Designing a New Course/Term Schedule-Making Website

Information Technology Senior Project Design

Term Final Report

Project Repository: https://github.com/gingebrigtsen/NMT_Schedule_Builder

By Gabriel Ingebrigtsen-Leiker

IT 482 - Mazumdar & Reinow - 5 May 2023

New Mexico Institute of Mining and Technology

NMT - Spring 2023

Table of Contents:

Project Introduction [What's the Project?]:

For my Fall 2022-Spring 2023 Senior Project Design, I designed and developed a website similar to the existing *Beanweb* site, which uses public NMT course data to present a better course lookup users can utilize to generate a term schedule and calendar. This was done on behalf of our client, the registrar's office, which needed a new platform for students and staff/faculty users. Starting last semester, I worked on researching and designing the project stack, consisting of two primary plans, A and B.

Plan A generally consisted of generating requests for the existing CourseDog-Banner Integration API to populate course information on the back-end. This ended up being essentially infeasible due to price, delays in access, and some technical difficulties with token authorization, meaning our focus first generally shifted to Plan B, before resorting to it in the end result.

Plan B generally consisted of scraping course data off the public NMT **Banweb** pages, using a Python script to maintain a comma-separated value (CSV) based course information database. This overall worked out well and uses some similar logic to the *Beanweb* site to collect the needed data on the back-end.

This data is used to populate the results of the project site's course lookup, helping users find what they're looking for more easily than the older *Beanweb* and **Banweb** sites. It also is used, in conjunction with a nice front-end Calendar module, to generate a more readable and appealing weekly calendar for users. This is all accomplished by letting users add their desired courses to a "Shopping Cart", managed as a session variable, that keeps track of the selected information.

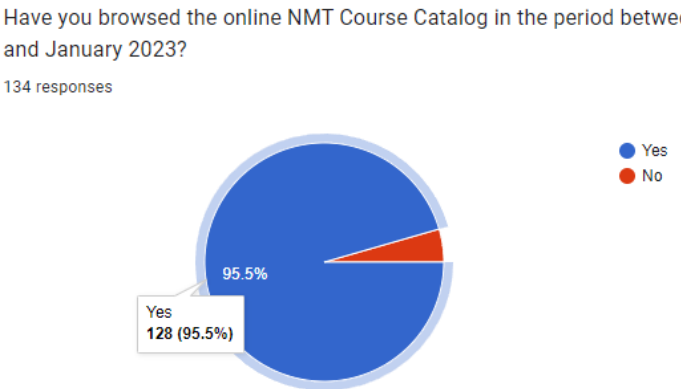
Overall, the project site is designed for a fast workflow – users select courses and add them to their cart, from where they can generate a corresponding calendar and export the schedule and CRN information. Users don't need to make an account, and the site doesn't collect *any* information from them, for the purposes of both security and simplicity in the project stack. In conclusion, my Senior Project Design terms had me designing and developing an improvement upon both the public **Banweb** course lookup and the *Beanweb* site, as was generally deemed necessary by the results of the survey conducted this Spring. The final product deliverable I've developed is my best attempt at meeting those requirements as further outlined and understood below.

***Project Purpose** [Why It Was Needed - Survey Data]:*

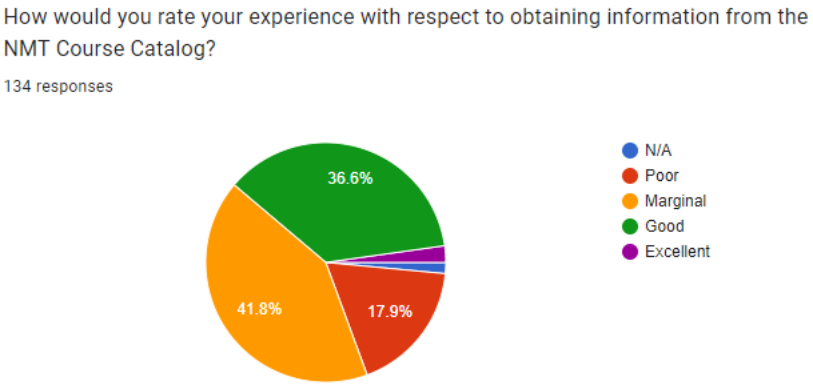
Initially, the purpose of this project was simply to improve upon the older existing platforms, as a way to offer NMT users a more modern and dynamic way to build their term schedules. The Registrar's Office needed something better that may more effectively allow students to create schedules with fewer conflicts and less required support from either their advisors or the registrar - thus lightening their workload during the busier registration seasons.

As I began the design and development phases of the project, the purpose evolved, from improving upon NMT's existing resources with my site to building an *entirely* new one, due to some of the limitations I discovered during research and preliminary development. The full purpose of the project would be to serve as a quick and easy-to-access schedule-making website for the NMT community, that offers easy-to-read data and supplementary catalog information. But, this meant that I needed to better understand what the NMT populace needs and expects from this project, in order to outline my goals and requirements.

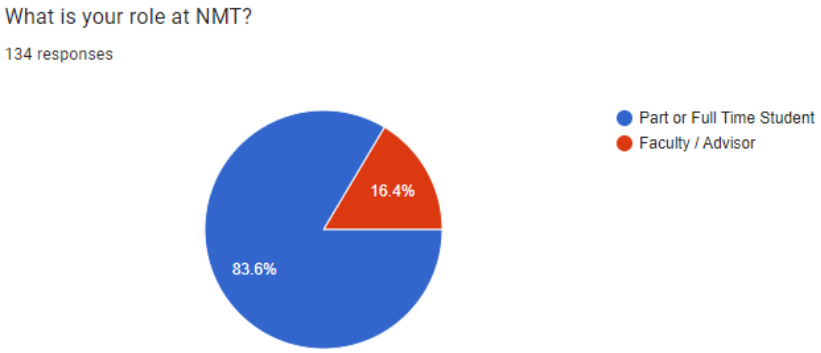
Then, accordingly, during Spring 2023 an official survey was conducted to see how existing NMT users feel about the current **Banweb** and *Beanweb* options, and this moderately altered the project to be slightly different and more advanced than the existing scheduling options. This was illuminating, and yielded the following results, providing purpose, direction, and motivation for the final product of this design project:



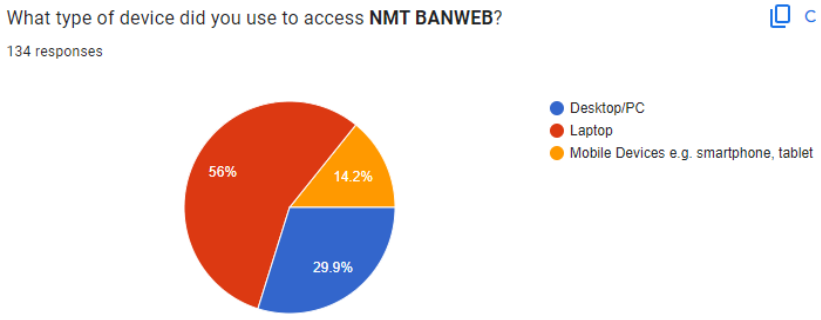
- *Figure 01: shows that a significant majority of the survey respondents, have in fact recently browsed the NMT course catalog, indicating that there's an existing need among NMT users to more easily access that course catalog information.*



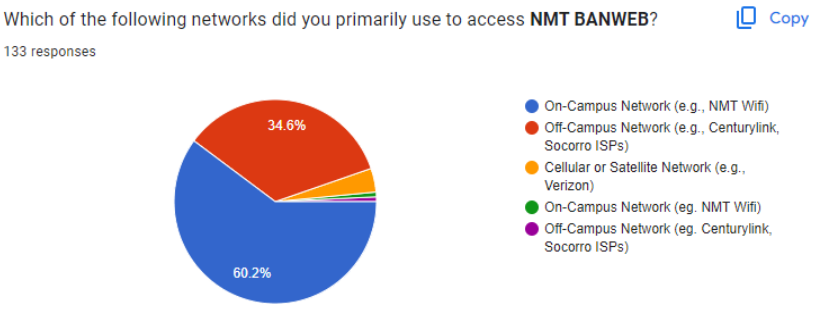
- *Figure 02: shows that a 41.8% majority of respondents had a marginal experience accessing the catalog, indicating to me that they need an easier way to find individual subjects or courses within the catalog site/document.*



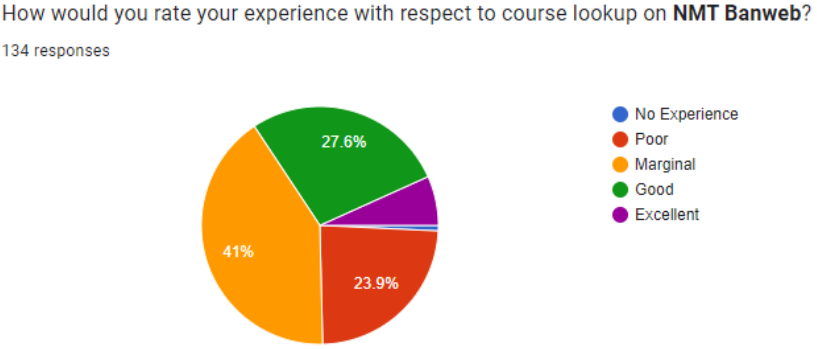
- *Figure 03: indicates an overwhelming majority of students composes the respondents to the survey, meaning that most users will have experience with registration, and be more familiar with the potential problems* **Banweb** *has.*



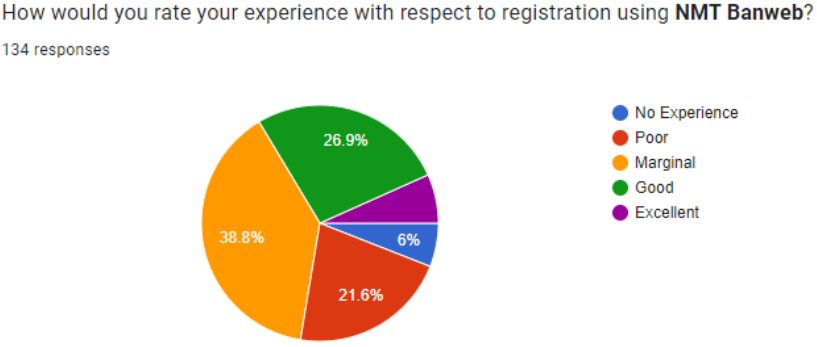
- *Figure 04: shows that a majority of users will be using a larger form factor display, and indicated to me that the site's front-end could be fairly wide and large in order to offer users a good view of all info and options.*



