code. They are as follows—

1. the website's front-end and back-end must be written in Vue, FormKit, Axios, Javascript, and HTML.
2. the website should be easy to navigate and intuitive to use for all users.
3. the website's format should be supported by all browser versions 2021 and later.
4. the website and code must be designed abiding by the ACM Code of Ethics.
5. individual risks and the principles of information security and safety will be considered at all time.
6. all code written must be pushed to the GitHub repository by EOD July 16th (07/16) for deployment.
7. and all code developed will have appropriate commenting and documentation.

IV. Risks

i. Skill Risks:
Learning or having a familiarity with Node.js, Vue, FormKit, Axios, HTML, and additional libraries and packages is necessary for the project. Some packages or libraries are outdated as well, so finding and learning new libraries is needed. Further familiarity with PostgreSQL database and general website design or development will prevent any issues with database calls. Familiarizing ourselves with the project's various coding languages, libraries, and files will be helpful to our efficiency on the project and ensure features remain with fewer bugs.

ii. Organization:
The files used for the project contain two previous semesters of field session work. Ensuring organization is maintained and functions, files, and other items aren't broken or lost while editing the

project will prevent other technical issues. Additionally, adding documentation, comments, and notes to areas of the codebase is needed to uphold the quality of code and ensure future authors of the codebase can quickly and efficiently understand files, locations, and other features of the website.

iii. Security/Technical Risks:
The website involves the use and collection of parent and child information. Ensuring that this data is kept protected and inaccessible to unauthorized users will prevent data from falling into the wrong hands. As detailed in a security audit, the website has a number of vulnerabilities, such as cross-site-scripting (XSS), clickjacking, man-in-the-middle-attacks, and more [Appendix A]. Malicious users or attackers could act upon these vulnerabilities and obtain or cause the involuntary release of sensitive parent and child information. Such a release would not look good for the DECTech team or the School of Mines as a whole.

While developing new web pages, or updating previous web pages, current security measures could be changed or reduced. If new functionality is added, new vulnerabilities could also arise. Ensuring security remains and is updated with new functionality and web pages is necessary for upkeeping protection.

V. Definition of Done

The list of minimum useful feature sets for this project is to have a website that has proper front and back-end functionality that can be used by consumers and admins with minimal issue. In particular, there should be a fully functioning registration page that processes data safely and securely and can be deployed. The client should be able to test the website's ability to prevent current cyber attacks such as SQL and Javascript injections, ensure the website doesn't have any improper links, and check that the website properly saves the correct information.

As per our secondary and stretch-goals, each added feature or functionality will be considered its own benchmark and will be considered done if the feature can be deployed. As each feature is added, the website must maintain full functionality and appearance, meaning each new feature will be implemented both from a back-end perspective and a front-end one. Due to time constraints and other issues, secondary and stretch-goals were instead pushed into documentation.

Finally once we have completed all the minimum useful features and any secondary or stretch goals we will deliver our product to our client. The way the product will be delivered is by merging our GitHub branch with main.

VI. System Architecture

i. General Architecture

A user can access and interact with the web page in a web browser. The front-end uses HTML, CSS, Vue, and FormKit, with the latter allowing users to input data into form fields. These form fields are used to collect user's inputs and act with Express.js, which serves as a middleware, before updating in the database. FormKit is used on the front-end to allow users to make accounts, add children to their accounts, register their children for classes, and update their account information. Users can also use various pages on the website to update or view previous information, then having the back-end Express.js take the data from the database and pass it to the front-end.

ii. User Flow/Process

Figure 1 is the flow diagram and shows the work of the two previous teams [2][3] and how we made sense of where and how data is moved in the project. It includes both admins and parents/guardians as users. The diagram provides a high-level look at the website, starting the user experience at the registration page on the left ("User loads DECTech"). Once loading into the DECTech website and going to registration, they have the choice to either log in or sign up. When the user signs up with a DECTech website account they input their name, email, and password. When the user logs into their DECTech account they gain access to (1) adding a child to their account, (2) being able to edit their existing children's information, and (3) being able to sign their children up for classes. Each of these functions update different parts of our database with this new information.

There is also the possibility for an existing admin to make an account an admin account. With an admin account, not only do you have access to all the features listed above, but you also have some new ones. Admins have the ability to add and edit all children, users, and classes from the DECTech database. They will also be able to pull users, child, child in class, and class data from the database.

