NC Daily

Scholarship project designed by:

Brandon Ru

Original problem

The original problem lies in the student notice system within my school.

The notice system of my school was once run physically, with important events, meetings and announcements read at the start of each day. However, as we pivoted into the digital age — my school has since abandoned this method. The notices must now be checked by students online — through a parent portal website. Students would need to remember to do this and diligently read through each notice.

Many would often forget to do this, or find the online notice board difficult to read, and this has led to a decline in the number who check the notices daily. The online notices system had a difficult to read horizontal layout— which clashes with a reader's vertical downwards 'F' shaped scan. There was also no structure within the notices, with frequent/permanent notices clogging up the view of the reader often. Not to mention, the layout wasn't device agnostic — meaning on a mobile view — the notices were extremely difficult to read. These problems would cause a significant amount of friction in the UX (user experience) of checking the notices — meaning many students would just skip checking them completely.

Impacts of the problem

The obvious impact of this problem is a reduction in student participation. Without knowing what opportunities exist from the notices, they cannot be taken up on. This is a serious issue both to the school and student as it means missed opportunities and a reduction in extracurricular activity. Over the years, many clubs of the school have declined in size consistently.

Furthermore, if the notices are not read properly — or with a low consistency— it means communications from the school must be repeated again. This creates an inefficient system. One example of this is sports trials and practices. Our school frequently, must simultaneously send emails, use the notices and a student messenger to notify students of upcoming events. This is an obvious disadvantage as more resources must be used and there are longer organisation times.

Initial meeting with client (ICT director)

To begin the project, I had consulted this with the ICT director of my school, Craig. He would be the client of my product, as he would be maintaining the app after I finish school and deciding if it was up to standard with higher management.

During a meeting, he expressed strong interest in the idea and agreed with the problem wholeheartedly. Craig mentioned the following points the solution should satisfy. These will be the core criteria that will guide the development of the project.

- ***It would need to be an add-on***. It shouldn't require any change to the current notices system nor modify anything. Teachers should not need to relearn a new way to submit notices.

- ***Be fully autonomous***, as otherwise it would be difficult to maintain over a long period of time

- ***Have ownership easily transferred***. I would be in university in the near future, thus unable to maintain this project.

- ***Be secure.*** In no way should students be able to modify the content distributed and change anything.

- ***Have a low maintenance cost.*** As this is simply an email newsletter, the costs to maintain the project shouldn't be too high.

- **Abide by the law.** The project shouldn't violate any copyright laws, and use ethical solutions.

- **Deal with repetitive notices.** The notices system currently has a lot of duplicate notices. The new solution should fix this.

Possible solutions to the problem

After the meeting, I began brainstorming several solutions I could pursue around this problem. For each problem, I weighed the benefits and drawbacks, before deciding on a final solution.

Graphical redesign of the notices page	
One possible solution is a graphical redesign of the web page students use to check the notices. This could be achieved by reworking the source code of the website which is connected to Kamar.	
Benefits	*Drawbacks*
- No new domain needs to be set up and the website remains at parents.newlands.school.nz - Teachers don't need to learn anything new - Relatively easy to set up	- Requires access to the source code used to run Kamar and the parent portal website. - Might be an infringement of copyright by modifying the website. - Highly dependent on being able to modify the Kamar code. - End product would require extensive testing, to ensure safety.

Email newsletter of the notices	
Another solution I proposed would be emailing out the notices in a daily newsletter	
Benefits	*Drawbacks*
- Allows for notices to be checked through emails, which could potentially be easier. - Does not require direct access to the Kamar notice data stream, as the notices can be accessed through web scraping / an API - Teachers do not need to learn anything new, and nor do students.	- Could backfire, resulting in students not checking their emails and notices. - Requires setting up a custom domain / website, in order to deploy the newsletter system

Completely new notices website	
A solution that was also suggested is a completely new notices website.	
Benefits	*Drawbacks*
- This would provide complete freedom in terms of the architecture of the system and how students navigate the website. - Provides opportunity to create an extremely well refined product, as we are not constrained by an email newsletter nor a pre-existing website structure.	- Requires teachers and students to learn a new method - Must be extremely robust, as once the switch is made, it cannot break. Otherwise, it means students and teachers are flicking between two systems, causing chaos and confusion. - Will require a significant amount of time to research, design, execute and test this approach.

Google document of the notices	
Finally, another alternative suggested was a Google Document with the notices on it.	
Benefits	*Drawbacks*
- Deployment is extremely fast - Builds off a pre-existing structure - Most easily executed, in terms of time and maintenance..	- Difficult to automate - Difficult to make students check, as they associate Google Documents with school work. It is also easily forgotten.

At the moment, I do not know what option to use as I don't believe I have enough information — hence I decided to arrange another consultation with the client.

Feedback on ideas from ICT director and technician

In order to properly decide on which solution was the best in terms of my client, I needed to discuss it with him.

To factor in technical feasibility of the solutions I was pursuing, I decided to also consult with the ICT technician of the school too. He would give me a better overview of how technically feasible each of my goals were and which options were the most easily executed as a student with a limited skillset.

The summary of the responses are shown below.

Analysis: editing the current site source code

Client feedback. He mentioned this idea was possible, however, he knows that the Kamar code is updated regularly and the school doesn't have much control over the source code for the application. But, he said, if I was able to figure out a workaround, it could be great as it means an add-on onto an existing structure. This would meet all the requirements, as long as we had permission from Kamar to edit their code.

Technician feedback: The technician also shared similar concerns, identifying this as a solution that had several flaws. He said the biggest problem was also maintainability. If I was to alter the source code of the current board, when Kamar decides to update the board eventually (which he noted was every year or so), all edits I had made would be replaced.

Another problem he noted was one I had noted, but he fleshed it out a little more. Editing the source code of the website would result in changes to proprietary software which is actually illegal, he noted. Furthermore, upon giving access to the source code, it would also grant to the database of the school. As this contains confidential data (stakeholders, financials, parents), it would have to be discussed with these individuals first, and was something the school would be very hesitant to do, he noted.

Analysis: Production of a new website

Client feedback: It addresses all the requirements of the application, however, the director said it doesn't really address the core problem of my project. Because people are still having to go to another website to check the notices, one that is even more obscure, it doesn't really solve the problem of students not checking the notices. He notes it could even be worse. He thinks this idea whilst allows for greatest flexibility, it doesn't address your problem properly.

Technician response: The technician actually noted this would be a pretty good solution. It would offer ultimate flexibility (wasn't tied into Google or Kamar), and he said it wouldn't actually be too hard to implement (a simple dynamic website) and I was overestimating the difficulty. He said there was an opt-in opt-out method (deciding to check the website or not), and it was a good idea.

Analysis: Google docs output method

Client response: The ICT director noted this was an interesting idea, because it had been used as the primary system before the Kamar system was implemented — about 10 years prior. He said it was a good idea, but it had several flaws. The document was less flexible in terms of formatting, it created a battle for teachers to put their notice at the top and still had the problem of duplicate notices. Another problem was that a Google Docs method still didn't address the problem of duplication - a key project expectation.

Technician response: The biggest problem he had identified with Google Docs was the inherent tie into Google it had. It didn't provide a large amount of flexibility with laying out my new notices (as I had to adhere to a word document) and users needed to be using Google in order to access them. This would limit how well I could design the new noticeboard, reducing the amount of incentive given to the user so they would switch, meaning the project could have not enough momentum to start up.

Furthermore, he noted that Google docs even had a stigma among adults. It wasn't something you opened and checked regularly, but a document for doing work. He agreed with my hypothesis, that students would quickly forget about the document if it was sent to them.

Analysis: Email newsletter

Client response: He thinks this would be an interesting approach, definitely from a psychological perspective. The director is a psychology teacher, and he notes, by sending the emails at a regular time in a regular fashion — it sets up a consistent frictionless system for checking the notices. How I would achieve the other specifications, was up to me and he said the biggest concern was figuring out how to obtain the data from the parent portal.

Technician response: The biggest problem he saw in this method was spam. The emails needed to be in an opt-in opt-out fashion, otherwise the email provider service would flag it as spam —- unsolicited emails. Even then, he said there was no guarantee that the students would receive it.

Another issue was people not checking their emails / ignoring their emails. If people just simply didn't find the email newsletter interesting enough, he said they'll stop checking it. Therefore, it is imperative that the newsletter should be interactive and offer new content every day.

Overall analysis of the three methods

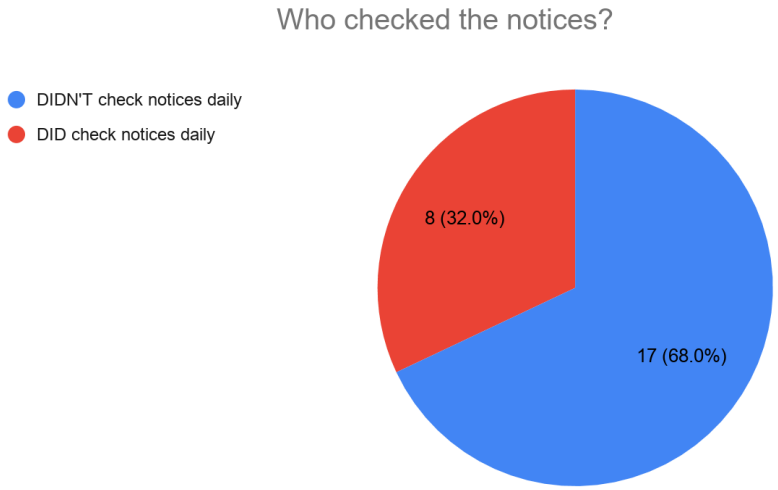
Identified by both client and technician, the largest discrepancy between the four ideas was the nature of the delivery system each of these ideas employed. Within the first three methods (editing current site, making new site, google sites) the delivery system was "passive". It meant that the students needed to actually make the mental effort to remember and go check these places for the notices. Compared to the current system, and said there wasn't much difference, and you aren't actually solving the major problems you are setting out to achieve. In my opinion , I think this is a fair evaluation. It did seem like these new methods of checking the notices only improved how they looked, not how they were checked. For the Google doc method and the new site suggestion , I predict that students would even quickly switch back to the old method of checking the notices. This is because it adds even more to the plate of students to check every morning . Prior to this, the notices were on the parent portal and could be checked alongside their timetables.

However, within the email newsletter delivery mechanism, they noted it was “active”. By delivering the notices in a mail newsletter, right in front of students to check it, it means they don’t have to remember to check it. In addition to this, as it is something students repeatedly use throughout the day, it is therefore a constant reminder for them to check it too.

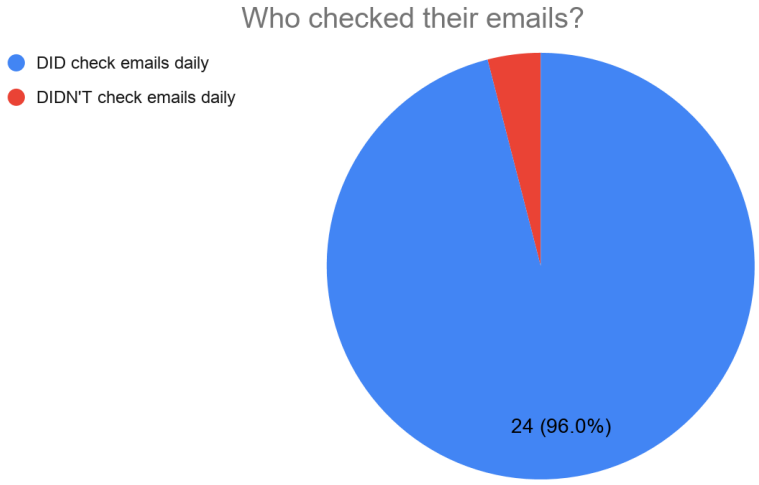
In my opinion , this is a justified opinion and makes sense. The core purpose of the app is to increase the number of readers of the school notices. This is directly proportional to the quality of the notices app, and inversely proportional to the cognitive friction required to check them. This is further supported by the informal market survey conducted - with students indicating they would be more likely to check the notices if they were delivered (actively). Therefore, by implementing an active mail newsletter approach , I hope to address my problem most directly.

Initial student survey

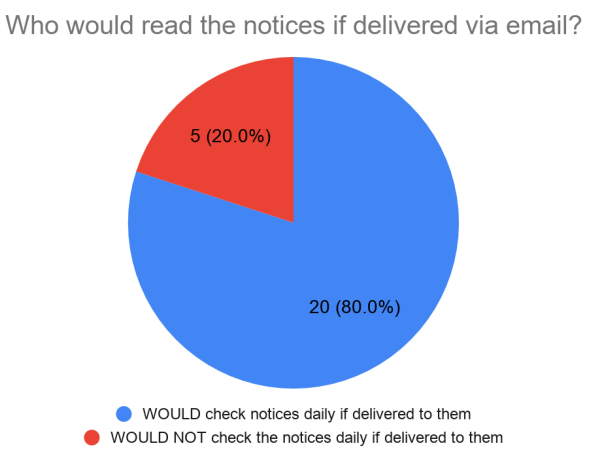
In order to test out what was mentioned about ‘passive’ and ‘active’ delivery , I have decided to conduct an informal canvas of the school. Below is a small market survey to see what the current methods of checking notices were, and how students might respond to this if it was to change. This is important as the end users of the product are ultimately the students of the school. The survey sample size is 25, randomly selected (roughly 2.5% of total school population)



In terms of students checking the notices daily, most students actually did check the notices on a regular basis. (17/25). However a significant proportion of individuals did not, 8/25, indicating this was definitely a problem worth looking into.