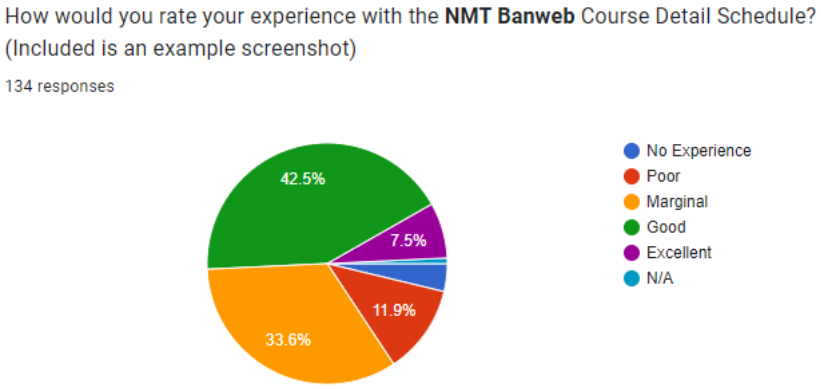
- *Figure 05: shows that a 60.2% majority of users normally use on-campus networks, and won't have to worry (as much) about being impacted by slower internet/loading speeds on the site, because I used a particularly slim front-end engine to display pages.*



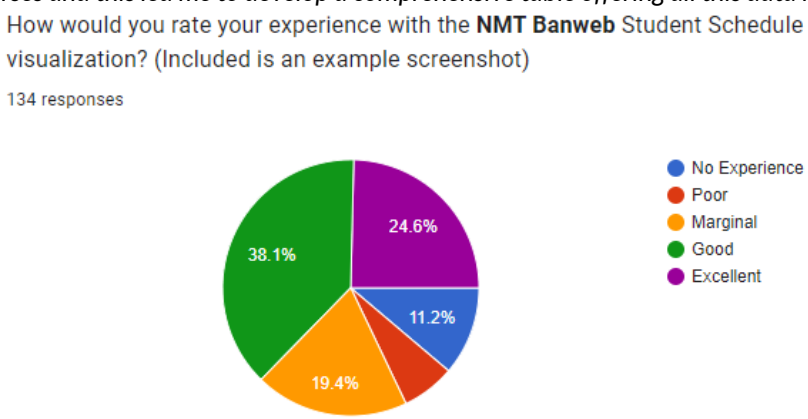
- *Figure 06: shows that a 41% majority of users had a marginal experience searching for courses on* **Banweb** *which inspired me to create a much more open course lookup allowing a global search as well as refined term, subject, and level searching. I also included the sortability of results to address this issue.*



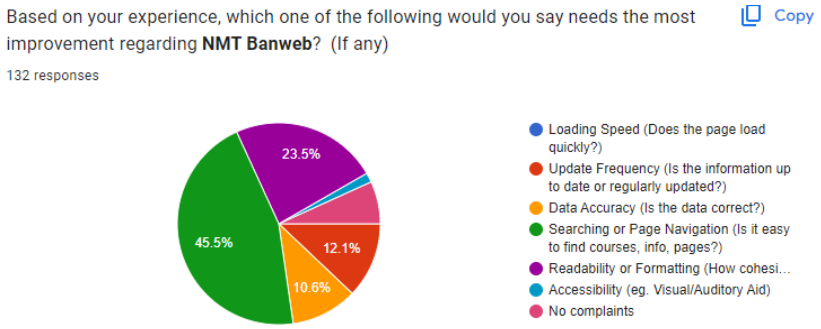
- *Figure 07: isn't particularly relevant to the scope of this project, but it indicates that the registration system itself may be in need of an overhaul, as it shows a 39% majority of users are only marginally satisfied with the registration process on* **Banweb**.



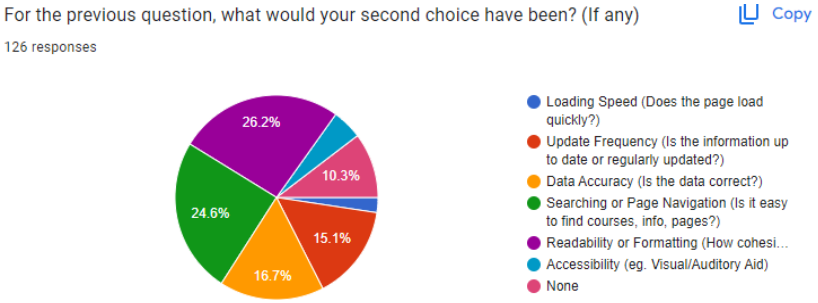
- *Figure 08: shows that a 42.5% majority of users do actually like the course detail schedule, meaning to me that fundamental data like the days, time, location, and the Instructor are a useful view to have displayed for most courses and this led me to develop a comprehensive table offering all this data in its own column.*



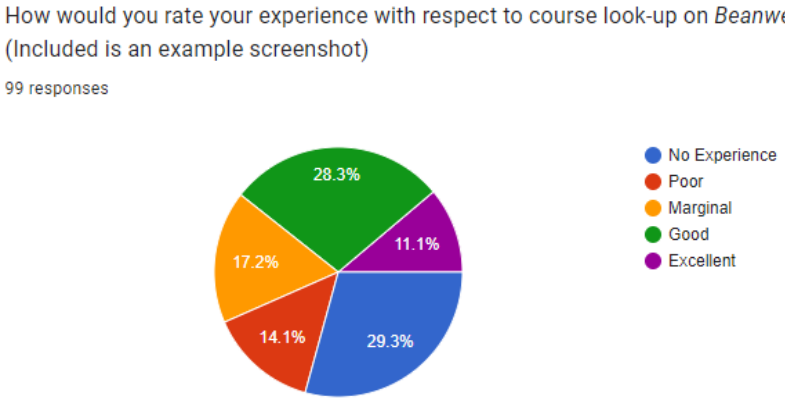
- *Figure 09: shows a relative balance between good and excellent – users usually like the existing weekly calendar view offered by Banweb. This led me to create a very similar breakdown of courses on my calendar with columns for days, and rows for hours, which allows me to more precisely show user schedules.*



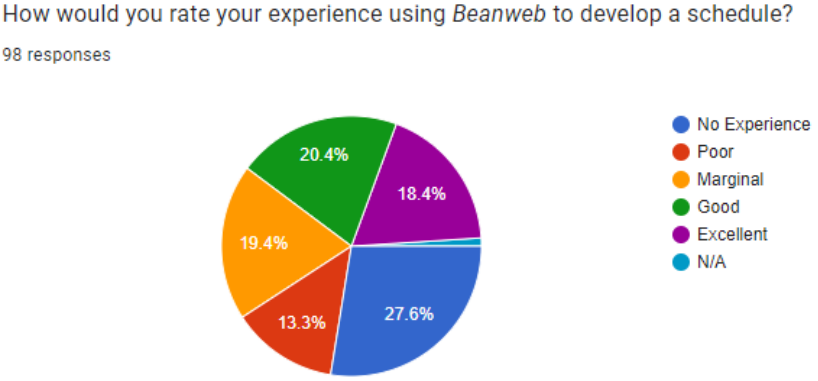
- *Figure 10: shows that a 45.5% majority of* **Banweb** *users say searching and navigating need the most improvement, and thus between not requiring any login, and a clean & clear header containing navigation links, I drastically simplified the workflow for browsing and searching for information.*



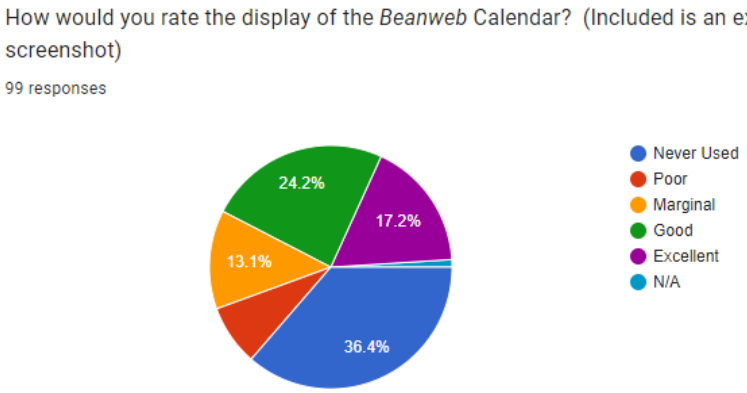
- *Figure 11: shows that users are relatively divided over other issues with the* **Banweb** *site, but approximately agree that searching, navigation, and readability/formatting are all in need of critical improvements on the new schedule platform I'm developing.*



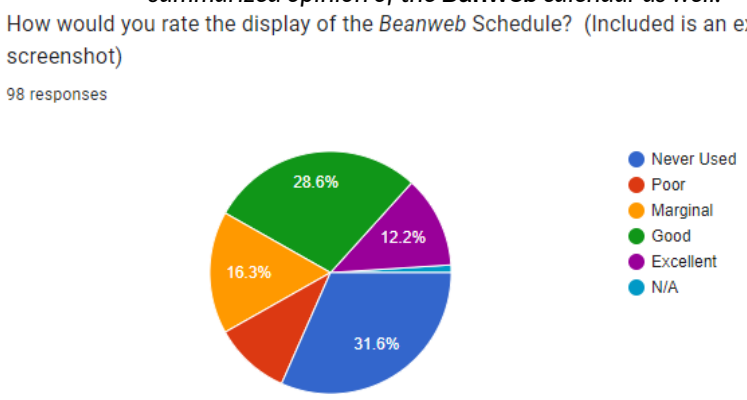
- *Figure 12: shows two things: a surprising number of respondents have no experience with Beanweb, but that a majority of those who do typically like the site. This means to me that Beanweb can be a go-to for styling or function inspiration, as some factors of it are still working well for the NMT populace.*



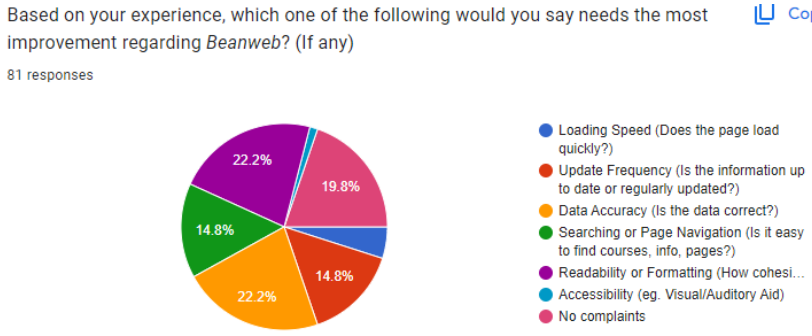
- *Figure 13: again shows that a majority of respondents currently like the structure and experience of using Beanweb to make their schedules. This indicated to me that some users might not start using the new option and that I can pull more inspiration from Beanweb for the new site, to get the approval of older users.*



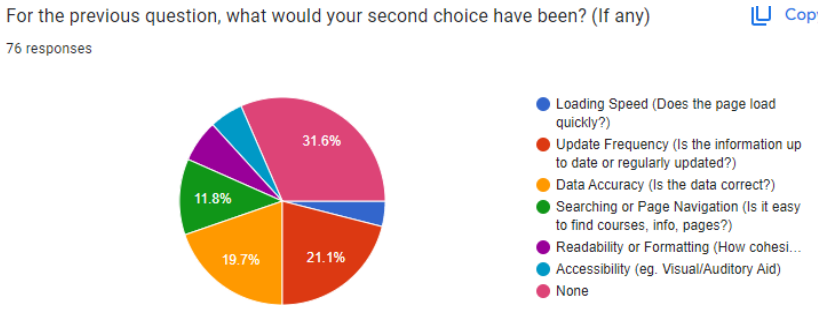
- *Figure 14: shows a vast majority of users aren't even familiar with the calendar I'm referring to, so that gives me some liberty in how mine looks and works, but also that most knowledgable users like the calendar; meaning that I should base some stylistic choices on this calendar, and blend them with the summarized opinion of the* **Banweb** *calendar as well.*