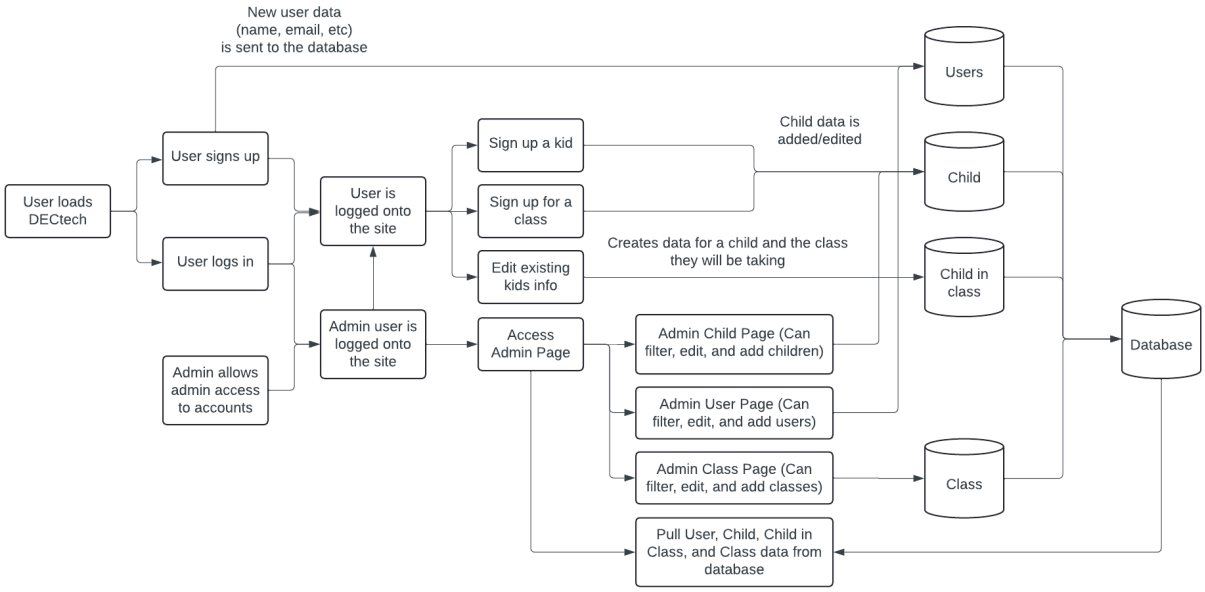


Figure 1 - DECTech website user experience flowchart

iii. Database Architecture

Figure 2 shows the database ERD [3]. The database was originally made by the first DECTech team, and then modified by the second. Written for PSQL, the database establishes entities such as the children, users (parents), and classes. The 2023 team adjusted the database by adding user tags for different permissions. The user entity now has a parent, admin, and instructor class. The parent tag is used for users, allowing a parent user on the website to create, add children, and register them for classes. The admin tag allows administrators to access and manipulate data on the website. The admin can also add and edit classes for registration, which then show up as registrable classes for parents.

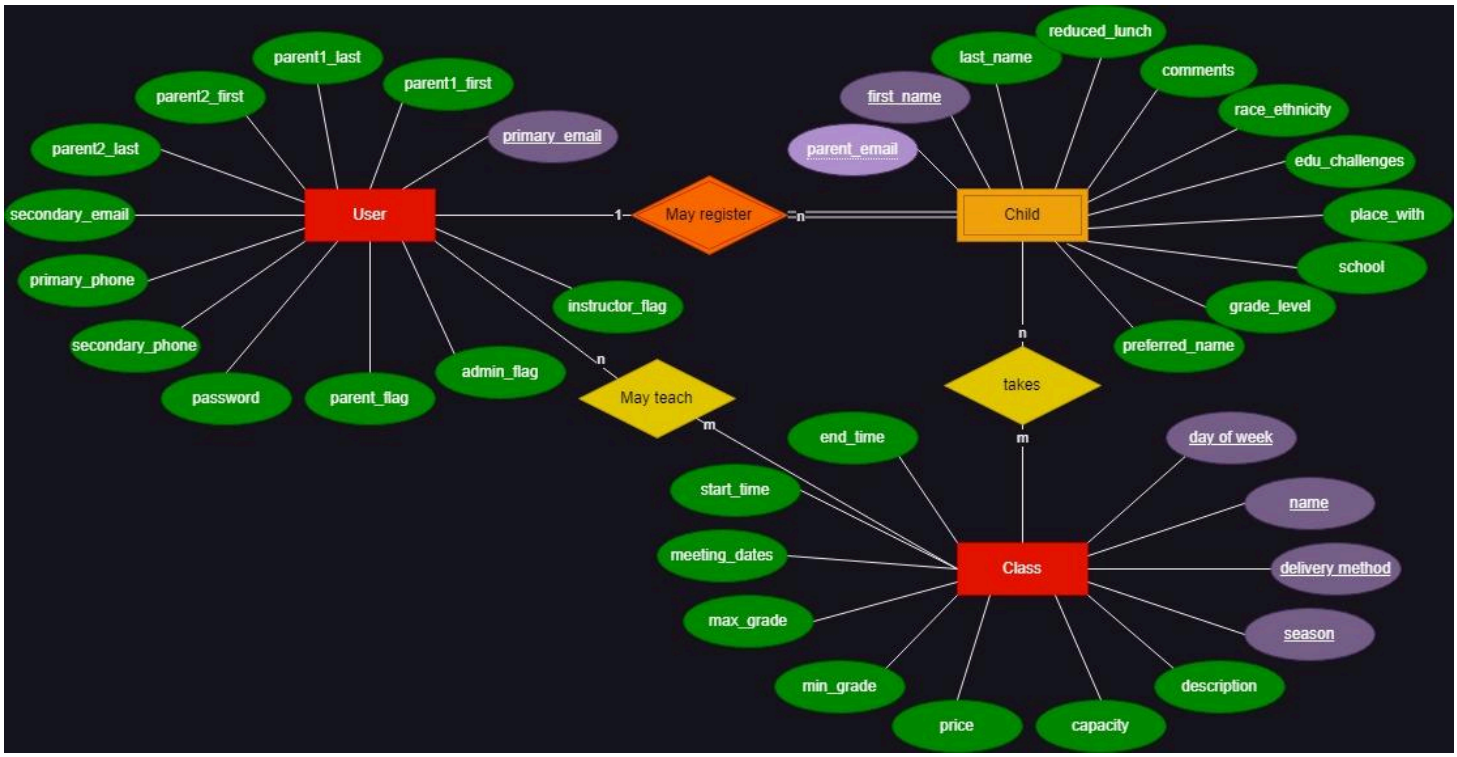


Figure 2 - ERD of the DECTech Schema

iv. Other Architecture

The previous team also implemented an automatic emailing system [3]. The system promptly sends an email when a user requests or performs certain actions on the website, such as requesting a password reset or when successful registration of their child occurs. The emails are sent through a stem-tech email address distribution group for professionalism.