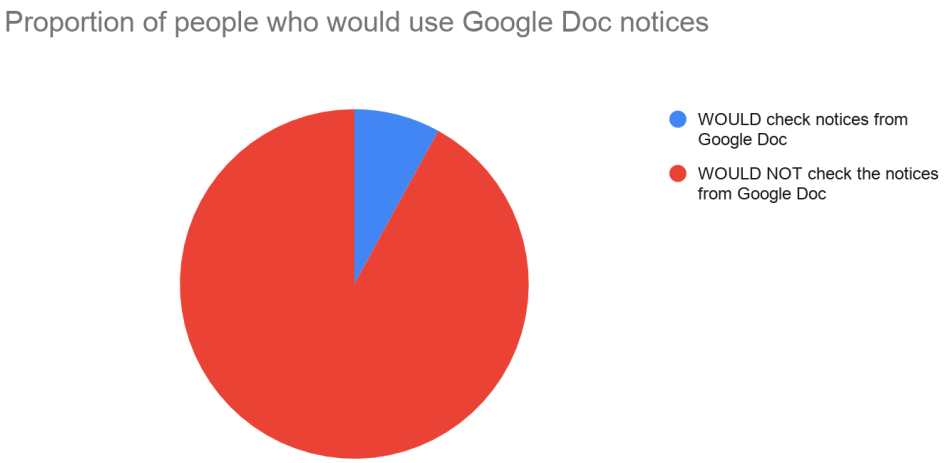
The proportion of people who checked their emails daily, however, was significantly higher. In fact, only one participant of the survey didn’t check their emails every day. This was to be expected, as the inboxes of students are important and used in everyday study.



Then I posed another interesting question which I aimed to identify if one of my ideas to solve this problem would be beneficial: merging the notices system with email.

The results from the survey were encouraging. There were more participants who would read the notices if emailing was used over the current method. Within a larger sample size, this increase would be substantial.

Finally, I decided to further explore the public stigma around Google Documents, and whether my hypothesis was in the right direction.



The results were very poor, indicating that students really did not see the point in switching to Google Documents to check the notices. After the survey, I asked one participant (who answered WOULD NOT) why he didn't like Google Docs. He said it was because it would be very easy to forget to check, even more so than the current system.

Final decision on plan

After considering each of the ideas, taking into account client criteria, technical practicalities and public responses — I have decided on the email newsletter as the final solution. Here's my final evaluation on all possible solutions:

Redesigning the current system
This option would be rather difficult to produce in terms of both skills required and the bureaucracy. I would have to figure out a way to safely integrate my program into the Kamar pipeline, which I predict is not easy.

Furthermore, I would need to convince several senior leaders within the school to allow me to tinker with the proprietary code of the parent portal and spend time developing it. As this contains confidential information, this will be difficult to achieve. Finally, as the technician mentioned, the app would need to be constantly updated with every Kamar update, as they would wipe my progress. This defeats the purpose of the app being maintainable, which the client has mentioned, therefore, this option will not work

Using a Google Docs approach
A Google Docs approach whilst easy to implement, is inherently flawed.

A quick market survey of students showed that many would be quick to forget about it, and wouldn't solve the core problem I originally had noted: students not reading the notices. The benefits in switching to Google Docs doesn't warrant their attention of students, and therefore it doesn't warrant mine either.

Redesigning the current system
Building a completely new notices website is off the table as it does not meet client requirements.

It is not friendly to current teachers either, as they would need to learn a new method of submitting notices. Whilst technically feasible, as suggested by the technician— in terms of client requirements, it falls short

Email newsletter
The idea of an email newsletter is the idea that is most likely to succeed and meet stakeholder criteria the best, therefore will be my final proposed solution. Both the technician and client agree with my decision.

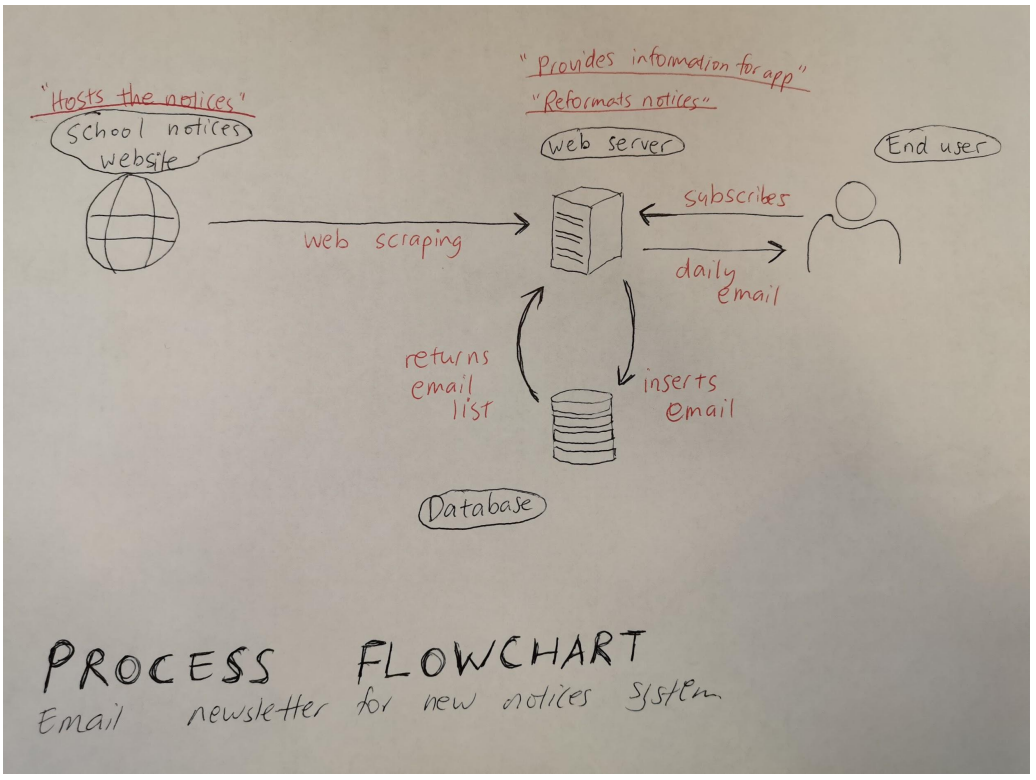
The email newsletter is the solution that tackles the problem the best: encouraging more students to read the notices. The drawbacks that this option comes with (requiring a custom domain, possibly being marked as spam) are outweighed by the macro-benefits offered by this solution.

Initial implementation plan

The plan was to create **a daily, email newsletter of the notices** — **reformatted with a clear structure** — NC Daily. It would revolve around web scraping, as the primary method of gathering data from the current notices board, as this does not affect the current system.

High level process outline

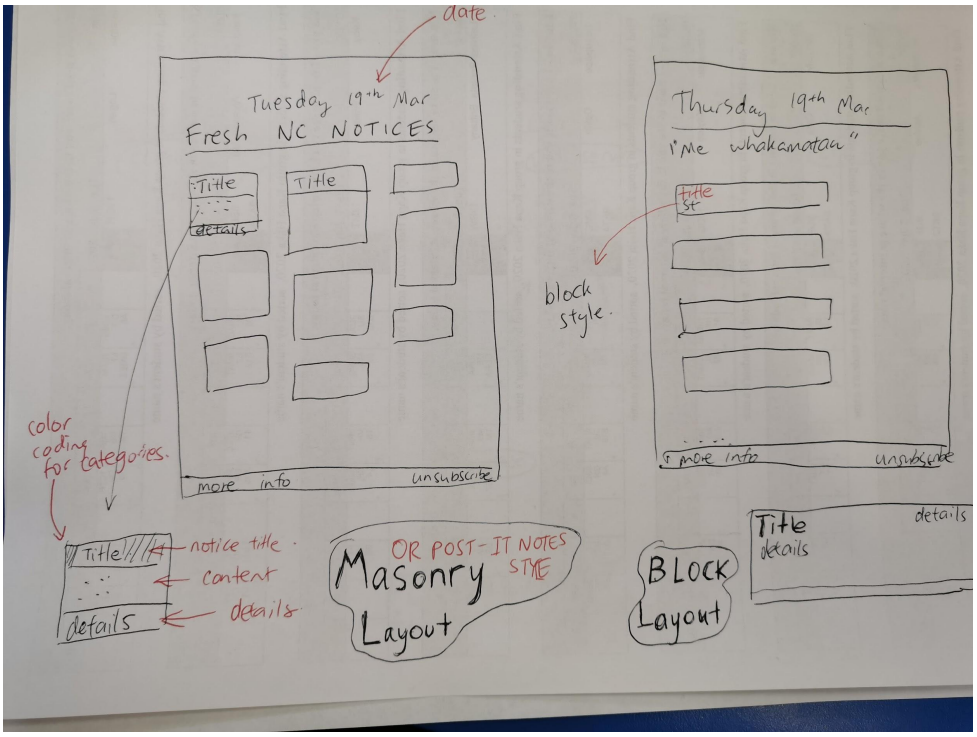
It would run off a server, and could be scheduled to send out the emails every morning, automatically. The newsletter would be formatted on this server, and sent out to a mail list from a database.. Students could sign up via a website connected to this server, and their email would enter into the database. This server could be potentially hosted in the school, or in the cloud, but at this point, I do not know. What I do know is that the ownership of this server will be easily transferred. Here is a diagram showing the process:



Reasons behind this design

I hoped by integrating the notices with email, it would reduce the cognitive friction required to check them. Students could check all of their school related content in one place. It also meant that the reminder to do so would be constantly there, as many students keep their emails open throughout the day.

I then could reformat this newsletter to have a high emphasis on readability and functionality to the student. These are a few initial plans of what it would look like. These are extremely rough plans, however.



It would have a vertical reading pattern, have contrast between elements of each notice, and use colour to create a structure. The reason I have only pursued a rough plan, as it is important to have a working prototype of the web scraping before we advance any further in our design — a proof of concept. This would reduce wasted planning as I know the limitations of the concept before I start.

Meeting with programming club

To begin the app development, I first had to identify how I would implement the idea. Having a concept was fine, but putting it into practice was another thing. Thus, I turned to my peers in the programming club for suggestions. This would allow me to get a variety of opinions from a wide range of different skill sets, beginning the project on a firm footing. This would also reduce the risk that I began the project in a 'dead end' manner — using languages that would be far too difficult and/or expensive to maintain.

How the web scraping would be implemented

One student who had prior experience in web scraping suggested I use the 'Beautifulsoup4' (bs4) module in Python. He also said I could use 'jsoup' from Java, but that would mean I would have to learn Java as well as learn how to use 'jsoup'. Thus, I concluded that the core of my project would be built with bs4 as this would mean I could use Python, which I already had prior experience with. I also knew Python would be popular and thus easy to set up on a web server — compared to java, which was a little more difficult.

How the email newsletter could be constructed

Another encounter with a student allowed me to realise I could use HTML5 and CSS within an email which was unknown to me. Before this experience, I was considering generating a pdf or image to send out, but this completely changed the direction of my project. Immediately, I realised the convenience this would provide (images are large and cumbersome to produce programmatically) and I decided the email newsletter would be written in HTML and CSS.

Brief insight into how I could deploy the application

I also briefly asked about how you could use form submissions in a website and received some short responses. I knew you could use PHP for the submission and SQL to construct the database, but not much more. I figured that this wouldn't be too hard to make and I should focus my attention on constructing the prototype product first — before focusing on the deployment of the app.

Proof of concept

I began constructing a proof of concept first, to identify how my project was to run. I knew I would be most likely using bs4, a web scraping module, however, for the remainder of the program — I still needed to figure out.

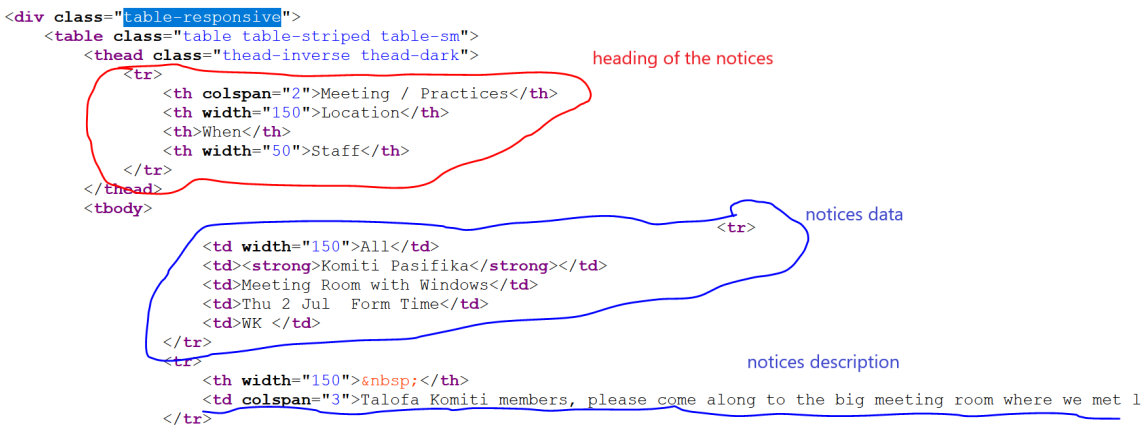
Constructing the web scraping component

Isolating the notices data from the webpage

I first did some research online and realised I was actually wrong about "bs4", as it didn't web scrape, but parsed HTML (given web code, it turns it into a data structure that is program friendly). I needed another module named "requests" to download the webpage from a given URL.