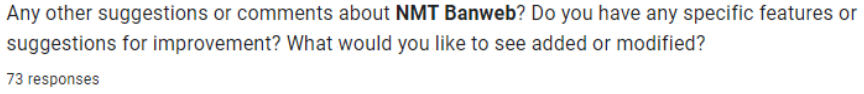
- *Figure 15: again shows a vast majority of respondents aren't familiar with the Beanweb schedule page, which indicates that I have some freedom in how I design it, though other respondents mostly said they like the existing page, so I'll be roughly matching it, with a more modernized version.*



- *Figure 16: shows the division between Beanweb users, approximately suggesting that all categories could stand for some improvement, though data accuracy and readability take precedence. I used this to ensure that my platform regularly collects all public Banner information, but selectively and intelligently displays more simplified views to ensure students can quickly and easily make their schedules.*



- *Figure 17: indicates to me that other than the issues already mentioned, they're mostly satisfied with Beanweb, I just need to be particularly wary of update frequency. This helped me decide that every 24 hours is a good balance between frequency and resource conservation for the final product.*



- *Figure 18: shows the open response question that concluded the survey was looking for any other information or opinions users might think is relevant, or other unmentioned desires for the new scheduling site. Among the most commonly given responses were ideas like: **"make time conflicts more apparent ahead of time"**, **"updated formatting, better readability"**, and **better searching** that doesn't require as many steps to complete. These were all expected based on the given content in the survey, but still yield valuable information regarding the wider perception of* **Banweb** *here at NMT.*

So, from the information collected, I was able to draw several conclusions regarding the desired styles and functionalities of the project website. The results indicated that there were 5 main categories that needed attention in order to improve the quality and accessibility of course/catalog information.

1. **Banweb**'s searching and navigating needs the most improvement.
2. **Banweb**'s readability and formatting also need attention and updating.
3. **Banweb**'s calendar view is a well-liked format but outdated and underused.
4. *Beanweb*'s data accuracy and page readability need to be improved.
5. *Beanweb*'s calendar view is a well-liked format but outdated and underused.

And using this data, I worked at incorporating solutions and responses to these primary issues into the final product over time to the best of my ability. Particularly, both respondents and myself alike wanted a much more modern-looking site, and thanks to using well-chosen color schemes, and the professional-looking Elastic UI library I feel like I achieved a look and feel as simple as **Banweb**, but much more modern, easier to read, and use quickly. Summarily, I used these responses and their opinions to define the purpose (as stated previously) and general objectives (design requirements, capabilities & associated back-end functions) of the project.

Project Planning [The Design Phase]:

Researching this project was a long process, taking a few months to complete due to the preliminary process of figuring out what was needed to make my designs work. Initially, my research pertained to how I could build an apt website for this task. I looked at various options for front-end/back-end stacks and various ideas for incorporating CourseDog data (Plan A), Banweb data (Plan B), or even Oracle/Banner data into my back-end/database and tried to be both realistic, but also minimal given that I'm alone in the design and development for this project. A significant amount of my time was invested investigating the feasibility, of using either scraped banner data, or CourseDog data in my Python-based back-end since I needed reliable information from somewhere. This led to plans A and B, which correspondingly dealt with the logic and logistics of collecting the data I need in different ways. In preliminary research, I had some correspondence with CourseDog engineers to discuss the logistics of executing Plan A, which was a great primary source of information for me, however, that relationship deteriorated and it eventually proved unreasonable to attempt to access data like this. This was because API assistance was out of scope for the NMT contract and because I couldn't get an authorized API account in time. Instead, Plan B uses the Registrar's Office for support, and collecting existing data from NMT's course offerings site proved more attainable, maintainable, and valuable to project completion.

From my research and from the opinions voiced by the survey respondents, I came to a conclusion regarding the actual components of the project stack: for the sake of simplicity, and known security, as well as because I have existing experience with this software that should expedite development. The front-end would use simple React components powered by Next and the Node engine because they're simple, universal, and incredibly easy to stylize as well as to link other pages or back-end components. This will allow simple implementations of JavaScript on each page/component and generally allow for a slimmer build. The back-end will be based on Python3, using the Flask framework specifically, as it makes it incredibly easy to define routes, services, and interaction with data models is streamlined.

Plan A was the initial primary plan: to rely directly on the CourseDog-Banner API to service data to front-end pages. The API required a regularly updated OAuth2.0 Bearer token,

which was logistically easy to get and handle, however between the staging status at the registrar's office, and the CourseDog account types available, I didn't have the authorization to collect a token in the first place, despite being able to relatively easily feed requests directly into Flask if it had otherwise worked.

This was a good plan in theory, but given the timing and time constraints created by the registrar's relatively slow adoption of the CourseDog software, and the technical /authorization issues encountered, it just wasn't truly viable in the end.

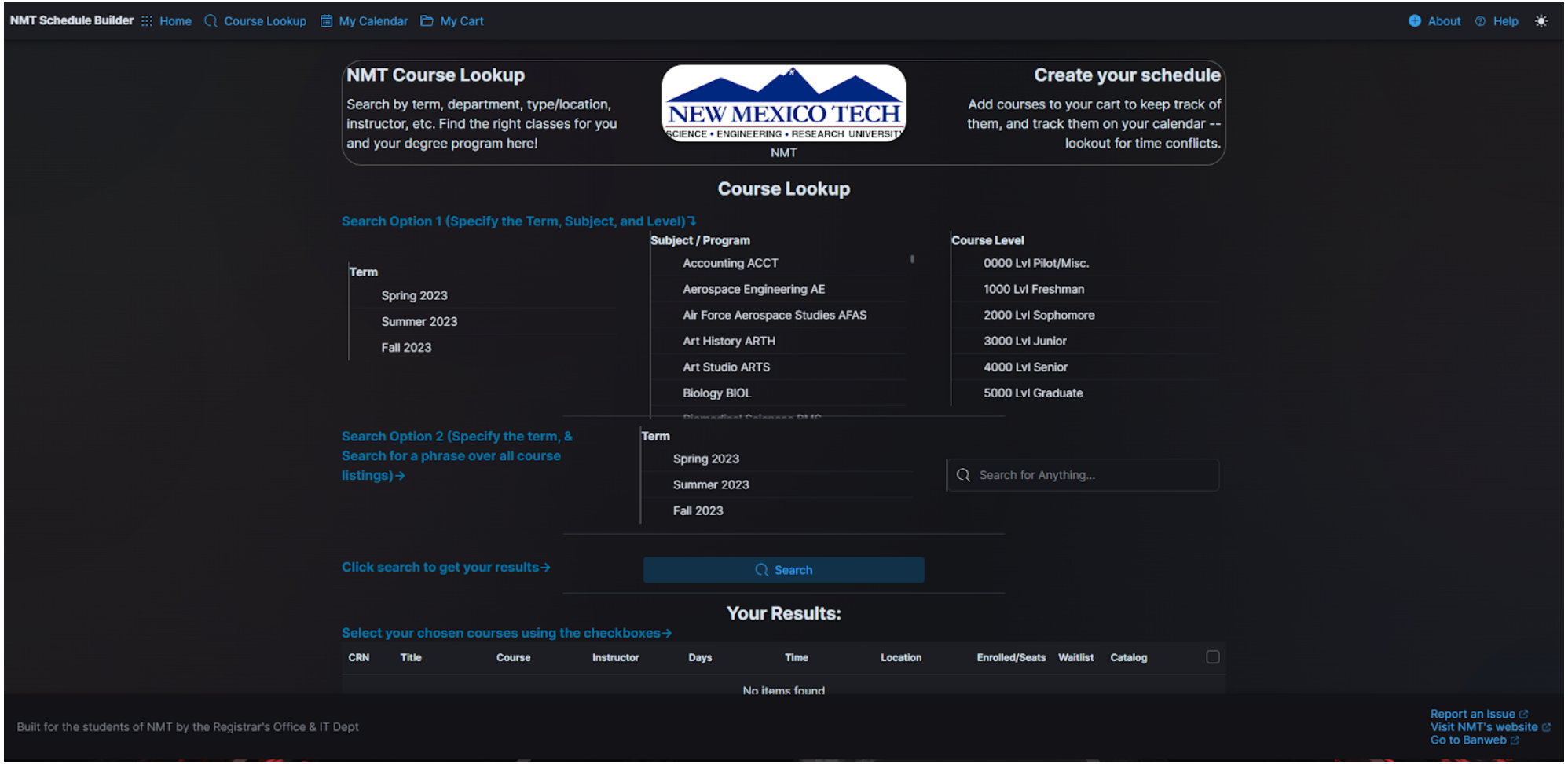
Plan B was initially the secondary plan, but it quickly became the best feasible option for completing the development of the project stack. Plan B was simply collecting all the information my project website will need from public NMT sources. Since Python3 is so capable, I was able to template and slowly refine simple code that scrapes course offerings and parses them into a viable CSV in line with Plan B's requirements. This fits well with the Flask/Flask-CORS back-end I decided to use, which is also Python-based, and meant I could just directly link up the data management code to the rest of the back-end.

This turned out to be a great plan, as Python is my most proficient programming language, and this allowed me to create a very slim and easy-to-understand/maintain back-end. Using simple, included, libraries to scrape and parse NMT public course offerings page, and the commonly used Flask core, my back-end is easy to work with and was chosen to be easy to troubleshoot.

***Project Implementation** [What I've Done]:*

The following is a breakdown of the existing pages any users will encounter on the site, and a guide pertaining to its functionality and capabilities. I did my best to color-match the **Banweb** environment, though I also included a light/dark mode toggle so the site can be more appetizing to more users. I strived to make each page modern but simple, fulfilling but easy to navigate and easy to read, and most significantly, better than their existing alternatives. The culmination of all my efforts in this project resulted in these pages on the website:

1.1. The Home Page: the index, or landing page of the site. The display consists of a branding tile and hook for new users, a guide for how to use the platform, and some supplementary registration links and resources.



2.1. The Course Lookup page: based on the logic of **Banweb** and *Beanweb*, but more sophisticated in that it offers many more search options, and more open-ended capabilities regarding results. Users can search multiple subjects at once, specify course levels, or search generally by term and query phrase. The page also outlines very clearly how to use the various options, to optimize navigation and efficiency for users.