VII. Technical Design

i. Validation

Each input field on the DECTech website is using FormKit, a comprehensive form builder. We are utilizing FormKit to make our input fields more user-friendly for an overall better user experience for our website. FormKit has a number of different form types, such as email, numbers, and text. Some of these form types, like email, have good built in protection, but basic text fields, the one we use the most, do not. This becomes an issue when malicious actors abuse this vulnerability via XSS injection. A data breach is the last thing we want, which led to us creating our own custom FormKit validation.

The way we do this is with the validation field in FormKit. This validation field makes it so user inputs must follow rules that are noted in the back-end before they are able to be submitted. Since we have a lot of text fields that require specific inputs, we can't use basic validation syntax, so we need to make our own using RegEx [Appendix A]. In the end, we made three: one for names, one for classes, and one for catching XSS characters.

The website contains many areas where a user adds a name, so we needed to make a comprehensive RegEx that allows for people to properly add their name but, at the same time, prevent characters not found in a name. We wanted the fields to be able to allow all letters that can be used to make a name as well as a few

symbols like hyphens and apostrophes. At the same time, we needed to prevent form submission of symbols not found in names, including carats, angle brackets, at-signs, slashes, and the usage of numbers. With these constraints, we created the RegEx and labeled it as name_validation.

Next we wanted a RegEx for entering the name of a class and or a school. This one was easier to implement, as we wanted it to keep most of the functionality of the name_validation but also include numbers, as classes had numbers in the name and some schools do as well. Thus, borrowing from name_validation, we then created school_class_validation.

Finally we wanted a RegEx to simply prevent any XSS injection characters, as some input fields are comments and we don't have any major restriction on those input fields. For this RegEx, we allowed everything except the angle brackets. It should be noted that this is a band-aid fix; we could not figure out how to implement HTML encoding, which would act as another (better) layer for preventing XSS, but for the time being, this is still a solid solution for the current version.

After we had made each RegEx, we made three different files for each one that declared what they were before importing them into main.js and calling them in the FormKit files.

ii. Registration Wireframe

Figure 3 shows a wireframe for the registration page. The main intent for this wireframe was to integrate the newly implemented database connections from last year's team while maintaining the look of the current website; it would replace a Google Form link, which is the website's current method of registration. Until cybersecurity concerns within the database are addressed, however, the functionality this registration page relies on remains insecure, and so the page's use (and wireframe as well) remains unimplemented.

This wireframe shows an updated user interface for the website, which would combine multiple pages together to keep registration as simple as possible. The wireframe also includes a login button at the top right corner, allowing users to return back to a login page. The login button and the pooled registration page would implement the aforementioned new functionality with the database.