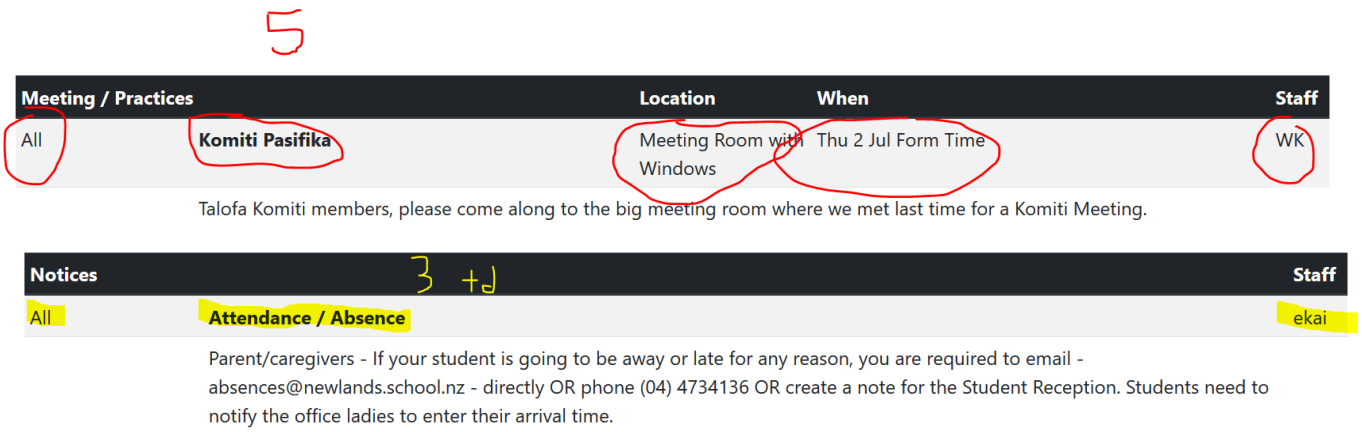
Therefore, to begin the program. I would use requests to download the HTML code from the parent portal URL — parents.newlands.school.nz/index.php.

Once I had downloaded the HTML code from the notices page, I then used bs4 to parse it so that the program could understand it easier. Specifically, I wanted to isolate the data in the notices for the program to process.



Upon looking at the code above, I noticed that all of the notices were encapsulated in <tr> tags. However, not all <tr> tags were necessarily notices. The notices were specifically the <td> tags within the <tr> tags.

I wrote the program to look for only <td> tags within each <tr> tag, and printed the output to see what would happen. It became quickly apparent there was an issue: the number of <td> tags in a notice was varied: either 0, 1 3, or 5.



Notices with 5 <td> tags were classified as meetings/practices and notably, had a *time and location.* Notices with 3 <td> tags were classified as *notices*, and did not have a time and location. In fact, they can be observed on the notices page quite clearly. Notices with 0 <td> tags, obviously, were not notices. Notices with 1 <td> tags were *notice descriptions that* belonged to the <tr> section that preceded this notice.

Therefore, using the td tags as identifiers — I was easily able to pair the notices into categories and I placed these into separate lists — notices_5 and notices_3.

Each element of notices_5 / notices_3 is an array that holds information about the notice (e.g. date, author, description, etc). An example is shown below

```
Notices 5:
['All', 'Komiti Pasifika', 'Meeting Room with Windows', 'Thu 2 Jul  Form Time', 'WK ', 'Talofa Komiti members, please come along to the big meeting roo
['All', 'Poly Club', 'GDRA ', 'Sun 28 Jun  4.00 - 6.00', 'WK ', "Talofa Lava, Malō e lelei, Kia Orana, Taloha ni, Ni sa bula vinaka, Fa'aka alofo lahi
['All', 'Student ID cards', "Bursar's office", 'Wed 24 Jun  interval & lunch time', 'ekai ', 'Students - if you require your student ID card, check wit
['Juniors', 'L1,  L2, L computer pod usage', 'L1 & L2', 'Thu 25 Jun  ', 'WA ', 'L1, L2 and L computer pod are special classrooms so no Year 9 and 10 st
```

Notice = Array containing information about the notice. Notice[i] = ith element of each notice array.

Notice[0] = Who the notice is for.
Notice[1] = Title of the notice
Notice[2] = Where the event is held
Notice[3] = Time and date of the notice
Notice[4] = Who uploaded the notice
Notice[5] = Description of the notice

Emailing with Python3

Deciding which emailing library to use

I now had to figure out how to email using Python. I had come across a few options whilst researching before, and these were Flask-Mail and “smtplib”, both using SMTP (simple mail transfer protocol) to send mail. Flask-mail was an extension of Flask, a framework written in Python for building web apps. As I had decided to build my web app at this point in time with PHP, I decided to not use this option. Thus, I chose “smtplib” due to its ease of setup and how it didn’t require the use of Flask.

I had created a burner Gmail account, and this would be used as the temporary pilot for the project. In the production version, I plan to use an email set up by the school (with an internal domain).

SMTP essentially acts as a server to which I can send my mail to, and it will reroute it to the recipient. Each email provider has its own server and this has a unique port number. For Gmail, this was smtp.gmail.com with port=587. The benefits of using SMTP was that I didn’t need to login to Gmail manually and could automate the emailing process using Python — a major benefit.

How I would structure the emailing process

For the email recipients list, this was also going to be a list. I would loop through this array and send an email to each one. Notably, in order to optimise the application — two key things would be done.

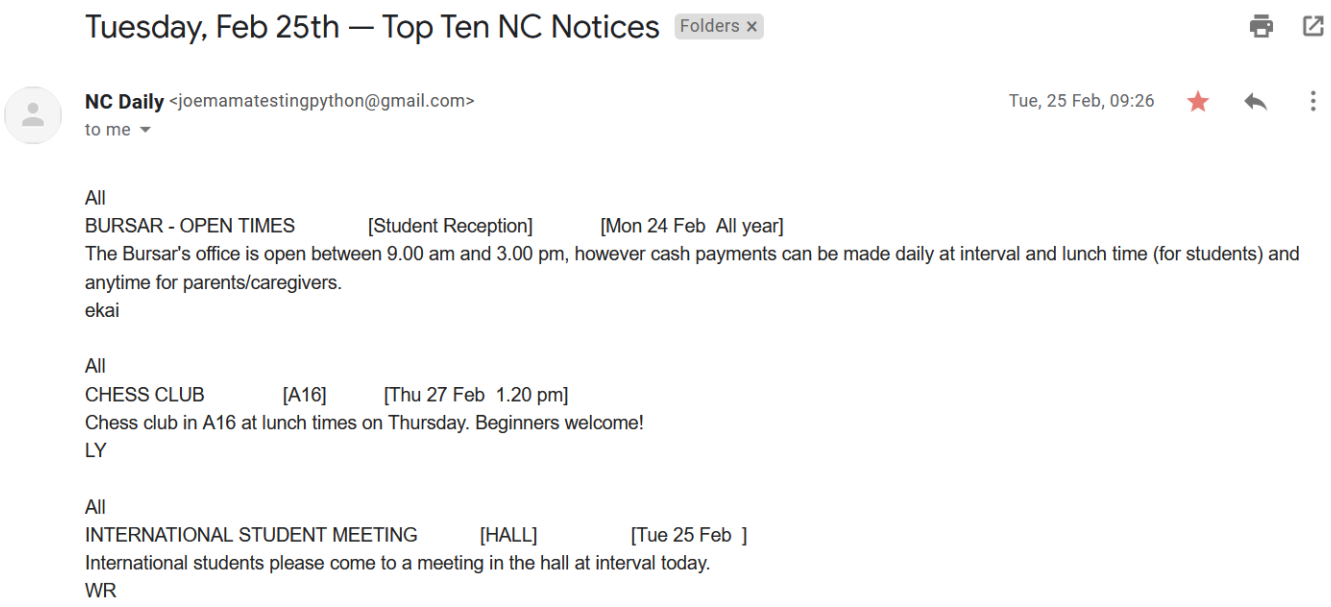
First the email newsletter (the notices) would be created outside of this loop — meaning everyone gets the same newsletter but also saving an immense amount of time (as the newsletter doesn’t need to be remade every loop). The only drawback of this at the moment is it doesn’t allow for personalisation of the newsletter. However, that is not a stakeholder criterion (in fact, it goes against one — as the email newsletter should have consistent school colors), it means this drawback is fine.

Secondly, each email (header + body) object would be deleted after it is sent. This is to prevent a memory overflow, whilst simultaneously speeding up the program.

Styling the email

I would include the date into each notice newsletter — to allow the user to easily identify which day the newsletter belonged to, and shorten it to “NC Notices”, as this sounded better than (and read easier than) “Newlands College Notices” in the context of an email.

Finally, I quickly took the notices I had web scraped before using bs4 and used smtplib to email it to my email — the first proof of concept.



Tuesday, Feb 25th — Top Ten NC Notices Folders ×

NC Daily <joemamatestingpython@gmail.com> Tue, 25 Feb, 09:26

to me

All
BURSAR - OPEN TIMES [Student Reception] [Mon 24 Feb All year]
The Bursar's office is open between 9.00 am and 3.00 pm, however cash payments can be made daily at interval and lunch time (for students) and anytime for parents/caregivers.
ekai

All
CHESS CLUB [A16] [Thu 27 Feb 1.20 pm]
Chess club in A16 at lunch times on Thursday. Beginners welcome!
LY

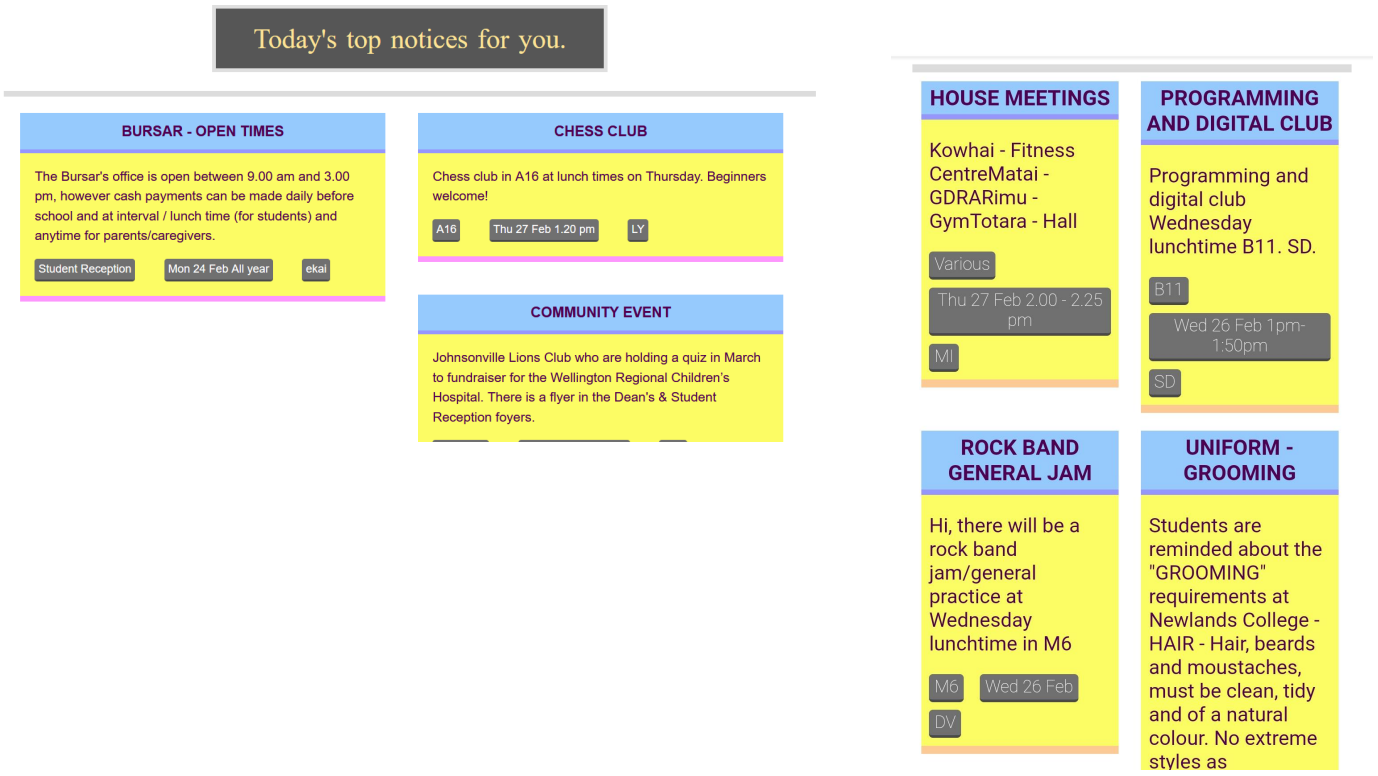
All
INTERNATIONAL STUDENT MEETING [HALL] [Tue 25 Feb]
International students please come to a meeting in the hall at interval today.
WR

Now that there was a proof of concept — the path to implementing this into a reality becomes more clearly set. I would need to:

- Produce and finalise the design of the newsletter
- Identify how to automate the process
- Create the signup website
- Conduct preliminary tests
- Launch!

Designing the newsletter layout concept

Sticky note layout draft



I first began to explore the sticky note style layout. I hoped it would make color coding easier and scanning for notices easier. However, creating the layout proved more difficult than I thought. In fact, creating an HTML email had several limitations I was not aware of.

HTML email could not use external stylesheets, advanced CSS elements (hovers, animations) and had to use inline CSS. This means the development of time of the HTML email was more time consuming, as I was writing code in a non-conventional manner.

Furthermore, CSS grid was not supported — a large concern as it was critical in my implementation. CSS grid is a CSS feature that allows dynamic rectangular designs — especially helpful in this scenario.