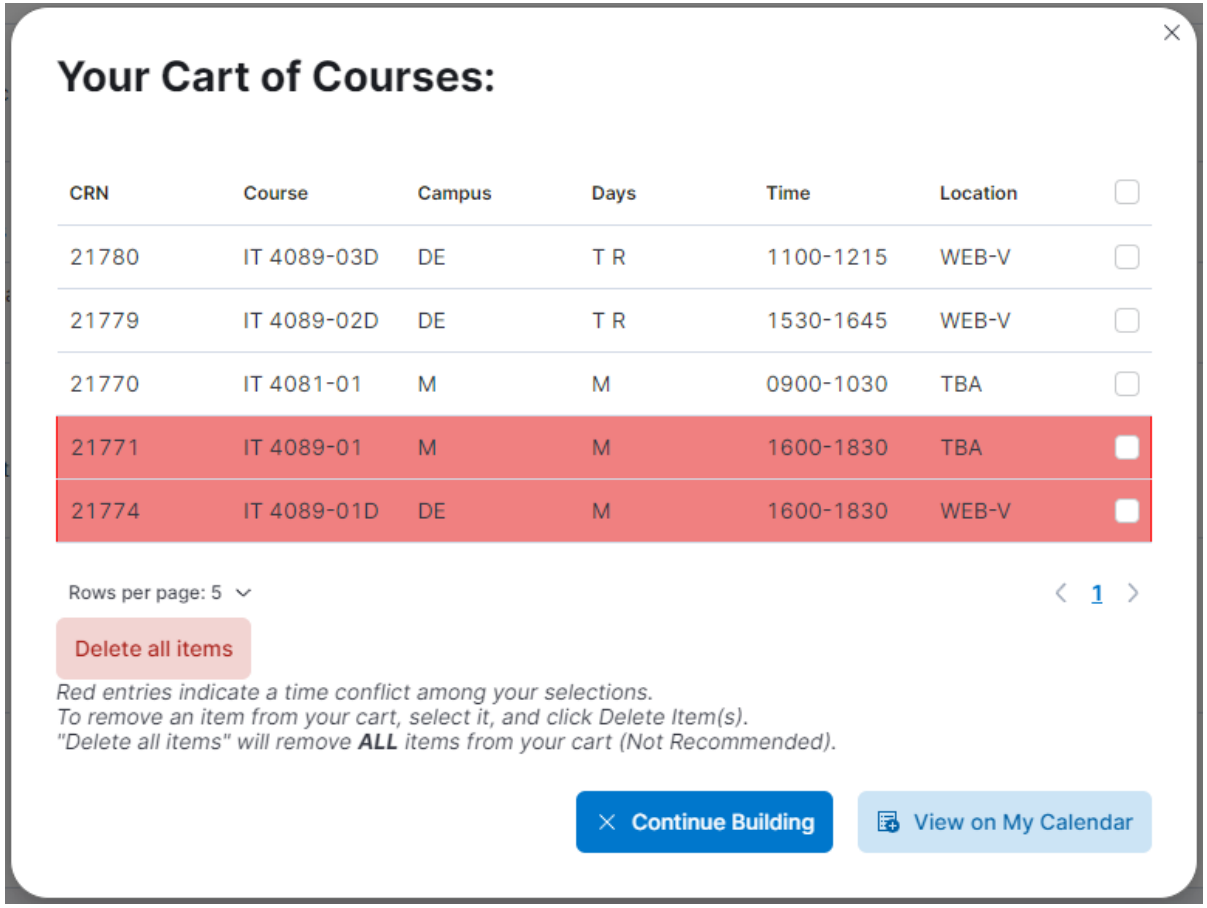
Your Results:

Select your chosen courses using the checkboxes→

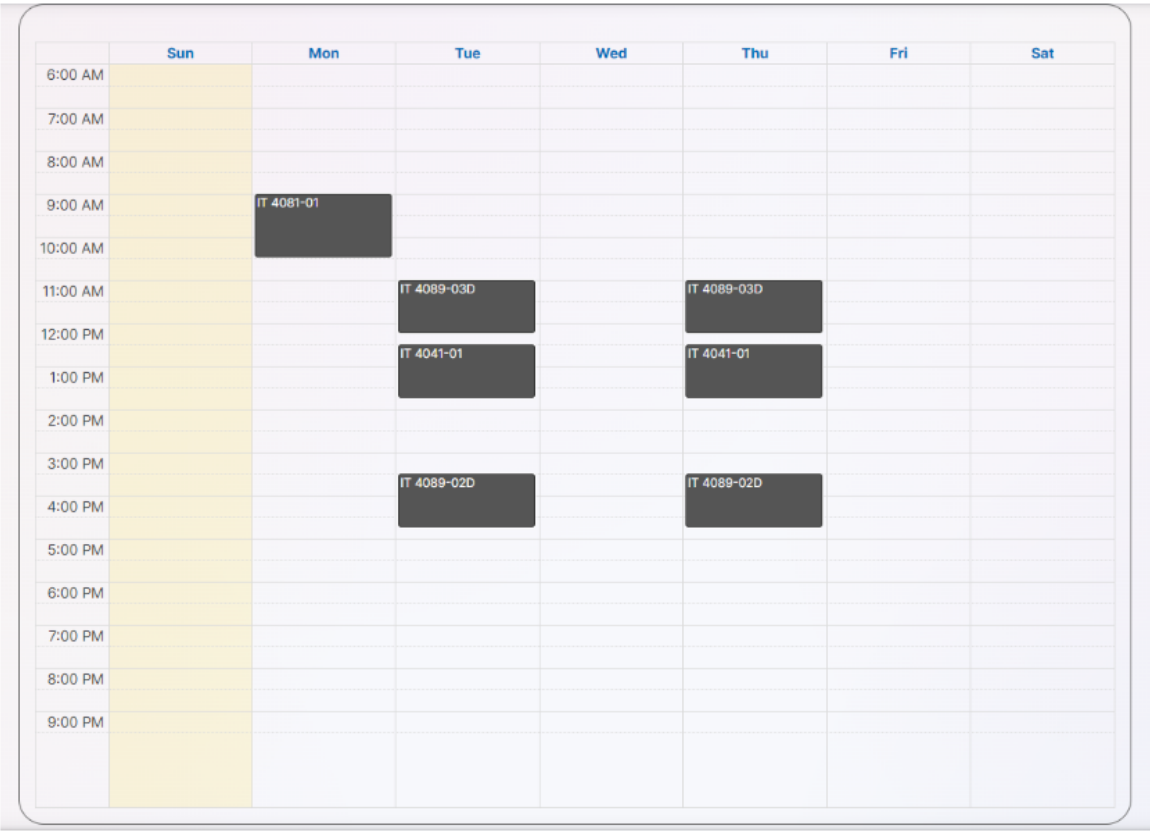
CRN	Title ↑	Course	Instructor	Days	Time	Location	Enrolled/Seats	Waitlist	Catalog	
21780	Blockchain & Cryptocurrency	IT 4089-03D	Dongwan Shin	T R	1100-1215	WEB-V	0 / 15	0	Link	☑
21768	Cryptography & Applications	IT 4041-01	Dongwan Shin	T R	1230-1345	TBA	0 / 15	0	Link	☐
21779	Predictive Data Analysis	IT 4089-02D	Jun Zheng	T R	1530-1645	WEB-V	0 / 15	0	Link	☑
21770	Senior IT Design Project	IT 4081-01	Franklin Reinow & Subhasish Mazumdar	M	0900-1030	TBA	0 / 15	0	Link	☑
21771	Smart & Secure Sensory Systems	IT 4089-01	Hamdy Soliman	M	1600-1830	TBA	0 / 15	0	Link	☑
21774	Smart & Secure Sensory Systems	IT 4089-01D	Hamdy Soliman	M	1600-1830	WEB-V	0 / 15	0	Link	☑

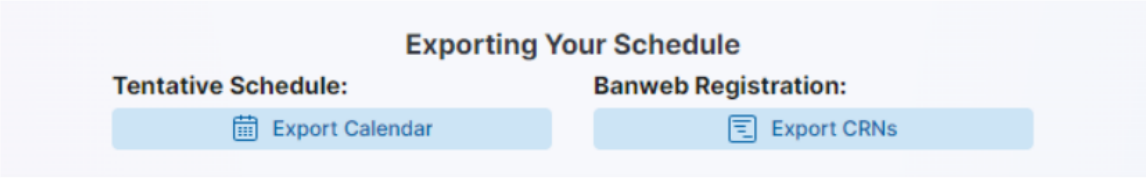
Rows per page: 10 ⌄ ‹ 1 ›

3.1. The course lookup results: a simple table, which I deliberately designed to be reminiscent of both **Banweb** and *Beanweb* gives an optimized view of course information, including links to the new catalog.nmt.edu resources for each course. The workflow the page clearly defines makes it easy to find data and add it to the Cart from here.

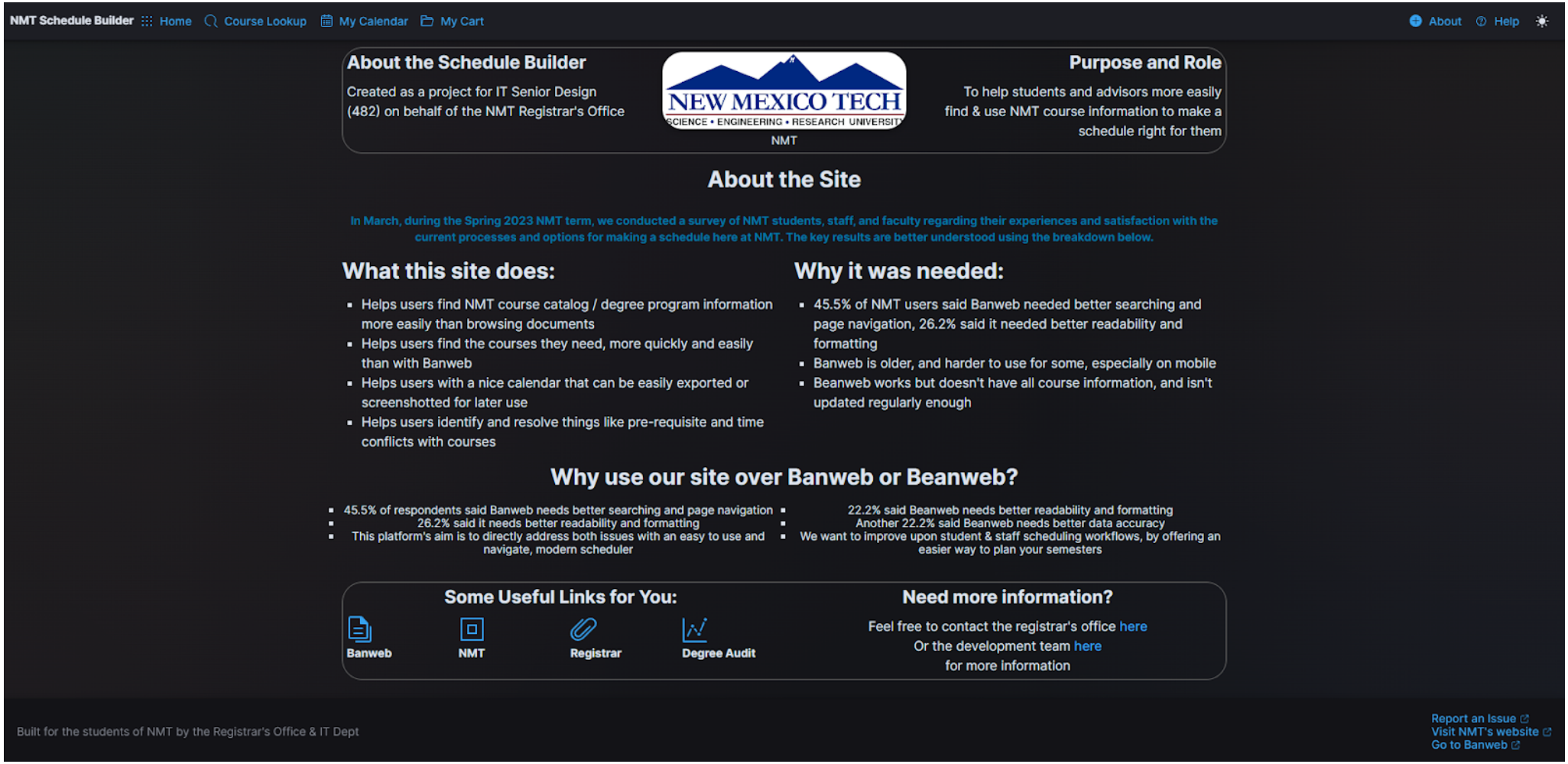


4.1. The user's Cart of selections: it's accessible as a popup modal from anywhere on the site, and allows users to easily see time conflicts, and remove conflicting or unwanted selections. It also offers an easy direct route to the calendar, if the user is happy with their current choices, or a delete all items button if they'd like to start over instead.