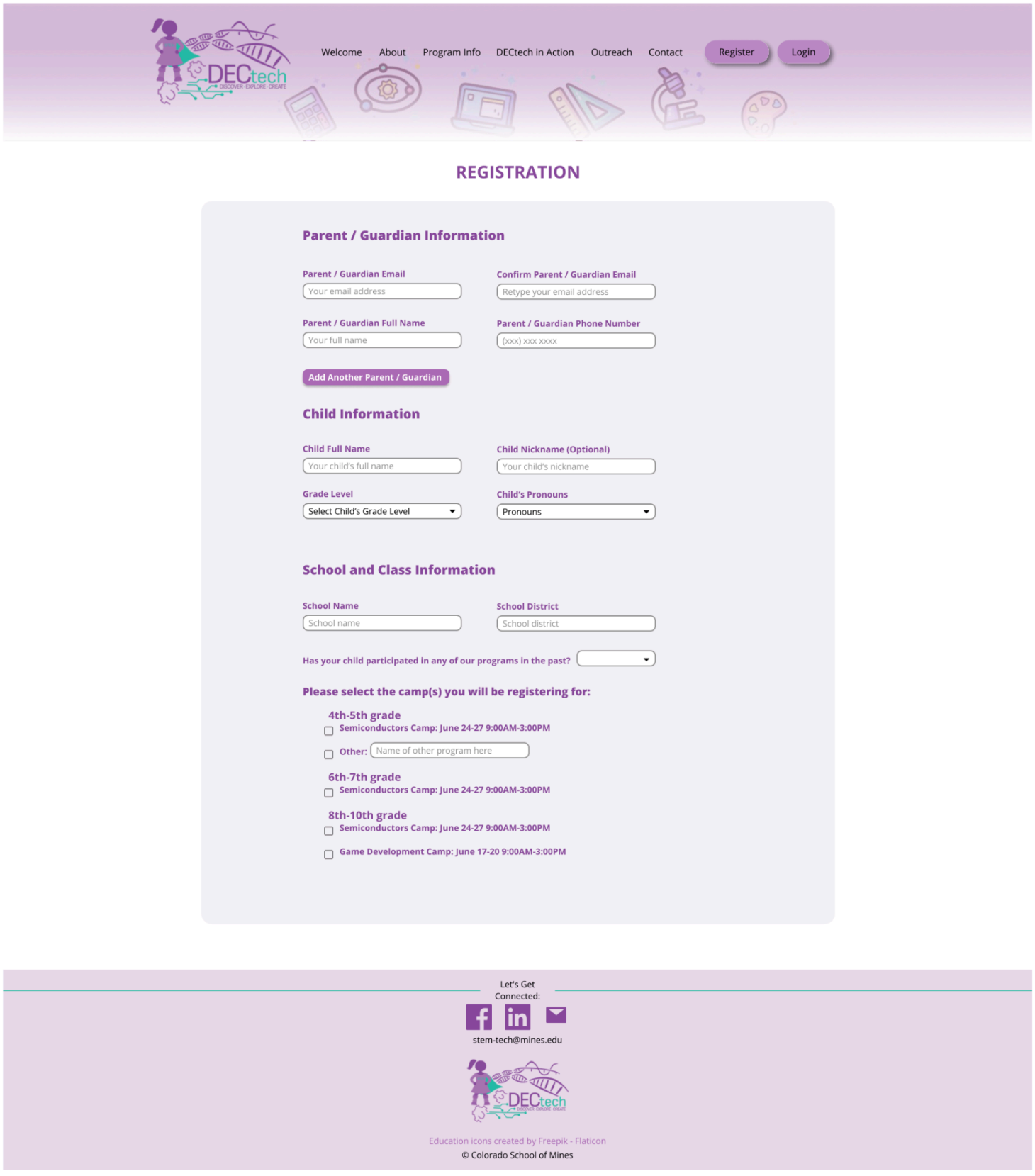


Figure 3 - DECTech registration wireframe page

iii. 404 Error Page

An issue we found with the DECTech website is that whenever you enter a wrong URL in the site, the full file path is copied into the application's response. For instance, if we entered summer/badpage.html, then the response would be <pre>Cannot GET /summer/badpage.html</pre>. This inherently isn't an issue, but these URLs are a way malicious actors are able to act upon XSS injection and clickjacking to do harm, something we want to avoid. To prevent this, we have created a custom 404 page for our website. The way we achieved this was first creating a custom 404.html page as seen in Figure 4. From there, we added code to the serve.js file [Appendix B] that would catch when a 404 error occurred and redirect the user to this page. With this page, not only do we make the website more user friendly, but we also make it more secure.

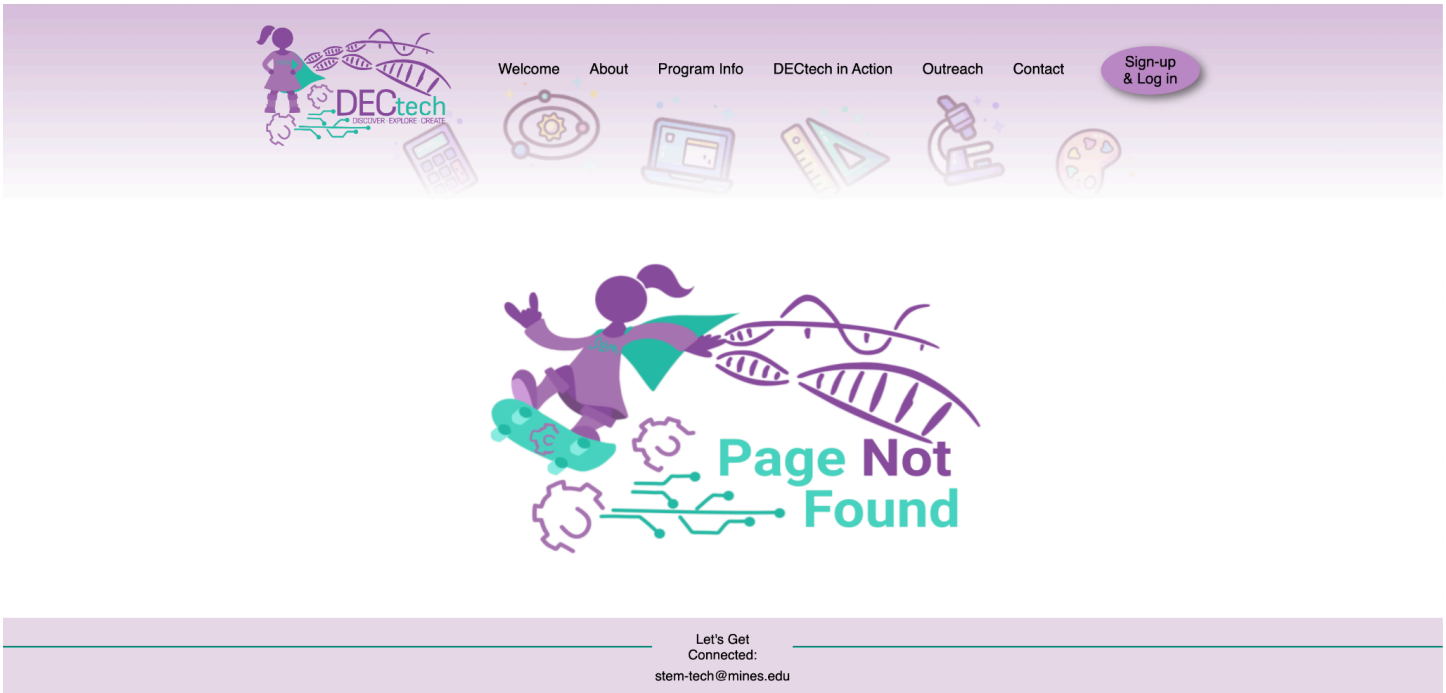


Figure 4 - DECTech 404 page

iv. Front-end Updates

Our biggest change was to move the site over to the more modern design that's present on the published website and to fix any bugs that emerged as a result. This ended up being easier than we'd expected, at least in terms of updating the design, but some of the bugs proved more difficult to deal with than we first thought.

The first major bug we discovered was that the image carousel on the welcome page was not properly deleting images that should have faded away from the left side. This caused their drop shadows to build up on top of each other and generally became visually disruptive. Along with this, the carousel also had an image of a clay snail that would overwrite other images with copies of itself as the carousel moved along. If the site ran for more than a few minutes, all these problems would culminate to cause severe lag, so it was a high priority for us to fix this "snail problem". While some visual issues still remain with the carousel, the snail has been defeated, and the remaining issues are difficult to notice unless actively looking for them.

The second major bug encountered was with image and text scaling to screen resolution. At many common resolutions, the carousel would be placed on top of the text, making it unreadable, so we had to implement a new breakpoint where just before reaching a resolution where this issue would occur with scaling, the images would change position to be above the text rather than directly on top of it. Due to extremely minimal documentation in the page prior to fixing this issue, finding the relevant section of the HTML proved quite difficult, taking much longer than expected, but the issue has been fixed.