HTML email also didn't allow for media queries. Media queries allow the newsletter to be adaptable and able to change depending on screen size of the user.

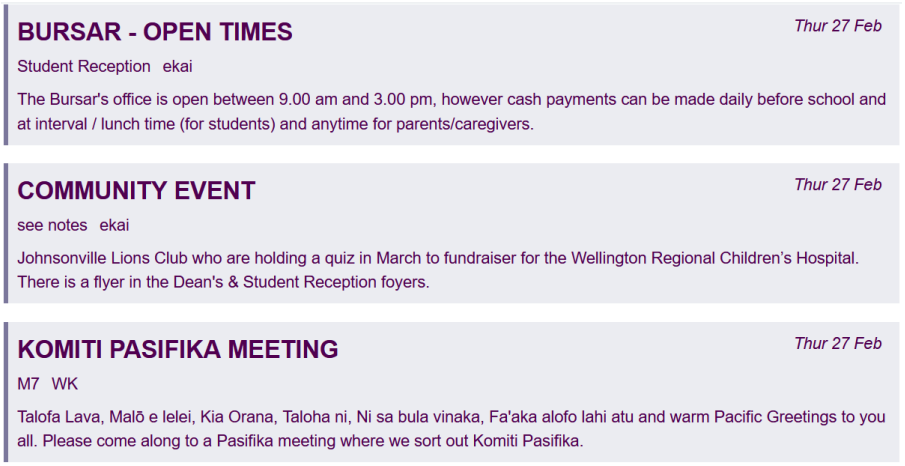
Overall reflection on the draft of this design:
While a promising concept, it didn't stand the test of implementation and failed to deliver. The mobile design is arguably worse than the current design, and the sticky note idea doesn't land. It could've been pulled off if it was a website, but unfortunately, due to the hidden limitations of HTML email — I believe this idea doesn't work too well.

Block design draft

This time, after reflecting upon the problems encountered within the first design, I would focus on responsivity and contrast between elements — using simple design.

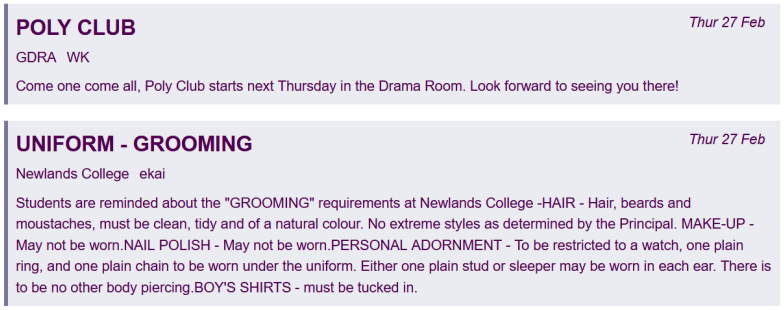
For the second design, I went for a block layout — similar to the successful layouts already in use in Reddit and Facebook. I have put the details of the notice on the edges to try and minimise the attention they drew, and ensured the notice title was the primary focus of attention.

Desktop layout

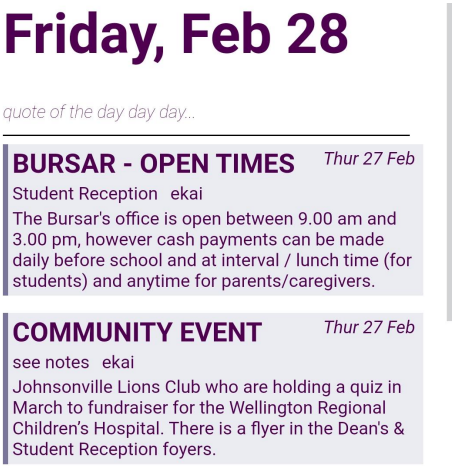


Already, this layout was far more readable than the sticky note design. It was clearly laid out, and had potential for *subtle* color coding (the left edge of each notice). Furthermore, because it was very simple, it used basic CSS and thus was supported on all platforms. Finally, the layout was consistent with varying size notices — something the sticky note platform didn't achieve.

Scales with different amounts of text

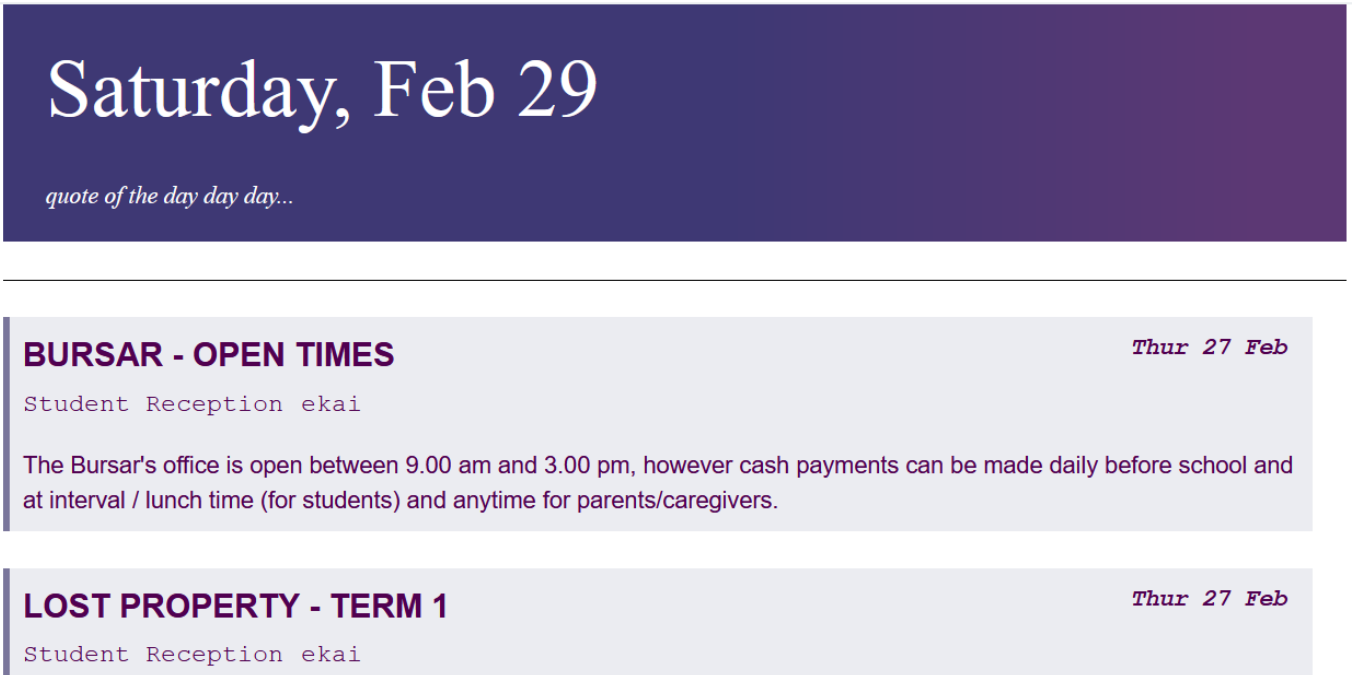


Mobile view.



The layout looked very nice on mobile, easily responding to change whilst remaining fully readable. Thus, I chose the block design as my final design concept to continue with on the project.

Designing the final concept

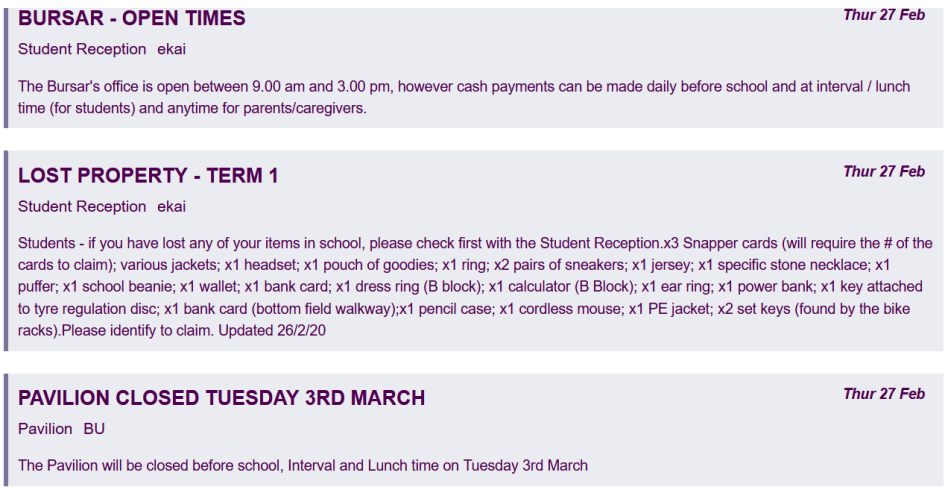


Header of each notice

The banner of the page would be purple as purple was a key color for our school. I added a slight gradient to this so it was more interesting — as a solid purple banner felt too desolate. This was also when I added a quotes section underneath the date to fill in space. It was a potential idea I could explore in the future — as I realise this would make the notices more interesting and unique each day.

Choice of fonts

The main text of the newsletter would be Arial as it was both ubiquitously supported on all platforms and highly readable. The secondary font would be a monospace font, “Courier New”, which was also a web-safe font and supported in all platforms. But, the main reason behind picking a secondary font was to create **typographic contrast.**

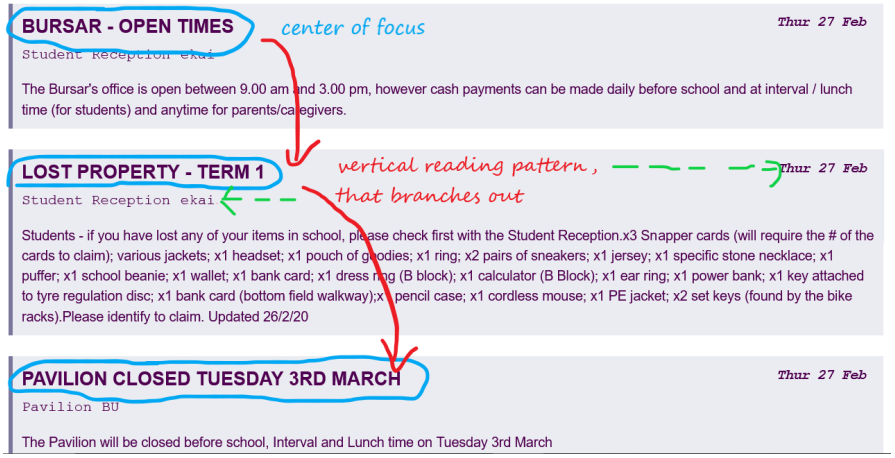


Without this contrast, the details of each notice interrupt the reading flow of the user

If the entire notice was Arial, it looked very dull to the eyes and according to one peer, “was like reading through a wall of text”. By having different fonts present, with different weights, stylings, and size, there is sufficient typographic contrast to maintain a high readability.[1]

Positioning of elements within each notice

On each notice, the main information would follow a strictly vertical pattern. The details of each notice would be in a smaller, lighter and different font and deviate from the vertical pattern. The effect of this is the reader can scan downwards, quickly picking up vital information. It is well understood by User Experience (UX) researchers, that when we scan for information in general — we read in an F pattern. Thus, by having the majority of the content in the left hand vertical column — it is already intuitive to follow. [2]



If they happen to find interest in one of the events, they can quickly scan horizontally and pick up this additional information — without interrupting the reading flow. These would be the branches on the F shaped reading pattern.

Overall reflection of the block design

This design is simple yet highly readable, which is something that I highly value. If we are to try and get more people to read the notices, readability, and thus user friction, is something we *must* minimise. By achieving this in an easy to execute design, it allows it to thrive in a multi-platform HTML environment, meeting stakeholder expectations.

Addressing the problem of duplication

Once I had a final design and a proof of concept, I set out to solve the problem of duplication within the notices system — another stakeholder requirement. With the data I had scraped, I had to figure out how I could identify which notices are duplicates and which aren’t.

---

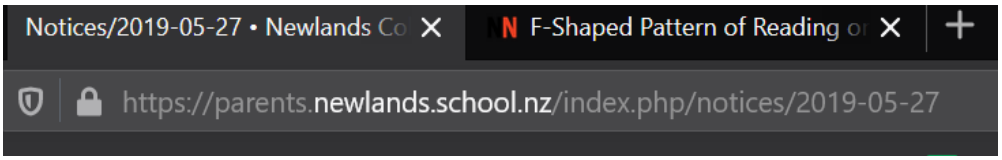
[1] "dot-font: Seven Principles of Typographic Contrast ...." 28 Jul. 2003, https://creativepro.com/dot-font-seven-principles-of-typographic-contrast/. Accessed 15 Jul. 2020.
[2] "F-Shaped Pattern of Reading on the Web ...." 12 Nov. 2017, https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/. Accessed 15 Jul. 2020.

Deciding what was considered a ‘duplicate’

The criteria I had settled on to identify an ‘old’ notice was if it appeared yesterday and appears in today’s notices. I considered extending this to a longer period of time, but realised this is simply an overcomplication.