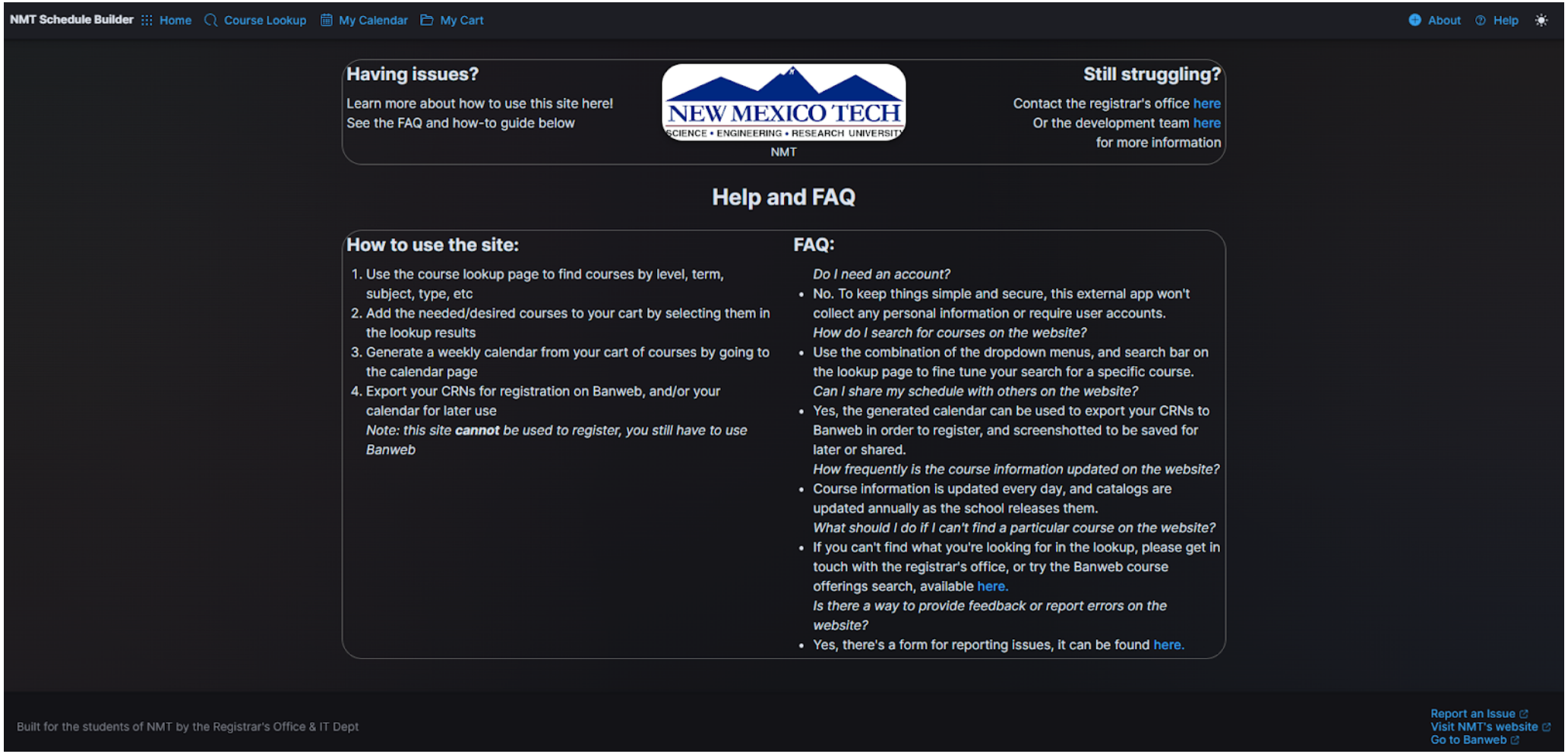




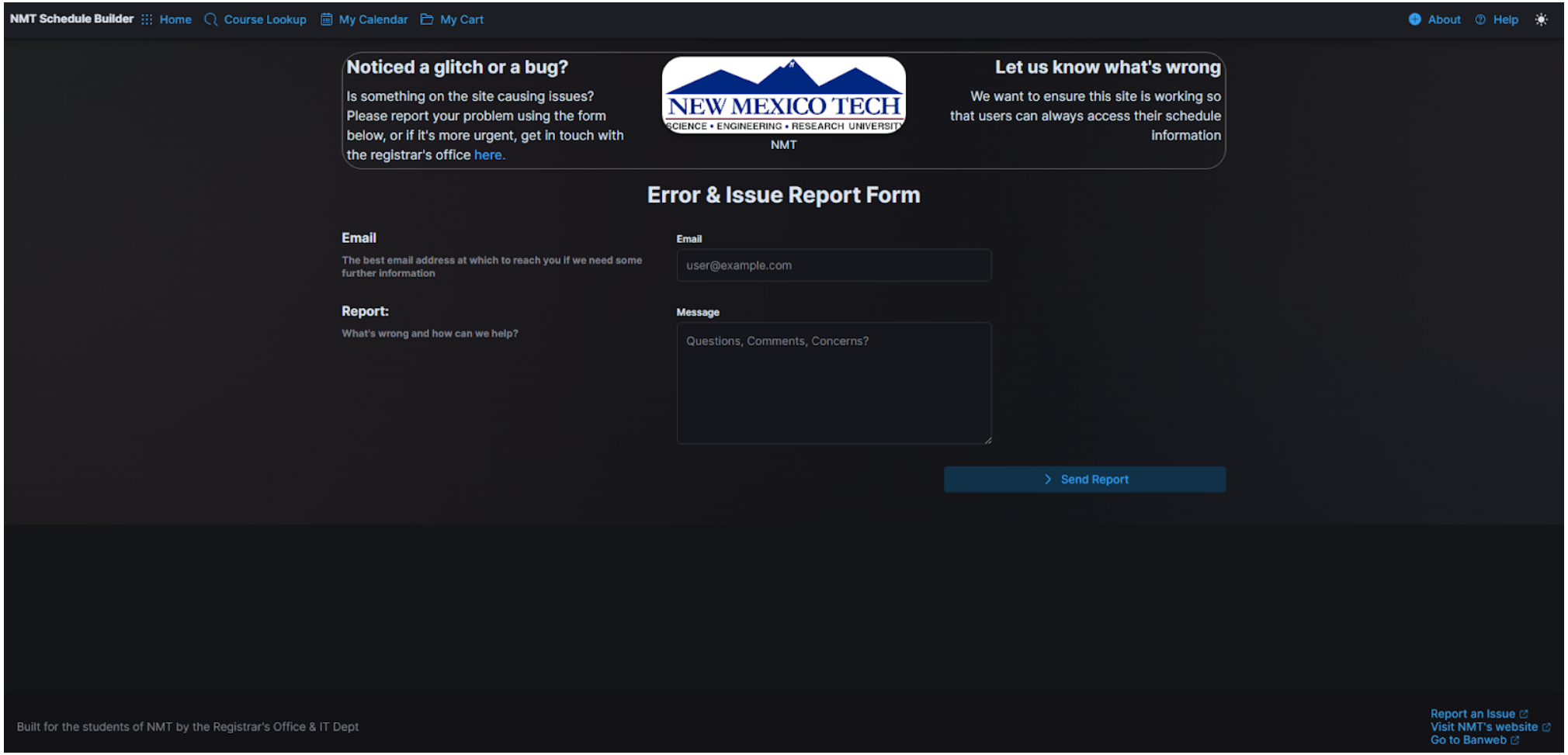
5.1. The Calendar page: displays users' selected courses on a weekly schedule paginated calendar format, with precise time divisions, a good view of the whole day, and abilities to export the information for later use on **Banweb**.