After our work on the front-end, all pages looked much more modern, were cleaned of irrelevant information, and were all around easier to interact with than the previous site.

VIII. Non-Technical Design

i. Library Updates

Going through multiple semesters, some libraries needed to be updated to upkeep security and usability. jQuery, used in our front-end, needed to be updated and was relatively easy to do so. Other libraries, such as Node.js and Express.js, might be deprecated for Vercel. These libraries are not as easy to update due to the project having such a large dependency on them. Due to time constraints and workload, these two libraries, among others, were not updated but instead documented as parts of future-proofing the website.

ii. Documentation

With little documentation to guide our team, our start to this project was very rough. On top of that, obtaining access to files, previous work, and other information became another difficulty. After how much we struggled trying to gain an understanding of the project and how it works, our team set out to develop more documentation in order to better prepare whoever works on it next. Our delivery includes further documentation of the project, as a separate document alongside the delivered files, describing how to get the project to initially run, an overview of all files, and a list of future implementations or improvements as well as current cybersecurity gaps in the project. We hope that this nine-page document helps future developers to get started with the project quickly.

IX. Software Test and Quality

In approaching quality assurance in our product, we have developed a number of tests displayed below (Figure 5). In each of these, we will be able to show how our product meets both requirements (defined in Section II) and a professional standard of quality. If all of these tests pass, then it shows that the features tested are able to be delivered to our client.

Test Name *& Category*	**Environment**	**Setup**	**Action**	**Expected Result**
Form sanitization *Cybersecurity*	Form input field (registration, adding child, adding class, etc.)	Look through all pages that have input fields and determine tests for each one	Manually test each form for script input injections or invalid data types/symbols	Form inputs only accept reasonable information
Broken Links *Front-End*	HTML	Go to locally hosted website	Manually go through all pages and test each link	All hyperlinks across website are fixed and send user to correct web page
Front-end changes *Front-End*	HTML, website itself	Go to locally hosted website	Manually check front-end changes after updating (changing size of windows, checking every link, testing website features, etc)	All form fields and other CSS or interactive pieces on site work correctly. Looks correct (pictures, GUI, etc.) regardless of screen resolution
User Experience *Front-End*	HTML, website itself	Have website being hosted locally	Have people (not from our group) test our website and see their user experience.	All testers enjoyed their experience with our website and didn't find any issues when using it.

Figure 5 - Test outline for various aspects of the website

i. Form Sanitization

File name	Form Field	Acceptable values	Description of acceptable values
adminChild.vue	filter	abd, -, '	With filtering children the user should only be able to input the same values as they would be for a name
adminChild.vue	First Name	abc, -, '	With names letters are needed, some names also include unique characters like - and '
adminChild.vue	Last Name	abc, -, '	With names letters are needed, some names also include unique characters like - and '

Figure 6 - example snippet of form sanitization fields

For form sanitization, all fields with an input on the website use FormKit. FormKit works with the framework Vue to provide easily fillable forms for users on the website to input names, age, and other information. Malicious data can also be inserted into these forms as the same kind of text inputs. To prevent this, a layer of security is added through FormKit's validation feature, allowing form fields to whitelist characters. FormKit's validation features include a number of presets to act like regular expressions, only accepting particular characters or symbols.

Figure 6 shows how whitelisting is enabled on the website's FormKit. This is only a snippet for one of many files on the website. The whitelisting needed careful considerations in order to properly allow users to register. An example of this is only allowing numbers to be used in a field for a phone number. These fields were tested with acceptable and unacceptable characters, with the main aim at preventing Javascript injections and XSS.

ii. Broken Links

The front-end of DECTech's test website contained a lot of missing and outdated information, many of which are hyperlinks between pages. In our efforts to update the front-end of this test website using the look of the published website, we scrapped a number of pages and mended any broken links that exist afterward. Once that was accomplished, we made a list of each link on each HTML file before going to the locally-hosted website and testing them one by one. This ensured there would be no more broken links and that our front-end worked cleanly and as intended.

iii. Front-End Changes

Broken Link Tests	Works?	Quality Tests	Works?	Specific Tests	Works?
Welcome					
About	Yes	Do all links work?	Yes	Does the carousel work? (wait 15 min)	Yes
Afterschool/Summer	Yes	Can the webpage be scaled well?	No	- Does the page run without slowing down?	Yes
In Action	Yes	Are the aesthetics updated?	Yes	- Repeating images? (the Snail Problem)	Yes
Outreach	Yes	Is the webpage easy to navigate?	Yes	- Do images clip?	Yes
Contact	Yes	Is all text readable?	Yes		
Register Button	Yes	Is it usable across all common browsers?	Yes		
Reviews	Yes				

Figure 7 - Criteria for front-end changes

Testing our main front-end changes is largely subjective, but still follows a list of criteria we expect it to meet. These criteria, for the most part, follow two questions: does it work, and does it look good? These two questions are rather vague, so the criteria narrow them both down into definite, qualitative features that our changes should accomplish. Pictured above (Figure 7) are some of the specific criteria we had created, specifically showing those for the welcome page.

iv. User Experience

This section fell outside of the scope of our testing, but for future testing, it would be ideal to bring in outside users to test the site's functionality. Account setup and registration features should be tested in particular, as it's possible there is some element in this functionality that could be used in a way we hadn't predicted or figured and could not find on our own. In these tests, users would be asked to make an account and register a hypothetical child for a class, but not given an outline of the necessary steps to do so; this way, we could see if the process is intuitive or not (make account -> add child -> register). After this testing, we'd want to see if the functionality for editing child and parent information behaves as expected, so we would test that as well, along with conducting similar tests on admin features.

After all of these tests, we would have a much better understanding of how users engage with the site; such would give us a better opportunity to adjust features to them more intuitive and also fix anything unintended.