Implementing a sorting mechanism

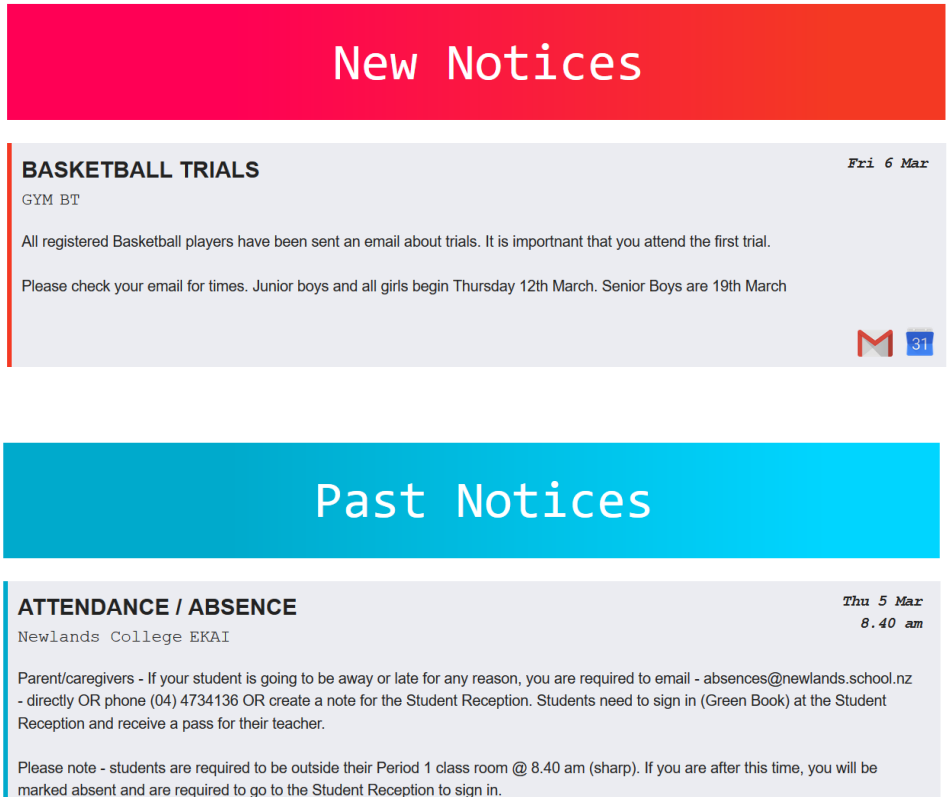
I noticed that our parent portal allowed access to past notices by simply appending a date after the notices page URL. The notices page acted almost as an API to access archived information. Thus, everytime the app ran — I would scrape in addition to today's notice, yesterdays.



New notices were notices that had titles which didn’t appear in yesterday’s notices. Old notices were those that did.

The reason I had compared the titles of notices is for efficiency. If the cross checks were done with all the other information, it would increase the search time erroneously.

Therefore, as I could now distinguish between new and old notices, I have used color coding to create a structure within the design. Here is what the iterated product looks like:



Choice of colors and names

The colours for new and old notice are red and blue, respectively. I had consulted with a design student, who is experienced in color choice and he gave some invaluable input. Red is a color that is strongly connotated with hot, new things. Blue is a cool complement that aesthetically matches the blue, and therefore its meaning will be the opposite of red: colder things. This is an important choice as these colors have **intuitive meanings** that any user will understand, reducing the cognitive load the user must take on to read the notices.

The colors red and blue also happen to be school colors — and this helps the newsletter maintain an identity.

Consideration of misrepresentation

In addition to this, I decided to rename the label ‘old notice’ to ‘past notice’ as the word old has a negative connotation. If I was to misrepresent these ‘old notices’, it could mean that students read even less notices — having an opposite effect to what I intended. While this may be a small detail, I believe it is very important.

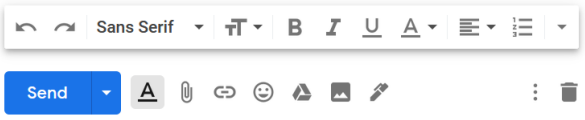
I did not want to portray the old notices as ‘inferior’, as the old notices still contained important information to reference. Thus, the word ‘past’ is used instead.

Adding extra features

However, even after adding the structure to the notices newsletter — it still felt incomplete. Looking back at all the information I had scraped, I was sure I could do something with it that would add value. Two fields that within the data scraped of note were the **author** and the **time and date.**

Creating an email feature

Within my school, teacher email addresses were formatted in the following manner: a two character teacher code followed by the domain “@newlands.school.nz”.



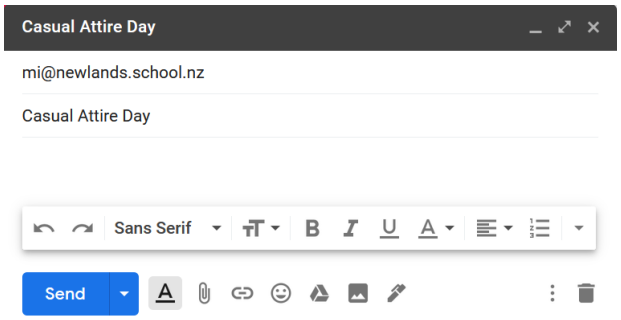
For example, John Smith could be abbreviated to “js@newlands.school.nz”. Very interestingly, the author field in the notices was also a two character teacher code. Thus, I could then use this to create a **mail to link**. This would allow the student to quickly email whoever made the notice and ask questions.[3]

```
mailto:mi@newlands.school.nz?subject=Casual Attire Day
```

For convenience, I have included the subject of the email too (as the title of the notice).

---

[3] "Mailto Link Syntax: The Complete Guide - Yoast Developer ...." 27 Oct. 2008, https://developer.yoast.com/blog/guide-mailto-links/. Accessed 16 Jul. 2020.



For example, if the link above was opened in Gmail, it would lead to the following window on the left to appear. Thus, I have added this link to each notice, incorporating an email feature into the newsletter.

Linking Google calendar

Furthermore, I had wondered if I could somehow connect Google Calendar with the notices. Students usually add events to their calendar to stay organised, and if this was possible, it would be a major improvement. After some research on the web, it turns out you can do this with Google Calendar template URLs. [4]

The template URL however, required some difficult elements of information in specific formats. Notably:

- The date must be in a 8 character format (YYYYMMDD)
- The start and end times must be in a 6 digit format (HHMMSS), in UTC time and cater for daylight savings.

```
string = 'http://www.google.com/calendar/event?action=TEMPLATE&dates=
{eightdigitdate}T{sixdigittimeUTC}Z%2F{eightdigitdate}T{sixdigittimeUTCend}Z&text=
{title}&location={location}&details='      You, 25 days ago • Initial commit
```

In addition to this, the title of the event and their locations needed to be provided as strings — but these were relatively easy to obtain. The major problem was interpreting the time and date. Often, teachers would include times that varied in format.

For dates, these could include:

- Long format: Thursday 12th Jun
- Short Format: Thu 12th Jun

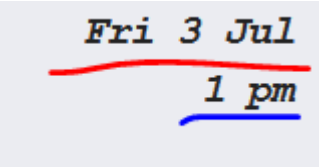
For times, these could include:

- Keywords: Lunch, interval, all-day, evening.
- Long Times: 2.00pm
- Short Times: 2pm
- Long Intervals 1.10pm - 3.10pm
- Long Intervals with ‘to’: 1.10pm to 3.10pm
- Short Intervals with to: 1.00pm to 3.10pm
- Long short mixes: 1pm to 3.10pm

[4] "What parameters are required to create an "Add to Google ...." 29 Apr. 2016, https://stackoverflow.com/questions/22757908/what-parameters-are-required-to-create-an-add-to-google-calendar-link. Accessed 16 Jul. 2020.

Initially, this was a daunting task. I wondered if there was a way to achieve this with artificial intelligence but quickly realised this was beyond my scope of knowledge. However, I noticed that there was a pattern, and by covering each branch of possibility — this you could properly interpret the times. Several strategies I used to normalise the data obtained are:

- The first three words of the time and date are always the date. E.g Thursday 12th Jun and the rest of the words would belong to the time.



Red = date component (always 3 words long)

Blue = time component (variable length)

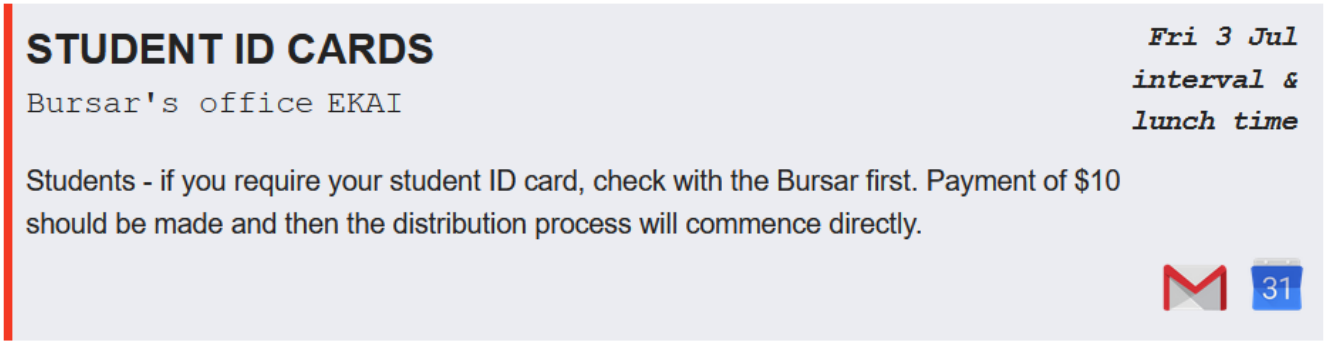
- I could use a decimal to encode a time, where the whole number component would be the hour and the decimal component would be the minutes, without switching to base 60. I.e. 12.40 == 12pm and 40 minutes. This method was chosen because of its simplicity over using complex date objects which would make the code very difficult to read. Python datetime objects, which are reliable and accurate, require long setups (e.g. datetime.datetime(year, month, day, hour, minute)) and as I was going to write numerous IF statements, this would complicate the code very quickly.

- I could search the time component to look for certain keywords, such as ‘lunch’, ‘interval’, ‘morning tea’, ‘period 1’, ‘period 2’ and assign these to specific times.

- Use defaults to catch any time component combinations that would be unable to be interpreted. i.e for this app, the default calendar time would be 8:15am — just before school time starts — which was chosen as it is the best time to remind a student about what's upcoming in the day ahead

Once I obtained the starting date and time, I could add 15 minutes to it to get the final date and time and substitute these values into the calendar template link. This link, when clicked, would automatically add this event to the student’s calendar — with prefilled times and event names.

Following the previous conventions within the final design, I would place the email and calendar buttons in the corners of each notice, to not disrupt the reading flow.



Possible methods of deployment

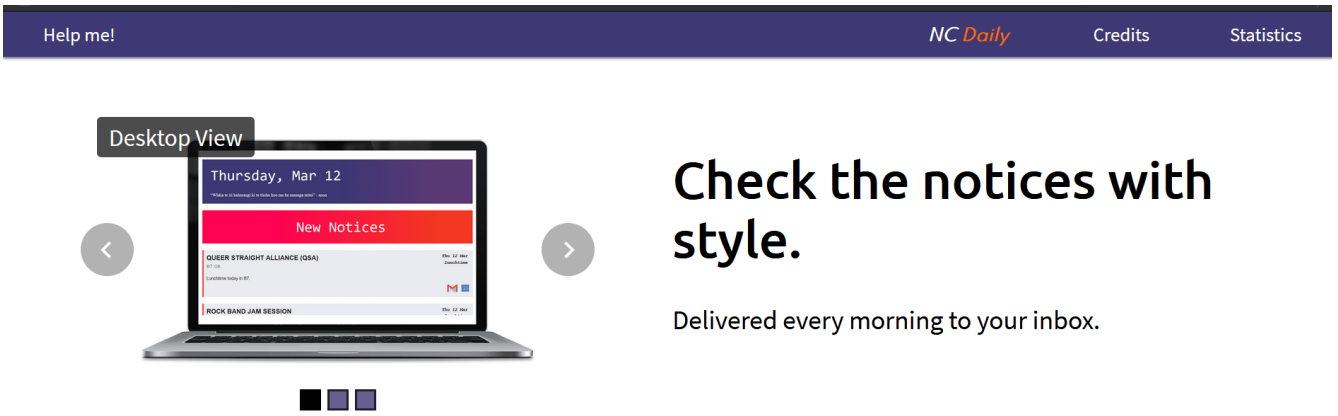
Now having a fine product is all and well, but being able to market and have people sign up (or off) to the service is another. The method I had decided on initially was through a **website** built with PHP and this could be connected to a MySQL database.