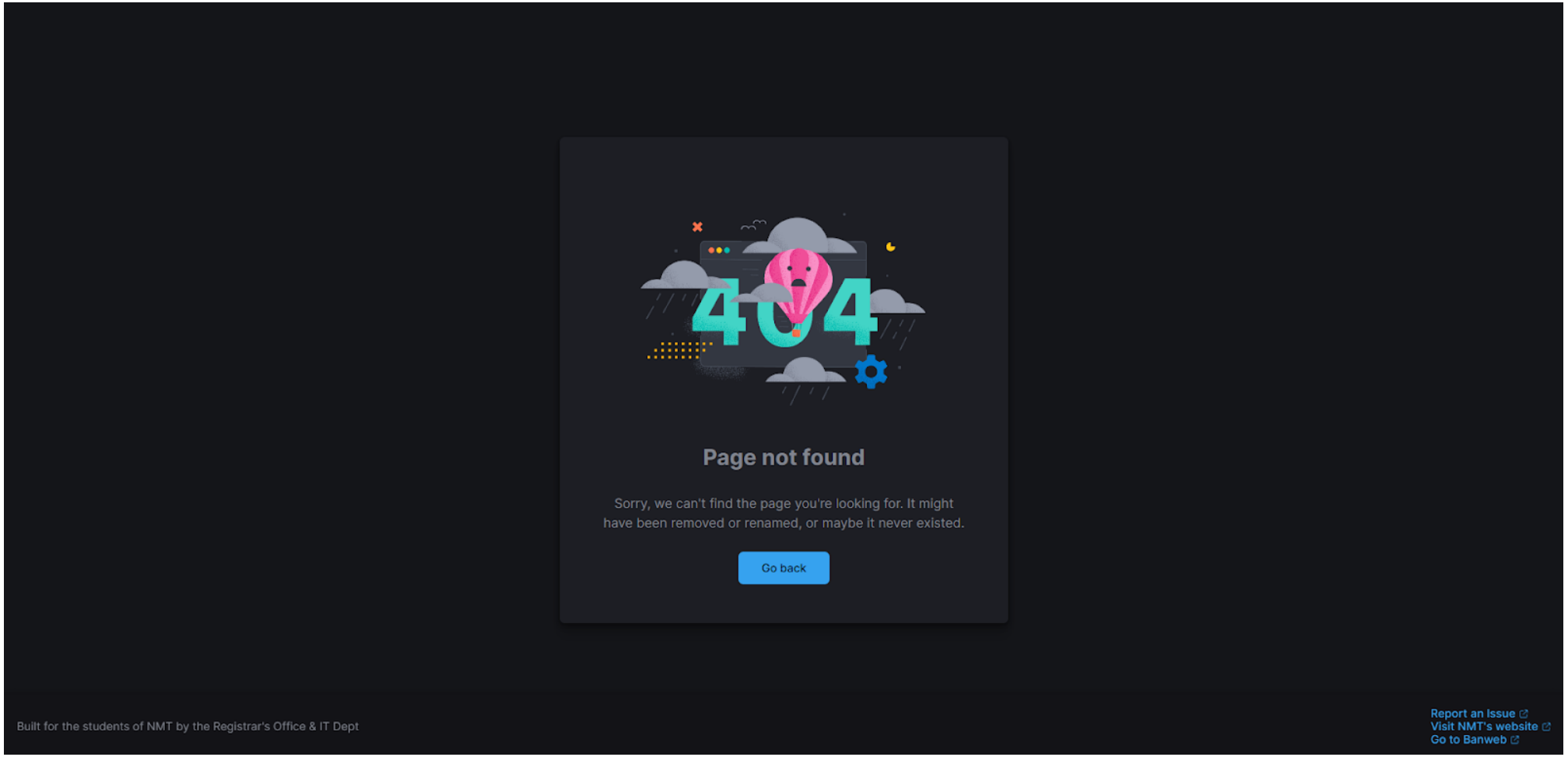
6.1. The About page: gives some reasoning behind why the schedule-building platform was needed overall, and uses survey stats to define the objectives and purpose of the website. It also offers some helpful links with which students can continue their scheduling/registration or find more information about the process.



7.1. The Help page: offers a simplified step-by-step guide for using the site, some additional contact information if users still have unanswered questions, and an FAQ to clarify different features and capabilities.



8.1. The Report page: this is a simple form, with which users can enter their contact info, specify their issues or errors, and send that information to the proper resources at NMT that'll be able to handle the problem(s). It's a simple page and relies on mailto construction to create and send the messages based on form contents, ***and*** this makes it easier for the user to communicate about the topic in the future as mailto will rely on their local email client.



9.1. The 404 page: a simple page from the next.js routing that offers a graphic, message regarding the issue of not finding the requested page or resource, and a button to go back to the last page. Because the overall site is so minimal in routing and the number of URLs, I don't expect this page to be particularly high traffic, which is why it's basically just the given template 404 page.

This is the end user interface/user experience created by my project's front and back-end branches. Though it was a struggle to produce and refine to this state, I'm proud of it and like to think it will serve its purpose well. It offers robust course lookups for students to find and select their courses. It has a familiar and easy-to-use shopping cart mechanism to manage users' selected data. And finally, it has a clearly legible calendar, reminiscent of both *Beanweb's* and **Banweb**'s, corresponding to the wishes of the survey respondents. Also, all along the way, it has clear steps to help users quickly and effectively get/generate what they need even if it's their first time on the site; which are complemented by multiple hints, tips, and links to official NMT resources such as the Registrar, and Banweb websites.

Project Stack Breakdown [What Makes it Work]:

The front-end of the site was based on a Node, Next.js, and React website starter-kit, that came with numerous preexisting components and node modules like ElasticEUI that simplified and expedited the development process. The majority of pages are ".tsx" files containing a data management section, and an exported React component that gets rendered for users. Sometimes, in order to simplify the code, larger elements like forms or tables were put into smaller nested components and then imported into a main component for display. Though the content displayed is actually HTML and the back-end is almost entirely Python, any programming functions included on the front-end are written in JavaScript, and all pages have clear commenting and a clearly documented code and scripts section. Also, though the back-end may interact with data as a CSV, most of the front-end components and modules require JSON formatting, a relatively minimal conversion that happens in some components to handle incoming data, as well as sometimes on the back-end to prepare the data requested by a rendered page.

The back-end of the site was based on a Flask (Python) and Flask-CORS setup, additionally using libraries like *[request, re, BeautifulSoup, and json]* to collect, parse, maintain, and service course data to the pages on the front-end. This Flask back-end was subdivided into 3 primary modules: Models, Services, and Controllers. The Models contain all the CSV data used the by site as a whole, as well as the Python code needed to collect and parse it. Services contain the actual code: Python methods and functions for manipulating or sending data between parts of the project stack and

connecting features on rendered pages to real data. Controllers are supplementary blueprint routes, similar to the top-level app routing, but for more precise and/or less common functions the back-end may need. The three branches are managed by an upper directory Flask `app.py` which coordinates all routing of requests and responses containing information between the users and the lower branches and course CSV data. The production build of the back-end requires the Python3 Gunicorn webserver service to stably host a WSGI build of my Python code, and this is a very simple install through pip, and a very simple command to run the service in a coherent configuration that the front-end can communicate with.

Development Issues & Overcoming Them

[What Went Wrong and How I Addressed It]:

During the design and development phases of this project, I encountered several issues with both the fundamental logic of my process and the actual code in the project stack. These were things that temporarily stunted project development or shifted my thinking, as well as things outside the project scope, such as my personal research/learning process, and my stress. I almost universally used my recognition of these issues to push myself past my comfort zone - in order to address known problems and complete the project as required, to the best of my ability. The following breaks down where and how I encountered issues during my Senior Project Design terms, and proceeds to address how I responded to them in order to meet the development goals and grow as both a developer and an IT student.

In order to meet the requirements of my Senior Project Design term, I needed to gain confidence in my capabilities in order to be less stressed while working - this was key to finding my resolve, and powering through to resolve final issues and address conflicts, and complete the project. More specifically, I overcame the known major issues I had, in the following ways, bolstered by an improved outlook/resolve and stress management strategy.

First, I was almost constantly making slight changes to the included modules and components, which occasionally had cascading effects like breaking peer dependencies or entire pages. This was because I was experimenting with packages to find out what's easiest to work with vs implement, and I eventually settled with one build in order to minimize both the number of requirements and the effort required to manage dependencies. The most direct example of this

would be the calendar, where since I'm using such an open front-end stack, I had many compatible options, but there were several tradeoffs to each one such as appearance and functionality I had to balance in order to achieve the project's main goal of a slim but useable modern weekly calendar. This challenge was pretty consequential time-wise, but it was relatively easy to diminish its effects of it by finalizing decisions.

In response, I had to get a concrete project stack that I could associate with my objectives and to-do list. Finalizing these decisions allowed me to go through and systematically reinstall or add the unmet or corrupted peer dependencies that were causing the majority of this issue. To continue the direct example from the last section, this was the point where I decided to go with the FullCalendar-React module configured using minimal plugins; because between other components, and the pop-up information display I have there were numerous conflicts caused by the variety of plugins and calendars I tried. Choosing the best options for the course lookup, results table, and calendar components let me go through one final time and resolve this issue. The dependency documentation was updated and I ensured I had the latest version of all the components and I moved on with development.

Because of the disparity between Plan A and Plan B, much of the stack's back-end remained a very open-ended work in progress. I'd been working on Python implementations of both schemes simultaneously, because of the delays in communications with both CourseDog Support & Engineers and the Registrar's Office. However, as a result of developing 2 Model schemes, the rest of the back-end like the routing and services idled until a decision could be made. Eventually, I all but gave up on Plan A. This was pretty late in the Spring term, and I was tired of waiting and going back and forth; I additionally realized that I really needed to buckle down and finish development so that the website would be ready to deploy.

Accordingly, I became heavily invested in finishing and polishing the process of collecting and parsing data scraped from NMT's course offerings page for Plan B. Once that was working smoothly, and I had the front-end pages working I was able to start syncing up the routes, services, and capabilities of the Flask back-end to match the CSV data I have and the components I need to send it to. This meant that development could proceed as planned and I was one step closer to completion.

Finally, coordinating the conversion of course data from CSV to JSON mid-route from the back-end to the front-end proved to be unexpectedly challenging. Because many of my core pages feature largely pre-existing elements and plugins, I had to conform to their rules for data inputs; typically consisting of casting a data frame entry from CSV values into a JSON dictionary format like the following: `{ Column: 'Value', }`. This was an absolute necessity to be able to serve my collected data into the elements I built to display them, but it proved to be problematic. This was largely due to my fading familiarity with the Pandas library (I had some conflicting logic based more on the `csv` library), and this also directly corresponded to the peaking stress of having an issue while implementing the final sync of the front-end and back-end branches and functions.

To address this and finalize the project so it could be deployed, it honestly took mere brute force. I spent a handful of sleepless nights working on researching, outlining, configuring, building, and debugging all of my backend routes. It meant some of this conversion ended up taking place in front-end JavaScript, but the hard work paid off in the end because the setup ultimately functions sufficiently well.

Right at the very end of the development process (literally the last week of the term), as I was switching the back-end over to the production branch, and compiling the front-end for maximum efficiency, I encountered a really unfortunate problem. At first, I was just completely baffled; I'd never handled this process before and though it should've been straightforward it turned out to be a mess. I contacted my old boss at the NMT CCoE, Alec Benson, who was our deployment manager, and evaluated and discussed my issue with him. Primarily, it's that the Calendar component, which works completely fine in the development mode, is broken in the compiled build, styling and formatting-wise. We tried several different versions of the package and several debugging options like completely reinstalling all node modules. We scoured forums and documentation, tried tips, and I submitted a support ticket to the (relatively active) package management repository. We also noted a handful of existing other tickets and posts related to this exact issue, and largely – the consensus is to wait a couple of weeks-months for the proper update to @FullCalendar-react and use broken calendar formatting. Well, that or run your frontend server in the development branch until the update is available, at the cost of speed and cleanliness in page loading, until the point at which you can re-install the modules, and re-compile the front-end and host it properly. This was immensely stressful that such a serious issue was pressing

me right at the end of the project development cycle, and that the solution was out of my control. Thus, the best I can do is provide an explanation of the issue and coherent solutions, and directions to the SysAdmins that will handle the deployment of the schedule-building website platform, on how to properly start the FrontEnd production branch server accordingly.

Overall, I honestly experienced some serious stress and hardships throughout the process of working on this project – because of being underprepared for web development, and because of being alone. But, I've grown as an IT developer, and especially as a student by learning how to work (with/against) my stress and the paralysis feeling it caused me at points. It hit me hardest as I had to recognize and overcome technical problems along the way, but it mentored me to learn better from my mistakes and eventually, I was able to pull through with a deliverable project for the IT Senior Design program.