X. Project Ethical Considerations

i. Privacy and Security

ACM 1.6: "Respect Privacy"

Knowing that this website is meant for parents of students, user accounts store information such as emails, phone numbers, names, etc. for their own account and their children's. This information falls under FERPA information (as children are registered as students), and as such, is legally classified as private information. Making sure personal information is private is necessary to prevent its use for illegitimate reasons. ACM 1.6, on a technical perspective, has been addressed by previous teams that have worked on this project, especially when it comes to data only being accessed by admins or authorized individuals. However, data must also be protected from unauthorized use and potentially malicious actors, so vulnerabilities **must** be addressed.

ACM 2.9: "Design and implement systems that are robustly and usably secure"

With the website potentially being a public site, certain precautions are needed to ensure user data isn't compromised. Misuse of the system needs to be taken into account when reworking previous systems that did not include enough security precautions. When threats arise, the deployed and in-development systems must be updated to mitigate harm. Known vulnerabilities should be addressed or documented in reports in order to keep up with future threat and transfer of code. Additionally, the system must remain usable for its intended purpose to prevent clunky or confusing implementations. The system should be able to remain robustly secure while upkeep accessibility and diversity, such as through name accents.

ii. Ensuring Product Quality

ACM 2.1: "Strive to achieve high quality"

With a codebase that has previously been constructed by other colleagues, maintaining the quality of the codebase while constructively addressing areas where improper upkeep has occurred is a principal focus. The codebase and supporting files need to be well documented to prevent delays, confusion, and gaps in the product's implementation. As it regards our group, this is a main focus due to much of our own time on the project being used to catch up and understand the codebase. Well documented code is a reflection of high quality work and this quality is needed in order to continue the project with future teams who work on it. Documentation of missing features, bugs, and lingering vulnerabilities or threats are needed.

This also applies to transparent communication to those involved in the project. While working requirements are goals for a project, communicating when or if these requirements need to be reconsidered or evaluated is necessary for transparency with a client. Additionally, communicating roadblocks and other issues such as vulnerability concerns in a timely manner to a stakeholder can help to mitigate negative consequences.

ACM 2.6: "Perform work only in areas of competence"

Work surrounding the project should be judged for feasibility and if the work falls into the team's area(s) of competence. It is our job to assess what should be worked on and if we have the ability to make what is worked on something of high quality. Ensuring features that are not fully implemented and tested enter the published version falls on us and should be prevented. Additionally, it is our responsibility to ensure that a stakeholder knows when we don't have necessary expertise and that, if a stakeholder continues with the product outside of the additional time, they have the resources or information to acquire other competencies.

XI. Project Completion Status & Results

The goal of this project was to work on the new DECTech database website by ensuring that registration and database integration functioned properly, by implementing as many security features as possible, and by updating the front-end. With access to the HTML files of DECTech's modern looking website, our team was able to upgrade the new DECTech website to have a more user friendly front-end. With a security audit from the last field session, we had a good idea of what we needed to do for website security implementations. We were able to get a majority of the issues resolved, like updating jQuery, creating a 404 page, and fixing stylesheet paths, and we were also able to implement features to address the issues we couldn't fix, such as adding FormKit validation to prevent XSS injection.

We were not able to complete the proper integration of the registration and database due to the lack of documentation of this project. Over the course of the five weeks, we spent the first two trying to understand how our given code worked due to such a lack of documentation. This resulted in us having not enough time and experience to be able to do anything substantial with proper integration of the registration functionality and the database. Due to our lack of understanding for how the back-end truly worked, we were not able to implement some more security features like HTML encoding, which would have been a more secure fix and an additional preventative layer for XSS than what we made. We also learned about other

security risks during week four that we couldn't get to, such as improper password hashing and a lack of database back-up functionality.

We did, however, manage to fix many visual bugs present on the front-end, such as the image carousel clipping into the welcome text, a duplicating image of a snail, some links to dead pages, and issues with web page scaling for all resolutions.

Our client also wanted our team to work on some stretch goals before we were done; these goals, to state again, were to make training materials for future teams, develop a more user friendly admin page, and research a way for a seamless registration-to-payment feature. Unfortunately, we were unable to work on making the admin page more user friendly as the HTML for the page was so deeply connected to the back-end, something we barely understand due to poor documentation. Meanwhile, we were able to start working on training materials for future teams with our work on adding more documentation to the project. It isn't a guide to how everything works, but we hope our documentation will help future teams be able to understand what is going on much faster than we did. Finally, for implementing a seamless register-to-payment system, we did some research into the topic and found some potential resources that future teams can take advantage of to add this feature.

XII. Future Work

Future teams might want to focus on adding more features to the registration page and admin page. For the registration page, future teams should try to implement a phone number collection system so users wouldn't need to go back into their account and edit their information to add it in. They should also focus on adding a go-back button in the registration and sign-up forms. In addition, a major thing that needs to be added is the seamless register-to-payment system as mentioned before. We found two different options:

i. The first is the service Formsite, which is easy to implement and user-friendly, allowing for users to pay for products. The only drawback is that it is a service and does cost a monthly subscription of around $21 - $34 per month.
ii. The second option is State Knowledge Base, which seems to be more difficult to implement but may be a good secondary option if Formsite doesn't work.

With the admin page, an important feature to add would be cleaner interfaces and web page elements to make it more user-friendly, as well as possible pages for constructing schedules and rosters. Additionally, it would be a good idea to do some more UI testing. The Protractor test framework or Selenium WebDriver were recommended for user interface testing. Exploring these applications might help to create a solid front-end.