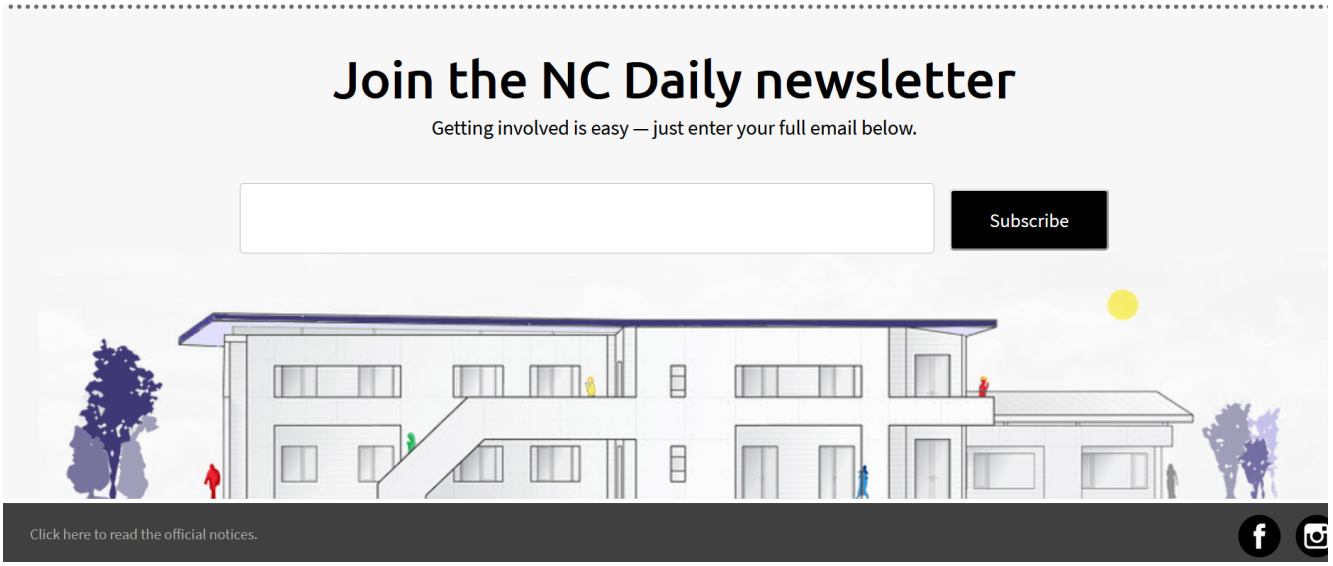
During my plan, I noted that the website should:

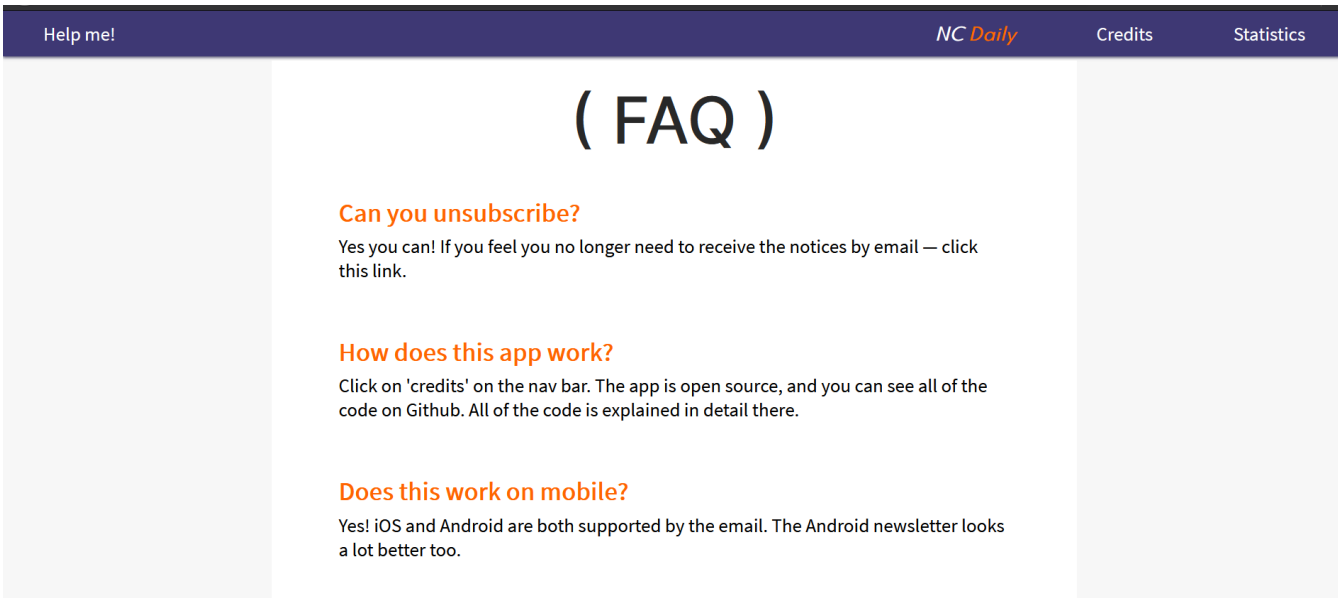
- Clearly outline the benefits of this new notices system.
- Provide an easy way to sign up **and** unsubscribe.
- Be easy to maintain (cost wise **and** time wise)

PHP website - first draft and proof of concept

I first developed the website locally on my own computer, and set out to create a simple signup form which would prove the application worked. This wasn't too difficult to achieve, and so far the website was going well. Here are some preliminary screenshots of the website







To explain the purpose of the application, I would show them photos of the new notices system in action and on different devices. It explained the purpose behind the application and convincingly showed why you should sign up.

The only problem was hosting. I couldn't find any reliable, cheap hosting services online, and this did not meet my criteria. I could find many cheap (or even free) services, but these were located in distant countries and also were difficult to use (constantly paywalling important parts, or predatory marketing) and I did not want to meddle in such satanic rituals.

Attempt at self hosting

Thus, I turned to the solution of self hosting — or setting up a server within the school. I learned that a Python script (or any executable file) could be automated to run every morning using Windows Task Scheduler. The problems I encountered were:

A. The app wasn't very portable, depending on multiple external packages and couldn't run on school computers
B. Hosting was surprisingly complex and wasn't one hundred percent reliable.
C. I still needed a custom domain, which would cost extra money.

Initially, I had overlooked problems B and C — as I set off to solve problem A. It should have been clear from the start self hosting was an unviable solution, but this is only realised in hindsight. To make the app portable, through some research, I had figured out how to compile and run the program on any Windows platform with PyInstaller. [5] Then I spent several weeks trying to figure out how to set up a server on the computer and didn't get very far.

In hindsight, I should have followed similar procedures in the past by extensively researching and creating a minimum viable product first. This approach was scrapped, obviously.

[5] "PyInstaller." https://www.pyinstaller.org/. Accessed 16 Jul. 2020.

Google form website first draft

Then, I attempted to explore using Google Sites to create the website. The subscription method would be through a Google Form and a Google Sheet would be the database. The Google Form would allow for ‘responses to be edited’, allowing students to unsubscribe easily. I even could use the Google Sheet to produce charts and graphs.

However, quickly, the largest problem was even though the Google Sheet offered the structure of a database and stored the information very well — **Google did not allow for external programs to manage data within a sheet.** Thus, the apps database content would need to be added to the applications local database **manually**, every time someone signed up for. This did not meet stakeholder criteria of being fully autonomous.

Decided method of deployment

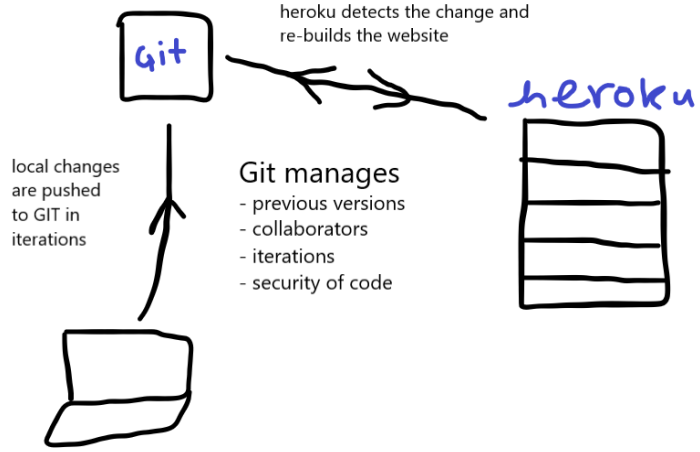
Thus, I took a break from the project — after multiple failed attempts in finding a suitable deployment problem over the lockdown due to COVID-19. However, during this period, I thought back to a very interesting framework I had encountered earlier — the Flask framework. It was a simple, lightweight framework that allowed for a rapid development. Learning from the previous examples, I quickly researched if there were suitable hosting providers that hosted Flask.

Surprisingly, there were many, and they were all of a low price ($3 / a month etc). However, one hosting provider was notably better than the rest — Heroku. Offering 1000 hrs of “dyno hours” a month, for free, the app could essentially run indefinitely at a cost of $0.

Decided method of project management

Heroku offered several methods of project management. One was a simple local file based method, in which the project would be linked to a folder on my machine. This can then be copied over to the cloud using the command line using the Heroku CLI (command line interface). Another method was simply uploading the files of the folder manually, each time the app needed to be tested. The benefits of the CLI method is it is relatively easy, where the file management is easily copied over, and is faster than manually uploading it to the site every time. However, it does not address one key problem which is maintainability. The app will not be maintainable because the Heroku link is to my local machine, which will be unavailable once I leave school.

One final method which stood out was the use of Git. Git is a version control system which allows for users to manage previous versions of code, and collaborate in a cloud based environment. This would be helpful for future expansion of the project or backtracking to previous versions of the code in case anything went wrong.



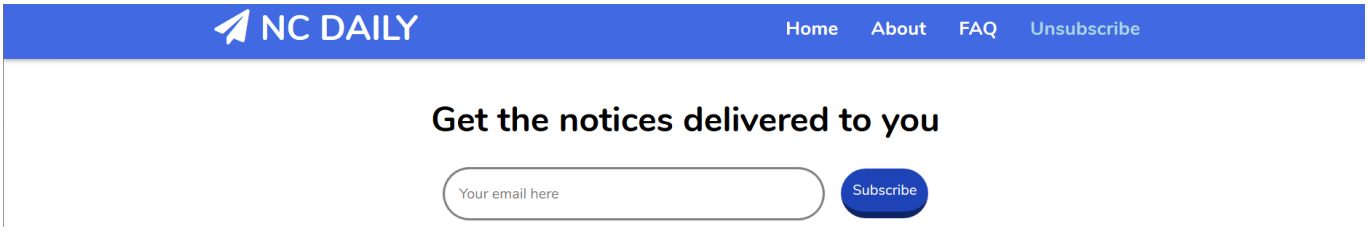
Git could also connect to Heroku, and it meant all project code would be easily transferred to the new owner. Furthermore, Git also allowed for automatic updates on the Heroku site as soon as any changes were made to the Git repository.

Thus, I decided to use Git as this method had benefits that far outweighed all the other

Flask website first draft

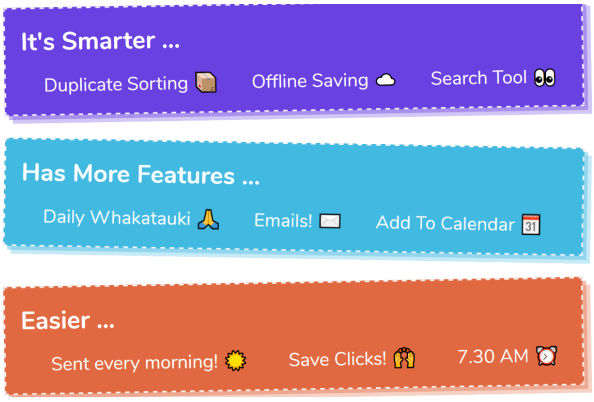
Design

The initial webpage was aimed to be simplistic in design. Featuring a bright colour scheme (that still incorporated school colors), and a single font type, the webpage was minimalistic in design. The sign up form would be on the front page, top and center. This made it extremely clear on what the website was about. Alternatively, I could use a more complex design. However, I believe that because of the nature of the website — being very simple itself (just for signing up) — there was no reason to complicate it further.



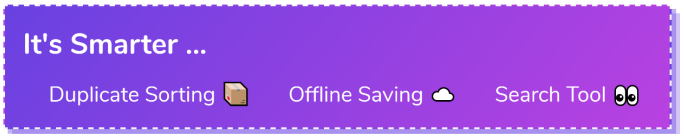
The reason a bright color scheme was used was to imbue the product with a sense of energy and ‘freshness’, something that they should try — to get them to sign up with. It is also realised that students (who are still kids), enjoy bright colors and this caters to the audience. However, an issue could be a lack of representation of school colors.

The crux of the home page was to convince the user to sign up by **showing** them what the app could do, but simultaneously, not providing **too much information.** Therefore, I constantly advertised what the app had over what was currently in action, but in a controlled manner.



I used emojis on the front page to provide variance to the content (adding images), convey meaning in a compact manner, but to also provide a relatable link to students.

These bars were interactive, hovering to expand and apply a gradient — adding to the user experience.



I hoped that this information would be enough to spark the interest of the user into signing up, getting their foot into the door and remaining a subscriber.

In addition to the home page, there were three additional pages: the unsubscribe page, the FAQ page and the about page. The about page contained more detailed information. The unsubscribe page had a single form which allowed users to unsubscribe if necessary. The FAQ contained, as followed, frequently asked questions.

Signup process

The signup process was a simple signup form. It would be server side checked (so that it is guaranteed to be input suited for the database), and it had to meet the following conditions:

- Is this a non blank string?
- Is this a valid / non-blacklisted email?
- Is this email already subscribed and present in the list?

If it passes all of these checks, it is added to the database, and the sender is sent a welcome message in the form of today's notices. This is to provide the user with what the product looks like, and will help reinforce their subscription while providing instant feedback to the user. I have deliberately avoided including email validation, as I will trust the students of my school to not sign up random people as they are representatives of the school. Also, if other students sign up other students, it is not necessarily a bad thing (as this is an improvement of the notices), but also boosts the growth of the app.