Conclusions and Final Thoughts [Ending the Project]:

There were some concessions that had to be made along the way, because of time constraints, technical limitations, and even to some extent the limits of my own skill and understanding. These included the ability to export one's calendar, containerization of the project stack using Docker, and SMTP form submissions. First, to accomplish exporting, I started by thoroughly researching the various API methods to export a Google, Apple, or standardized calendar format, and attempted a couple of implementations of relevant code. It ended up being out of the scope of the project however, mostly because to utilize those services on a project at this scale would've required a monthly subscription and some more complex authorization implementations that wouldn't fit well with my existing backend. This meant I eventually conceded and reverted to suggesting screenshotting the calendar, something many Banweb users already do. Because of this familiarity, and the message explicitly recommending users to screenshot their calendar, I feel like this concession will have minimal impact on the use and workflow of the platform.

Next, while investigating containerizing my project stack, it quickly became clear that because I'm using a WSL2-based environment for development, I'd have to use Docker Desktop with the WSL2 engine. This seemed fine at first until I attempted further, and discovered that I'd

have to start from a template in the Docker repository, I couldn't start up my own container with my own configurations. To add insult to injury, it also would require a subscription service to obtain one of the templates due to the scale of the project, and I eventually just conceded this objective because it's easier (and I'm more comfortable with) just transferring the project files and doing some setup on deployment than changing development environments just to containerize.

Finally, I fully implemented the issue/error reporting form on both the front-end, as well as its Flask route on the backend to handle sending the contents of the form via SMTP. I had it all set up and logistically debugged but upon actual testing, Google's security protocols were largely too strict to allow my Python code to sign into the generic new email address I'd created for the site to send mail from. I checked out a couple of different email platforms but eventually conceded that further authorization protocols were out of the scope of my backend app and switched the logic over to using the mailto protocol, allowing users to fill in the form, and upon submission transfer the contents to their own email client in order to send it in. One benefit of this alternative, I suppose, is that the logic to make it function is limited to frontend javascript, making it a more simple and more efficient process. This concession didn't really impact the overall use or workflow of the site but it was unfortunate to have to alter one of my development objectives. Though it was unfortunate not to meet some of my own design objectives and project objectives, I worked hard to minimize any noticeable loss from the final product and to optimize the objectives that were otherwise met during development.

Regarding the final status of the delivered/deployed project stack, I feel as though I met the design goals set forward by the survey respondents, and my own design objectives reasonably well. ***The only*** exception is that I was unable to get an updated version of the FullCalendar-react package before the end of the term. This is something the SysAdmins managing the deployment and hosting of the site will have to look out for, because it's now out of scope for me to affect it, and far too late in the development cycle to make any drastic(package) changes to the project stack. The purpose of the platform I developed was to offer modern, easily navigated, easily readable course data services so that NMT users can make their term schedules more efficiently. I polled the general population to find out the problems with existing options, and their requirements for improvement and set out to implement corresponding features and mechanisms in my project stack. The Lookup and Calendar are apt examples of this process, as they were

tailored to what NMT users wanted. In the survey that I conducted in early Spring, users were relatively unanimous regarding the biggest issues with both *Beanweb* and **Banweb**. A majority of users suggested that:

- **Banweb's** searching and navigating needs the most improvement.
- **Banweb's** readability and formatting also need attention and updating.
- **Banweb's** calendar view is a well-liked format but outdated and underused.
- *Beanweb's* data accuracy and page readability need to be improved.
- *Beanweb's* update frequency could be improved.
- *Beanweb's* calendar view is a well-liked format but outdated and underused.

Resulting in my design objectives and evolutions, and the final product, that was tailored to respond to these specific needs and suggestions provided by the respondents. The most affected pages were the course lookup and the personal calendar. The lookup page, rather than being step-by-step and rather limited like **Banweb**, has 2 main search options that give users much more control and allows them to access multiple subjects at once by hinging on the term rather than a department. The calendar looks a lot like **Banweb's**, but it includes labels and divisions similar to *Beanweb*. It also has a cleaner display offering additional course info as a mouse popup and is more precise than other options because it measures from 6:00 to 22:00 in 00:30 increments. Then, generally, in response to users' requests for modernization, formatting, and readability, the more static pages of the site like the index, about, and help pages are simple in that they have minimal content, modern in that they're well-written and modularly organized, and easy to navigate using the limited number of familiar header/footer buttons. In order to additionally satisfy the users of the new site, I generally color-matched it to the **Banweb** platform to make it look familiar, and I developed a duplicate (inverse) theme in order to offer an easy toggle between light and dark modes that will be additionally appealing. All in all, I do feel that my final product offers a considerable improvement upon the issues reported in both *Beanweb* and **Banweb**.

I did have some other problems during development, but I recognized and overcame these issues instead of conceding my objective to them. First, I was having numerous peer dependency issues because the front-end stack utilized the node engine, complemented by NextJS and React, meaning I had around 650 requirements – and some didn't play well together. Overcoming this was simply a matter of making final decisions about what elements and plugins will be used to

render front-end pages and components, as I'd been experimenting with multiple options for the calendars, search results, etc. Another notable hurdle was getting over the paralysis caused by the deviations and disparities of Plans A and B. In the end, I just decided that time was running out and thus Plan A would be infeasible, and I resolved to finish out the backend requirements to meet the needs and objectives of Plan B and abandon the logistics for importing course data from the CourseDog software. Ultimately, I also ended up having several small issues during the final linkage and debugging of the front/back end stacks resulting from the various conversions between existing CSV data and needed JSON data. Overcoming this development obstacle essentially boiled down to research, time, and brute force to get those conversions working properly. Accordingly, the logic to do this is a little strange and separated but it's at least well commented and documented. Overall, despite the issues I was hard-pressed to resolve along the way, I submit via this report and the contents it elaborated upon that I've fulfilled my project's design and development requirements for a scheduling website the general NMT populace can use for scheduling their semesters and generating a corresponding calendar.

In conclusion, this project went approximately as expected. Though I've been reasonably confident with my skills in this field, Senior Design really put that to the test and force me to learn I was wrong. Not necessarily wrong about my Information Technology skillset, but more about my ability to apply them to a real project where there are things like time constraints, real problems to solve, and significant unknowns. However, I still conclude that the project ended as expected. I often must force myself to deal with my stress, which corresponds to the problems I'm having. So, while I felt very good about the prospects of the project at the beginning, I had some real struggles throughout; creating a resultant project largely similar to most of my other higher education endeavors. I start with an idea and am forced to evolve by both constraints and stress, before arriving at a final product shaped by the process. I expected to develop a specific workflow to ensure the completion of this project but ended up learning more about the nature of how I actually work through problems, learn, and respond to challenges. Overall, this web development project was stressful, but rewarding, thanks to the huge learning curve and the experience that it gave me in conflict resolution and working hard; it was a good fit for my Senior Design project and I only wish that I hadn't been alone to work on it.

*Guide: Use, Maintain & Diagnose the Platform**[Debugging & Deployment]:***

Project Repository: https://github.com/gingebrigtsen/NMT_Schedule_Builder

Needed `FrontEnd/.next/` directory: here.

The front and backend production branch servers run at localhost `(127.0.0.1):3000` and `127.0.0.1:5000` respectively – these can be connected to the host domain URL in order to access the site at a larger scale and thru a cleaner address. The hosts/sysadmins will typically want to start the back-end first, as it has multiple threads and will take a little bit longer to start. This requires a pipenv environment, with the proper dependencies installed, as well as the Gunicorn WSGI server service. Starting the back-end properly can be done by:

- Start the Flask-based back-end server using the pipenv environment, and install the proper requirements from the pipfile.lock using `pipenv install`.
  - Ensure that pipenv is locally or globally installed by installing it with pip.
- Next, start it with `pipenv shell``.Then, execute the properly configured WSGI Python server with `gunicorn -w 4 -b localhost:5000 app:app`.
  - Ensure that Gunicorn is installed and up to date in the pipenv shell, it can be installed with `pip install gunicorn` and version-checked with `gunicorn --version`.
- Ensure the redis session management service is installed and running in the pipenv shell, with `sudo redis-cli ping` and expect `PONG` if working properly. This service is what maintains users' cart session data and must be running for Flask's configuration to work properly.

Next, when starting the front-end, it's a good idea to re-compile the build (in case of any issues or artifacts in the process of cloning from the project repository), then start the server using the native package manager, yarn. Here's how:

- If you need to install the project dependencies, use `yarn install` this will use the existing `package.json and yarn.lock` to fetch all the dependencies and requirements specified.
  - Sometimes, yarn will erroneously install nested `node_modules` folders inside packages within `node_modules`. These will stop the build from running or compiling.
  - They must be deleted before the front-end will work so if possible using the modules directly off the cloned repository would be the best way to avoid this issue.

    - If you do need to find and delete them to resolve build errors, they're in `node_modules/@types/`:
        - hoist-non-react-statics
        - react-beautiful-dnd
        - react-redux
        - react-virtualized-auto-sizer
        - react-window
- `yarn build`
- `yarn start`

The back-end has a debugging route, a `Hello World` page that will display at the base URL if the backend server is working properly, and the front-end's base URL will render the index/home page of the website when fully up and running. A clean startup for the backend will look like:

```
(BackEnd) st-n9177@ST-N9177:~/work/NMT_Schedule_Builder/BackEnd$ gunicorn -w 4 -b localhost:5000 app:app
[2023-05-04 19:55:49 -0600] [722] [INFO] Starting gunicorn 20.1.0
[2023-05-04 19:55:49 -0600] [722] [INFO] Listening at: http://127.0.0.1:5000 (722)
[2023-05-04 19:55:49 -0600] [722] [INFO] Using worker: sync
[2023-05-04 19:55:49 -0600] [724] [INFO] Booting worker with pid: 724
[2023-05-04 19:55:51 -0600] [732] [INFO] Booting worker with pid: 732
[2023-05-04 19:55:55 -0600] [740] [INFO] Booting worker with pid: 740
[2023-05-04 19:55:58 -0600] [748] [INFO] Booting worker with pid: 748
[2023-05-04 19:57:26 -0600] [722] [CRITICAL] WORKER TIMEOUT (pid:732)
```

And a clean startup for the frontend will look like this (the next config warning is about exporting and can be ignored):

```
st-n9177@ST-N9177:~/work/NMT_Schedule_Builder/FrontEnd$ yarn start
yarn run v1.22.18
$ next start
ready - started server on 0.0.0.0:3000, url: http://localhost:3000
info  - Loaded env from /home/st-n9177/work/NMT_Schedule_Builder/FrontEnd/.env.local
warn  - Invalid next.config.js options detected:
  - The value at .output must be one of: "standalone".

See more info here: https://nextjs.org/docs/messages/invalid-next-config
```

This process can be automated by creating a bash file e.g. `start.sh` that runs these commands properly according to the host environment. That generally concludes the process of starting the web platform. To stop or shut down either branch, the goto method is the keyboard interrupt, `Ctrl + C`. This will abruptly but cleanly stop the front-end server, and issue an atexit thread cleanup in the back-end before a clean shutdown. This shutdown protocol should also be automatable with a bash script written according to the host environment. Or the automation of

the web platform can be done using an external management library or software that SysAdmins are likely to already have access to in the host environment.