The cybersecurity side of the website is something that still needs to be worked on in the future, even with the additions we made for it. The website, in its current state, is still susceptible to spam or click spam attacks. Finding a good way to prevent these types of attacks should be a future concern. Another issue is that the website, in its current form, has no system for creating back-ups. On the off chance that records from the database are somehow deleted, there is no way of recovering it, which could become a major issue if large amounts of the data are deleted by accident or after an attack. Meanwhile, as mentioned before, the passwords are not hashed properly; this is because they are moved across the network to the server before being hashed. This opens the website to man-in-the-middle attacks, where malicious actors could steal user passwords, resulting in a massive security breach. As for data sanitization, none of our FormKit fields have HTML encoding and other layers to prevent XSS. Future teams should look into all of these issues, as well as researching a secure layer certificate (SSL) (which might be added when the server is deployed by ITS) and adding Helmet.js to configure security HTTP headers. Additional security assessments should be conducted outside of problems listed here.

Another component for future work could be adding tutorials for administrators to develop schedules or change and edit information. Although we are making documentation for future developers, there should

be some documentation for the admin users (aka, DECTech team instructors) so they get a better understanding of how everything works.

Regarding front-end changes, we've noticed a few small visual fixes that need to be made but that we didn't have time to implement. In particular, the header of the welcome page, an image in the about page, some text in the afterschool programs page, and the registration form all have slight visual issues at various mobile screen resolutions, where they either clip into other page elements or scale in an unexpected way. We don't expect these will be very hard to fix, but they don't fall within our current scope.

We have done our best to add as much documentation as we could. The project's Google Drive link will also include a future team notes document, which includes notes about the project, high-level overviews of all of the files, and how to set up and use the website locally for testing and development.

XIII. Lessons Learned

Our group has learned a great deal about working with a pre-existing website code base, namely back-end libraries such as Node.js, Express apps, Vue, and FormKit. We've gathered a general understanding of each of these libraries through working on this project, having had a minimal understanding of them to begin with and learning mostly through self-exploration. Cybersecurity with private data has been an interesting point in this project as well, increasing our prior knowledge of preventing attacks through the careful use of coding techniques and system architecture. Additionally, our team has learned of the complications of working with previously developed codebases, as improper documentation prevents future developers from working at full capacity. Adding documentation helps other developers, whether present or future, to quickly grasp what is needed and how they might start working; most of all, it provides necessary information that might not be seen or can be easily glossed over when purely reading the code. Finally, one last lesson we've all come to realize is the slowness of communication by email, which is something that can't exactly be fixed, but can still be accounted for when planning work. Communication isn't something to be avoided, so doing all one can when just writing a usual email makes all the difference.

XIV. Acknowledgments

Amia Castro - Previous work on HTML files

Mike Diplock - IT contact for Vercel deployment

Addison Hart - Access to mono github for HTML files

J. Jackson - Illustrated our team logo

Professor Kathleen Kelly - Clementines and help with getting access to the mono repository

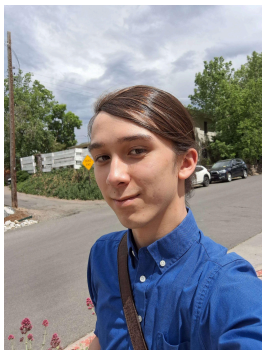
Dr. Christine Liebe - Client, help with tracking files, people, and more

Bodie Lutz - DECTech lead TA and additional point of contact

Dr. Phil Romig - Help with identifying further security issues and tracking back-end functions

XV. Team Profile

2024 Team



Curiosity Steen

Computer Science - Data Science track; Applied Mathematics and Statistics
Minor in Public Affairs
Hometown: Cedar Rapids, IA
Experience: Administrative Assistant @ UHSP House
Coding Languages: HTML/CSS, Python, Java, C++, R, MATLAB
Activities: oSTEM, McBride Honors, creative writing, tennis, video games, being queer
"Team building exercise: you're getting married."

John Jagger

Computer Science - General track
Hometown: Pueblo, CO
Experience: HIVE editor
Coding Languages:, Java, C++, Python
Activities: Video editing, Swimming, Destiny-ing
“Bazinga.”

Nathaniel Garner

Computer Science - General track
Hometown: Memphis, TN
Experience: Summer API Distribution Internship, front-end development
Coding Languages: C++, HTML/CSS, Javascript, Python
Activities: Video games, creative writing, eating aquarium gravel, pondering the orb
“Yeah, I can find him.”

William Wheaton

Computer Science - Computer Engineering track; Pursuing Master of Science in STEM Education
Hometown: Golden, CO
Experience: DECTech Teaching assistant, Summer IT internship, various volunteering
Coding Languages: C++, Python, Java
Activities: Dungeon Master for Dungeons and Dragons, video games, lore reader
“I just want an orange.”

2023 Team: Braden Gehr, Trevor Hartshorn, Micha Munoz, Tyler Rotello
2022 Team: Anna Pitcock, Audric Wang, Audrey Powers, Natasha Bailey

References

[1] Extra graphics and team logo (cover page) by J Jackson.

[2] N. Bailey, A. Pitcock, A. Powers, and A. Wang, “CSCI 370 Final report, DECTechTives,” Dec. 2022.

[3] B. Gehr, T. Hartshorn, M. Munoz, and T. Rotello, “CSCI 370 Final Report, DECTechnicians,” Jun. 16, 2023.

[4] “ACM Code of Ethics and Professional Conduct,” Code of Ethics, https://www.acm.org/code-of-ethics.

Appendix A – Key Terms

Descriptions of technical terms, abbreviations and acronyms.