Unsubscription process

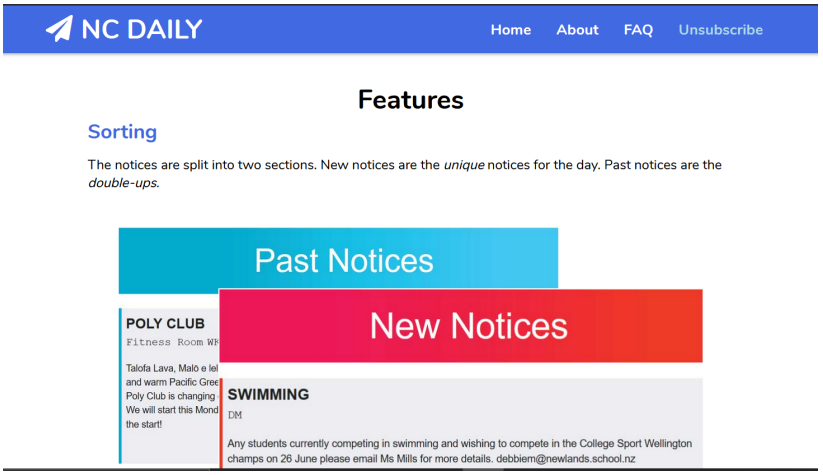
The unsubscription process will be similar, however, there must be a method of validating that it is actually the given person's email unsubscribing. Otherwise, it would be easy to unsubscribe strangers, and would be a large inconvenience. To unsubscribe, the user enters the email address they want to unsubscribe and a 5 digit randomised code is sent to the email address. If they enter this 5 digit code correctly, they are unsubscribed. This prevents unwanted unsubscription.

Student feedback on the first flask iteration

After releasing the first website to a small controlled group of individuals, my digital technologies class, they offered some feedback to me. In addition to this, I also showed a group of Year 9's the website, to gather feedback from a different demographic.

Pros	Cons
- Colour palette was refreshing and engaging - Signup and unsubscribe process very streamlined - Abundance of information available, if further questions were present.	- The about page is clunky, and students often asked why it wasn't merged with the home page so more users could see more. - Disliked the button structure, it wasn't very modern. - At times, some Y9's further asked why they should bother to sign up, so could be a sign that persuading them with just reasons wasn't enough - Not much interactivity present on website

In hindsight, the about page is rather clunky, with large pictures and being on a separate page. It appears through this informal survey, this is a negative aspect of the design.



Furthermore the button design did not appear to meet expectations. I thought that the design was very modern, with a 3d shadow. However, it appears to be not, as many people pointed this out to me.

Client feedback for first the iteration

I also obtained feedback from the client regarding the app. Surprisingly, the client said the app's design was fine and it was suited to its audience. However, I knew this wasn't the case as I had asked students what they thought of it and there were a few things that needed work.

The client also said the colour scheme of the app is fine. He said it was only a small part of the school digital system, and the colours were suited well.

He was slightly concerned about the signup process, and I did understand his concern. However, surprisingly, it wasn't regarding the validation process. He said there was a potential problem in someone signing up the whole school as I had not taken into account the school groups system (all@newlands.school.nz, is an alias for the whole school, and many more aliases existed). This would be a big problem, and will be addressed in the second iteration.

The client supported the lack of email validation upon signup. He said it was not a large issue and didn't see it as necessarily a bad thing, students signing each other up.

Reflection on the feedback for first iteration

The feedback on the initial iteration is mostly positive. The most important part of the site is working, which is the delivery (sign up and unsubscription process). However, what was lacking is how persuading the design of the website is. While I had aimed to omit too much information on the front page, it appears I had cut too much from it. Furthermore, there appeared to be a potential problem with the signup process if groups/aliases were used.

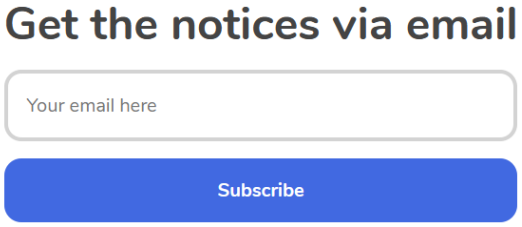
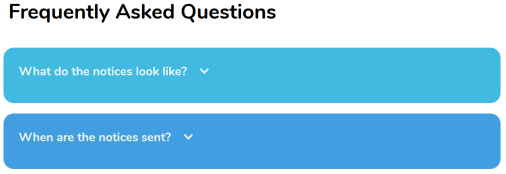
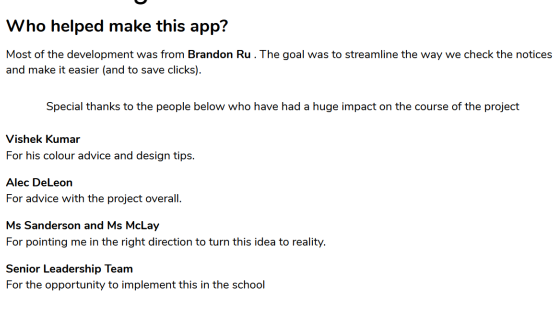
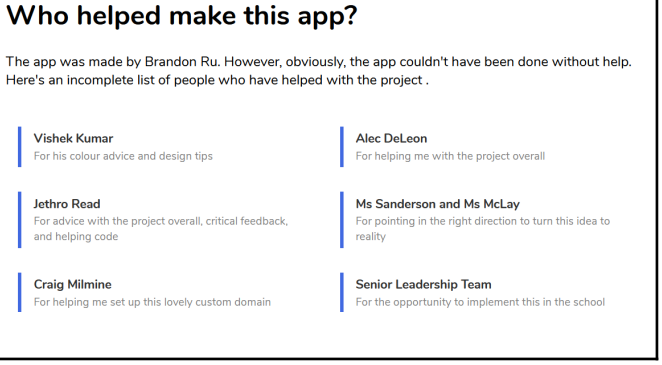
Next steps are:

- Increasing the amount of information on the home page and possibly removing the about page.
- Continuing to modernise the design of the website and add interactivity
- To explore new persuasion strategies
- Patching the bug, where students could sign up groups to the NC Daily website.

Flask website second iteration

Improvements on modernity of design

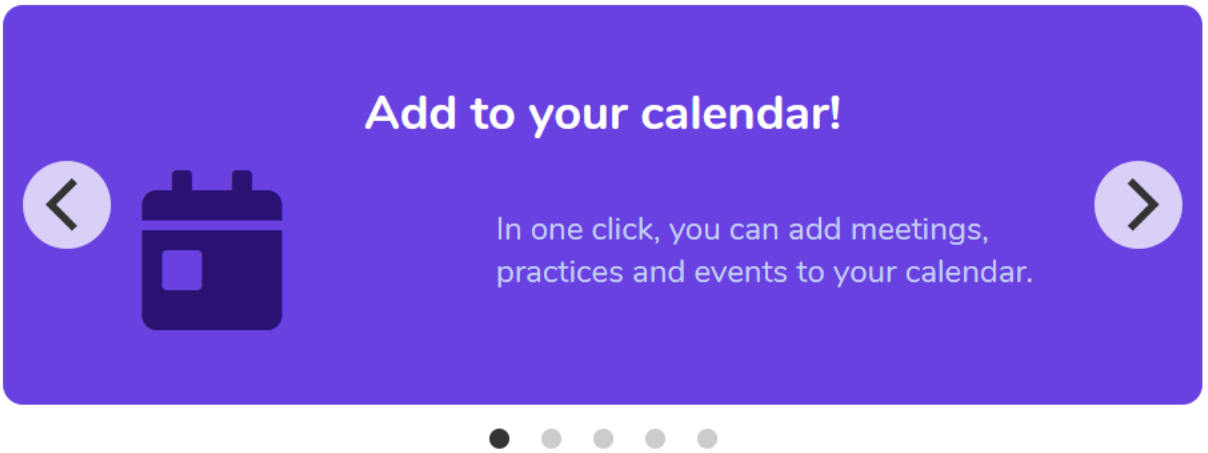
I continued to modernise the website by altering different elements to follow modern design conventions. Here is a list of designs altered to look modern and their before and after.

<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>Sign up button
 
</td><td> 
</td></tr>
<tr><td>Frequently asked questions section
 
</td><td> 
</td></tr>
<tr><td>Acknowledgements
 
</td><td> 
</td></tr>
</table>

The modernisation of the website is in response to the feedback from the first iteration, in which many students disliked certain elements of the design. These help improve user experience and make the website better, aesthetically.

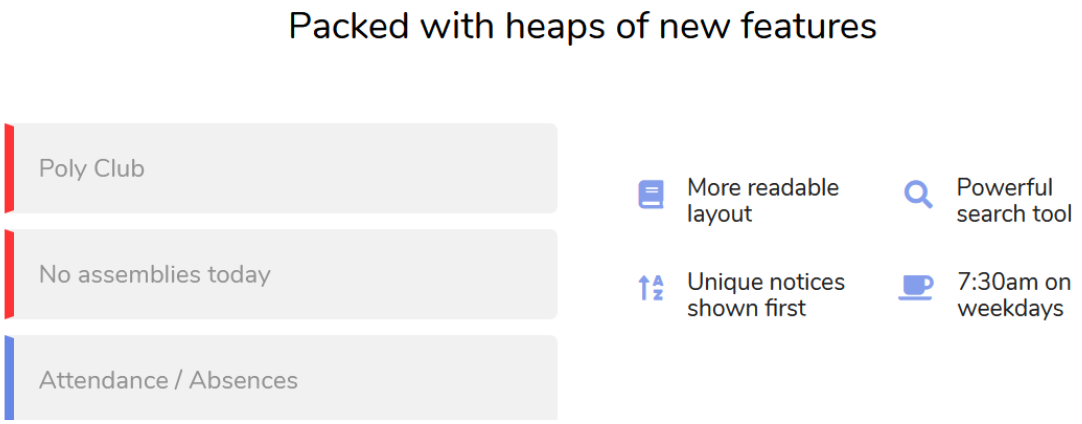
Improving the home page and scrapping the about us page

I decided to scrap the about us page, and to instead condense the information into the front page. This was achieved through an interactive slideshow located under the sign up form — an intuitive position.



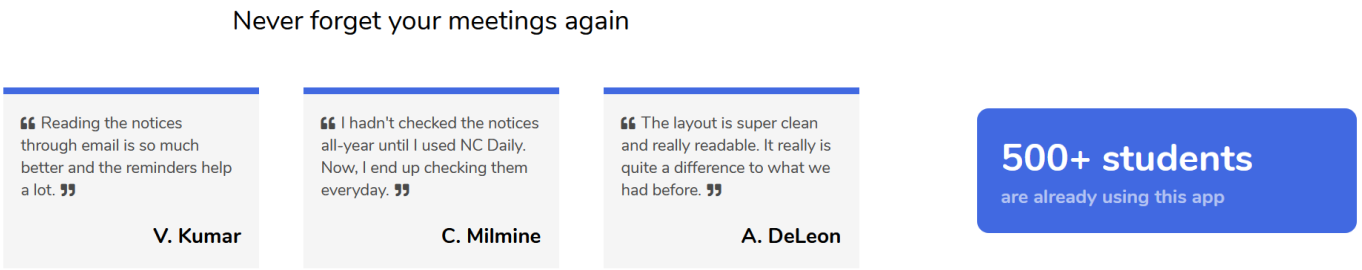
While featuring slightly less information (without live images), it could be argued it makes it more persuasive as the user is inclined to sign up to test it out. It continues to meet my design goal of introducing the app, without using too much information.

I also decided to include a more brief section that outlines the most important features. This will be for users who are skim reading, don't want to use a slideshow, or don't have a browser that supports it. These descriptions would also be interactive, changing color on hover and displaying more information. I would deliberately not use actual images of the products to maintain the theme of the website, but also to spark the curiosity of the user.



Including social proof

I also decided to include social proof, to make the website more persuading. Social proof is providing evidence that other people use this app and find it helpful too. I chose this method of persuasion as teenagers tend to be very conscious of their peers, and if their friends are using something, it often gives a lot of trust to the product too. Therefore, something as simple as including a live statistic on the front page, or including reviews (of students and teachers) on the front page can have a profound effect on their decision to subscribe.



Improvements on signup process

In order to fix the problem of unwanted signups of email groups, I have decided to address it with a blacklist. After consulting the I.T technician about this problem, it was noted that he has access to a list of all the aliases the school uses. After obtaining this list, I simply added another criteria to the signup process — that the email being added wasn't in this blacklist. This quickly solved the problem of group email signups.

Feedback of the second iteration of the website.

After asking the initial test group what they thought of the new iteration, all of them said it was a definite improvement. They said the theme was more coherent and they believed it was also more persuasive. The website was also much more concise in purpose, with the majority of the content on the front page, and the lesser importance content on the FAQ pages.

At this point I believe the website in terms of client side features is complete. End users are happy with the UX and the unsubscription process is also clean.

However, I have not addressed the maintainability of the app yet, and nor the automation of how emails are sent from the app. These are the next steps for the project

Sending of the emails from the app

The sending of the emails from the app is done with SMTP. The emails would be sent through Google's SMTP server. However, a potential problem I had identified with a larger scale mail out from the app is Google Servers has a soft cap on how many emails can be sent per minute. This was supposedly 50 emails per minute and proved a serious obstacle, as the app would break easily if it was simply let to run.

There were two possible solutions: the use of a BCC list and throttling the app. Initially, I thought the BCC option was brilliant — speeding the process up to one email, and sending all of them in one go. However, the problem was that google still counted BCC lists as emails and therefore the BCC would terminate in the middle before being able to complete and did not work.

Thus, I ultimately throttled the email sending process. Whilst I did not want to do this, after extensive research on Google Forums and stack overflow, there did not seem to be any other options. I settled on sending out 30 emails per minute. I have deliberately made this much lower than the soft cap of 50 emails per minute as I would like the app to run with 100% consistency, then risk the boundaries of the soft cap.

The costs of throttling the app were minimal, however. It did not use extra hours on the Heroku server (as the website was still running and therefore was considered running under the same hours). The app did not use much extra computational power either (as it was mostly idle time). I consulted with the client and they said this was fine. They agreed with my decisions and definitely said the app should run consistently, rather than with extra speed

Scheduling of the app

As the app must be fully automated, I have decided to use Heroku's built in scheduler to help with running the script on school days. The problem was, Heroku's built in scheduler only allows for the daily running of a script at a certain time — and I couldn't set it to specific time intervals.