The back-end branch `/BackEnd` is relatively simple and stable compared to the front. The overhead `app.py` has all of the Flask, Flask-CORS, and Redis application and session configurations, as well as all of the routes/endpoints utilized by the front-end. There are also several subdirectories: Controllers, Models, and Services. For this project, the Controller branch is unused and unneeded – it's only included because it's an initial part of the boilerplate Flask project. The Models branch contains my Python scripts for scraping and parsing data from **Banweb**, which is then stored in a local subdirectory `csv`. These scripts are controlled by an autonomous thread started and managed by the overhead application, and will run every 24 hours. Finally, the Services branch contains a Python script for some side/helper functions that are key to both the back-end endpoints and the front-end course lookup. They regularly check, and update configuration data based on information provided on Banweb. After extensive testing, these routes, services, and scripts seem to be working properly and stably and thus shouldn't need much in the way of diagnosis, interference, or maintenance. But in the case they do, my Python code is well vertically separated for readability, and also very well commented. This branch should be easy to work with.

The front-end branch is much more complex: it came from a starter kit and as such it has numerous modules and dependencies. That being said, I only actually worked on a handful of files and anything that I've touched is extremely well commented and additionally formatted by prettier to be easier to read. Those can be found in `FrontEnd/src/pages/`; they're executable typescript files (.tsx) that utilize javascript, ElasticUI styling elements, and rely on the Next.js service and Node engine to run and render React pages. The Node packages, types, and modules can be found in `FrontEnd/node_modules`, these are the custom web programming JavaScript files the front-end relies on. As further explained and itemized in Section 3 of the Appendix, there ***may*** be occasional issues with this branch, in that certain Node @Types can end up with a nested modules folder in them, and it'll need to be removed in order to build/start the front-end server. There are several other directories and loose files in the FrontEnd branch, but they're of less significance. Things like the public directory (for local media and styles), package.json, and yarn.lock requirement configurations, and also some generic scripts and documents that came

with the starter kit. Finally, there's the `FrontEnd/.next/` directory. This is the actual service that runs the production server, and it contains the static build (compiled) website pages, which will run and load much more efficiently and cleanly than on the development server. This directory is very straightforward but also problematic in that it's not available as a part of the GitHub repository. It contains numerous large files and will have to be transferred over an external medium such as Google Drive, etc from myself to the hosts/SysAdmins in order to include it in the FrontEnd branch. That being said, the build in the repository theoretically ***can*** generate this directory and its resources using `yarn build`, but to be safe I'd like to supply the existing build first which the host environment can then build on top of to correct any new issues.

There's also a significant issue in the build that the hosts/SysAdmins will need to resolve in a matter of weeks or months – once a particular package update is available. I contacted Alec Benson, former NMT CCoE project deployment manager, and we evaluated and discussed my primary issue, the @FullCalendar-react component, which works fine in development mode, but not in the build, in terms of styling/formatting. We tried several different versions of the package and other debugging options including reinstalling all node modules and changing the .next.config.js configurations. We scoured forums and documentation, tried tips, and I submitted a support ticket to the (relatively active) package management repository. We also noted a handful of existing other tickets and posts related to this exact issue, and the consensus is to wait a couple of weeks-months for the proper update to @FullCalendar-react. Until then, I can propose a handful of temporary solutions for the hosts:

- POSSIBLY: run the existing build with broken calendar formatting until the proper update becomes available and the front-end can be rebuilt
- OR: run your frontend server in the development branch until the update is available, at the cost of speed and cleanliness in page loading until the proper update becomes available and the front-end can be rebuilt
- OR: do not host the project until the packages can be updated properly and the front-end can be rebuilt
- THEN: when the fix is available re-install the modules with `yarn add @fullcalendar/react@latest` (being wary of the possibility of nested modules issues) and re-compile the front-end and host it properly.

I'm so sorry to pass on such an issue but it's essentially out of the scope of my control of this project so outlining this solution for the hosts is the best I can do to mitigate the risks to the quality and objectives of the website and project overall. I believe this will just be a temporary setback and not affect the platform in the long term.

Finally, it's strongly recommended but was unfortunately out of scope for me to handle, that both branches of the project, FrontEnd, and BackEnd, are containerized using a service like Docker. This would serve to support my projects' use of `localhost 3000/5000`, make it easier to simultaneously boot or shut down, and connect to the host/public domain URL. Containers have never been a particularly strong skill for me but in my development experience at CCoE, they were extraordinarily useful for the ending phases of development (and though I've never done it myself, the SysAdmins require them for ease of deployment and security), making sure that the branches can talk to each other coherently, and are presented at the proper public URL. The previously mentioned bash scripts for automating running the platform could then be controlled by the docker container instead, as configurations support startup command sequences. With access to a better environment than my WSL2 development IDE, and more resources than my laptop, configuring this container, and compressing/formatting the project shouldn't be too complicated and there may even be good templates available. In conclusion, this would likely be the easiest way to adapt the project to and stably deploy it within the destination host environment, but I couldn't handle this task because I couldn't access the proper packages/templates to do so as they were behind a paywall.

For any other information, refer to the included Appendix and Attachments section below, which provides more direct steps with less narrative explanation, and some tips/documentation.

Appendix and Attachments [Code and Documentation]:

Project Repository: https://github.com/gingebrigtsen/NMT_Schedule_Builder

Needed `FrontEnd/.next/` directory: here.

1. ***Front-End Code (excludes starter kit & components)[`FrontEnd/src/pages:`]***
   - 1.1. *header.tsx (The top bar, primary navigation menu for the website)*
   - 1.2. *footer.tsx (The bottom bar, secondary navigation menu for the website)*
   - 1.3. *index.tsx (The main/index/home page, the landing page for all users)*

1.4. *lookup.tsx (The course lookup and results page where users select courses)*
1.5. *calendar.tsx (The weekly calendar generated from users' cart of courses)*
1.6. *about.tsx (Largely static page about the purpose and uses of the website)*
1.7. *help.tsx (Completely static page, giving simple step-by-step directions, and site FAQ)*
1.8. *report.tsx (Simple report form on a plain page, for users to report glitches, technical issues, or other problems)*
1.9. *404.tsx (Completely static Error 404 graphic page, which offers a back button to return to the last page, though it still has the header and footer so users will still have navigation abilities)*
1.10. *_app.tsx (React Component template for the setup of each page, controls style, and <head> configuration for each page on the site)*

2. *BackEnd Code (Main files, excludes data CSVs and minor files)*

2.1. *app.py():*
2.2. *BackEnd/Models*
  2.2.1. *conf.json: JSON formatted configuration for collecting and scraping from Banner, as well as populating search options from valid **Banweb** options.*
  2.2.2. *CollectData.py(): actually scraping **Banweb** utilizing the requests library, BeautifulSoup4, and html5lib. Generates an un-parsed CSV file, which is then passed into the parser*
  2.2.3. *ParseData.py(): parses broken CSV entries by fixing lines with missing data, secondary instructors, no CRN, etc.*
  2.2.4. *DEBUG: Also included in the project repository are isolated debug functions for dumping banweb HTML, collecting, and parsing smaller sections of course data. These can be run as standalone methods using `` `python3 {script}.py` ``.*
2.3. *BackEnd/Services*
  2.3.1. *requestService.py(): background helper functions for the scraping and parsing processes, managed by discrete threads in the overhead application.*
  2.3.2. *Services are used to hold the actual Python scripts and methods the overall app and routes draw from and require – they're where actual computation tends to take place after being properly routed by the app or the controller.*
2.4. *BackEnd/Controllers*
  2.4.1. *requestController.py(): unused.*
  2.4.2. *Controllers are used to build more precise route blueprints, for properly finding and executing methods within the Flask back-end as a whole – typically for more intensive or less commonly used functions.*
  2.4.3. *No controller was necessary for this project; though originally I had all of my routes templated here, later on in development I concluded doing this was not necessary for the stack to run effectively.*

3. *General Guidelines for Startup & Use*

3.1. *Start the Flask-based back-end server using the pipenv environment, and install the proper requirements from the pipfile.lock using `` `pipenv install` ``.*
3.2. *Next, start it with `` `pipenv shell` ``.Then, execute the properly configured WSGI Python server with `` `gunicorn -w 4 -b localhost:5000 app:app` ``.*
  3.2.1. *Ensure that pipenv is locally or globally installed by installing it with pip.*

3.2.2. Ensure that Gunicorn is installed and up to date in the pipenv shell, it can be installed with `pip install gunicorn` and version-checked with `gunicorn --version`.

3.2.3. Ensure redis is installed and running in the pipenv shell, with `sudo redis-cli ping` and expect `PONG` if working properly. This service is what maintains users' session data and must be running for Flask's configuration to work properly.

3.2.4. Note: Banweb Data is automatically collected and parsed every 24 hours.

3.2.5. Note: Updates to Banweb data are automatically checked every 24 hours. Term codes and subject codes will automatically be updated when new course data becomes available, corresponding to announcements on the Banweb page.

3.3. Ensure the yarn package manager is installed, `sudo apt update` then 'sudo apt install yarn' (If the repository is already configured).

3.4. Even though the deployment-ready project stack will already have been built, it's a good idea to do it again, using `yarn build` from within the FrontEnd folder.

3.5. The project can also be exported to an extremely minimal production build that can be run with a Python HTTP server, for debugging purposes or if it's too resource intensive for the host environment. This can be done with `yarn export`.

3.6. Start the Node-based front-end using a terminal from using yarn to boot the server with `yarn start`.

3.6.1. Note: front-end pages are accessible via `url/page`, whereas back-end functions are accessed via `url/api/{route}`.

*3.6.2. Note: watch out for peer dependency issues, the front-end has **a lot** and may need to be restarted if it overflows and stops working.*

3.7. Navigate to the site in a web browser to ensure it's functioning properly.

3.7.1. Note: All main pages, including the users' cart, are accessible from the top header. The footer, help, and about pages all offer supplementary pages and links for users in the case of an error or if they need more information.

3.7.2. Note: Typically, the largest thing pages must load is the NMT logo, but it's been optimized to load fairly quickly.

3.8. This startup process for the front-end and the back-end can be automated by creating a bash file to run these commands – not included because I have no reference for the platform on which the project will be deployed. The sysadmins will need to either use their existing software management options or quickly put these commands together according to their environment.

End_of_File

Thank you for reading.

Thank you for working with me this year, and making my senior year an opportunity to really grow as an IT student and as an individual, thanks to the design and development practices I learned, as well as stress management strategies I ended up teaching myself in order to power through to the end. I also thoroughly enjoyed the connections I was able to make between what was taught to me in Project Management this semester and what I learned over the course of this project. It was illuminating and I look forward to working on and/or leading projects in my career in the future.

-Gabe Ingebrigtsen