Term	Definition
Node.js	*An event-driven javascript runtime for building network applications. Main engine for the DECTech website project web page.*
Axios	*HTML client for Node.js, driving server calls for various things like the database.*
Vue	*Javascript interface for user interactions*
FormKit	*Javascript interface for user inputs. Works in tangent with Vue for form fields on the website.*
Vercel	*Server-hosting service company; what we’re using to run test site deployment*
Cross-site-scripting (XSS)	*Cross-site-scripting, a type of cyber attack where an attacker inputs malicious code into an input field and the application reads the input as code to run.*
Clickjacking	*A type of attack where malicious overlays can be seen on top of web page objects and once interacted with redirect users to potentially harmful sites.*
Malicious actor	*Individuals or groups that intentionally cause harm to digital devices or systems.*
Middle-Man-Attack	*A cyberattack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other, as the attacker has inserted themselves between the two user parties.*

Term	Definition
Validation/whitelisting	*FormKit's method for sanitizing data inputs and checking the accuracy and quality of source data; as it regards this project, checking the accuracy of user inputted data.*
ERD	*Entity Relationship Diagram, a representation of relationship between objects and connected tags or characteristics, mostly used for database representations.*
PSQL (PostgreSQL)	*Database management system using SQL or Structured Query Language.*
RegEx	*Short for Regular Expression; a sequence of characters used to match specific text patterns*

Appendix B - Components/File List

High level description of files present in the project. Added for future team clarity. Additional information is provided in "Future Team Notes," in the delivered files for the next group of developers.

Custom validation rules: Used to validate/prevent submission of improper data to the website.

name_validation.js: validates a name, only allowing accents on regular characters (abc…ABC…) and punctuation such as -',..

school_validation.js: Like the name_validation.js above, but accepts numbers.

css_validation.js: should be "XSS", just checks for Javascript carats (<>).

Previously made files:

App.vue: This holds all of the components we have made for this project. The components are pages that we've created. All of these pages run with the "register.html" file from the original website. The App.vue file handles all of the page changes as well.

database.js: This file creates a PSQL database on your local system. This file will probably change once we hook up the connection to the Mines database and servers. UPDATED: This file is where you put in your PSQL superuser login info. This file also has all of the tables/query functions in them. This is the file that you will call from the other .js files to access/update anything in the database.

main.js: This file creates the app using the App.vue file and mounts it in the register.html file. 2024: Edited to include FormKit custom rules mounting for validation.

serve.js: This file handles all of the API calls from the website. It makes sure we can send the correct data to our database, and makes sure that data actually gets delivered to the database, by checking for response codes.

userAccounts.js: Handles all the adding users into the database. It also verifies a user's credentials if they're trying to sign in. It also generates tokens and hashes passwords.

Assets folder: The assets folder holds any images or files we need to call to insert anything into our website. For example, we have two images in this directory for the two pictures we have on the landing page if a user isn't signed into an account (where they can choose to create an account or log into an existing one).

Components folder: The components folder holds all of the pages we have created for the website. These files are as follows:

childInfoInput.vue: This page is for adding a child to a parent's account. This page is included in the account page once a user is logged in. Once a child's information is all filled out, this page also handles the call to the database to either insert or update the information.

EditChildren.vue: This component is incorporated in the account page. This data will be auto filled in if the database has some of the child's information already. If the parent changes or inserts new data, the database will also be updated to account for those changes.

loggedInPage.vue: The logged in page is where the website goes once a user has logged into their account or created one. So this will act as their account page once logged in. [UPDATED] It grants access to account management, as well as guardian or admin related pages depending on flags.

SignupForm.vue: The sign up form is the page the user gets taken to if they click "create an account" from the registerHome page.

registerHome.vue: This page is the first page the user will land on when the register.html page loads, if they are not already logged in. This page gives the user two options: create an account, or log into an existing one.

registerForm.vue: registerForm is the form the parent will fill out to actually register their child for a class. This form holds only the necessary data for the classes, since the parent and child information is held in the account page. This form will only be loaded if the parent has everything filled out in their account page.

registered.vue: This page is a confirmation for the user that the registration has completed, meaning their child has successfully been registered for a class. This page will only show if the data from registerForm has successfully loaded into the database. Another thing this page could be able to do is send an automatic email to the user as another form of confirmation of registration.

loginPage.vue: The login page is where the website goes to if the user clicks "sign in" instead of "create an account" on the registerHome page. This will ask for the parent's username and password, and if they put in the correct credentials, the site will load their account page.

admin.vue: This page is accessed when a user marked as an admin logs in. It contains links to download spreadsheets of information such as class rosters, and account information.

downloads.vue: This component has links to download .csv files of the tables in the database.

updateChildren.vue: This is a duplicate form of the one used to enter a child into the account, but is automatically populated by existing children's information and will update their details once changed and submitted.

updateParent.vue: This component allows the database to be updated with any new information the parent chooses to change or add to their account.

admin.vue: This vue is accessed through the loggedIn.vue only if the current user is an admin. This page holds a variety of buttons that are used to transfer the admin to more specialized vues, such as adminClass, adminChild, AdminUser. It also has the download.vue at the bottom so that admins can download .csv files from the database.

adminClass.vue: This vue is for admins to modify or add classes in the database. There are three buttons at the top of the vue, namely Filter, Update Class, and Add Class. The filter button makes a filter feature appear that the user

can use to alter the amount of classes shown to the admin. Update Class and Add Class both make a form for class information that can be used to add or modify classes depending on which button was pressed.

adminChild.vue: This vue is for admins to modify or add children in the database. There are three buttons at the top of the vue, namely Filter, Update Child and Add Child. The filter button makes a filter feature appear that the user can use to alter the amount of children shown to the admin. Update Child and Add Child both make a form for child information that can be used to add or modify children depending on which button was pressed.

adminUser.vue: This vue is for admins to modify or add users in the database. There are three buttons at the top of the vue, namely Filter, Update User, and Add User. The filter button makes a filter feature appear that the user can use to alter the amount of users shown to the admin. Update User and Add User both make a form for user information that can be used to add or modify users depending on which button was pressed.

resetPass.js: This file handles most of the reset password functionality. The function to actually update passwords is in userAccounts.js, but everything else should be in here.