The workaround was to introduce internal scheduling in the app. That is, the app will be run every morning at 7.30 am, but will not fully execute to completion until certain conditions are met. These conditions currently are:

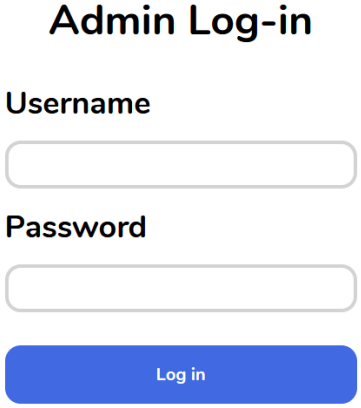
- If it is not a weekend

I also intend to add conditions that determine whether the date it was run on is a school term and also the app is on. These conditions ensure the app is run automatically, but only on the days needed.

Addressing maintainability

Of course, the app needed to be maintainable and at the moment it wasn't very friendly to anyone at all (except the end user). In order to delete emails, you had to manually SQL query, and navigate through wades of login pages. Thus, I decided to create an admin panel that offered an easy UX and UI to achieve these tasks. In addition to this, provide vital statistics / visualizations of the user of NC Daily. This is also a key criteria the client has outlined in the initial meeting, as the app must continue to run after I have left school.

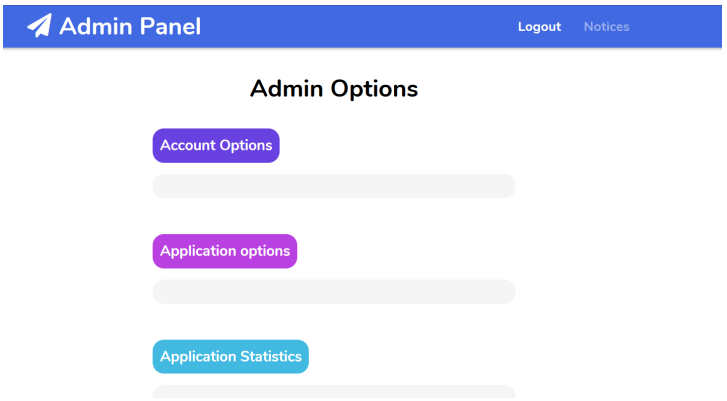
Creation of admin page



The admin login page will be located at the subdirectory /admin. This will have to be manually entered, and won't be a link on the main page. This is so that students are less likely to tamper with this page and cause potential havoc.

The passwords are stored with SHA256 hashes which are cryptographically secure and will be secure for the near future until quantum computing becomes cheaper.

Upon logging in, there will be a main page in which the entire list of things an admin can do will be present. At the moment it is empty but will be filled soon. There will be three main sections: account options, application options and application statistics. The account options will concern the admin account details. The application options will control critical components of the app (whether it is on etc). The statistics will display vital analytics.



Changing password

In order to change passwords, it will follow a similar method to most websites on the web. The user will have to enter their current password before their new one (twice), and then submitting. This is to prevent the password from being changed if they remain logged on accidentally on a machine, and to also prevent typos from occuring in a password.

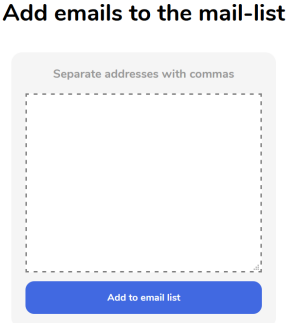
Logging out

The status of the user is managed by session cookies, which are encrypted pieces of local data that allow the user to navigate the website without having to login every time. Thus, the logout button simply makes this session cookie invalid, and therefore the user is no longer logged in

Disabling the app temporarily

If the app ever goes awry, or the app needs to be stopped, I have added a stop button. This stop button will toggle a global system boolean. If this boolean is true, then whenever the main app is run, it will exit, thus never sending emails out to students. This button only stops the emailing process, however, and not the website.

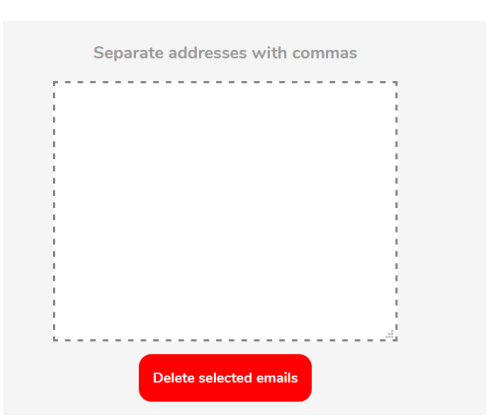
Managing who is subscribed

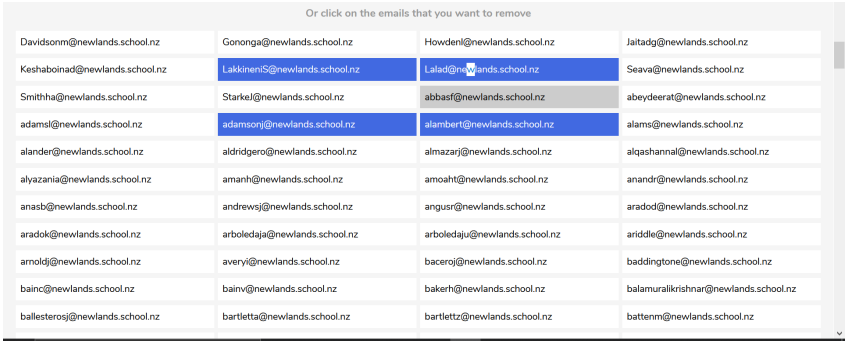


The admin will need to be able to add subscribers and delete them. I have decided to manage this via dynamic SQL queries. Subscribers can be added in a comma separated list, via the click of a button. The comma separated list is unpacked (separated), and substituted into an SQL query, and submits the emails into the database.

In order to delete subscribers, I have decided to include an additional graphical method. This is so the admin can either type in the emails they want to delete, or click on the ones they want to delete.

Remove from email list





This graphical user interface is achieved using Javascript and Flask. The Flask dynamically generates the page using a for loop to display the blocks. Javascript allows the buttons to paste their string (the email) into the delete email block.This graphical interface allows for a more intuitive management of the products subscribers, and also reduces the risk of error (mistyping emails.)

Managing when the app turns off

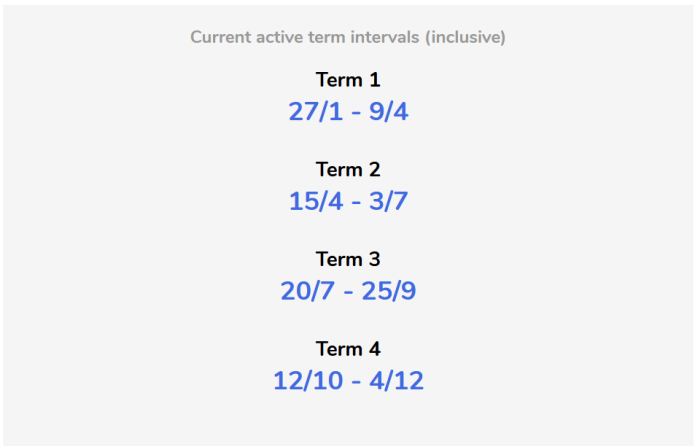
The app needs to be turned on during the start of school terms and off during the end of the term. Therefore, I have decided to run the app when the run date falls between a term interval. Otherwise, the app will not run. The app will also never run on a weekend (as this is always never school).

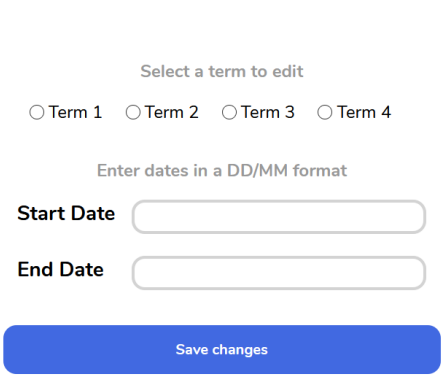
This is achieved by representing today's date as a tuple. If this tuple (x, y), where x = run month, y = run day, satisfies the following equality, then it must be during a given school term.

For each term (from 1 to 4):
(start month, start day) <= (x, y) <= (end month, end day)

2 dimensional tuples are compared on the basis of the first element then if a tie occurs (same month), the tie is broken on the basis of the second element (the day). Therefore, by having them in month, day order, the correct comparison can occur

Manage when the school terms are

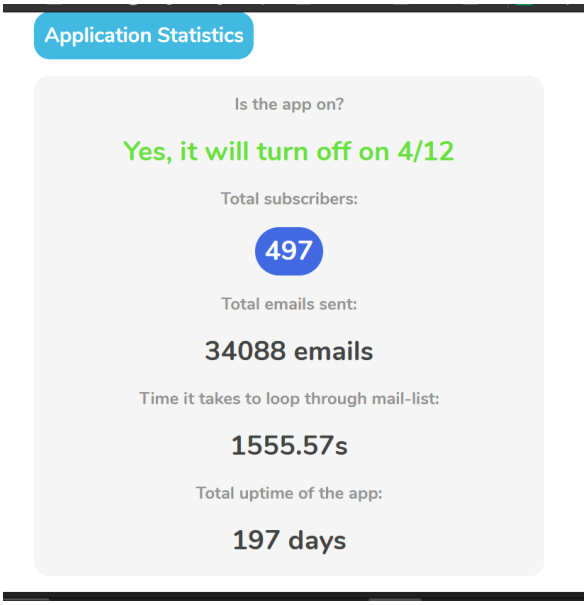




I have decided to do a two way graphical interface where the current intervals of school are displayed on the left and the editing can be done on the right. This was done so that the interface was more intuitive to use (as it provides instant feedback on change on the left), and also more compact (taking up one horizontal block on the screen). These term dates are stored into a JSON (Javascript Object Notation) file, where each term is represented as an entry into a dictionary and each term has a start/end value.

Creating statistics

Some important statistics I have decided to include are whether the app is on, when it will turn off next, how many emails have been sent, how many subscribers, how long the app takes to send all its emails, and the historic uptime of the app.

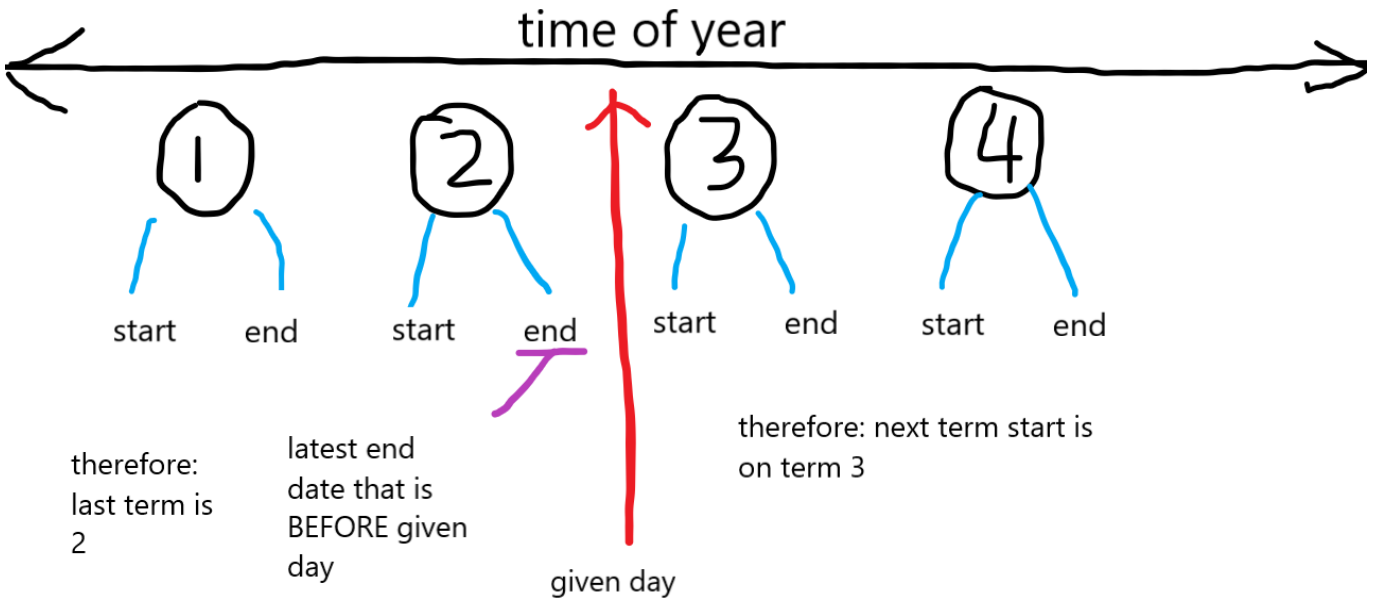


Is the app on, and if so when will it switch off?
To determine if the app is on or off — I will conduct all the preliminary checks of the app, and if they all return true, it is true. The preliminary checks done were:

- Is it run on a weekend?
- Is it run between school intervals?
- Has the app switched off manually?

If the app is on, I will fetch the next date it turns off. To do this, the app needs to know what term it is in currently. This is achieved by checking which term interval the date is between — by doing comparison checks between all term intervals. Then, once the term is identified — the next date the app turns off is simply the end date of this term.

If the app is off, I will fetch the next date it turns on. This is slightly more complex, but explained below. The app will determine the latest end date that has already occurred before todays' date. This gives the last term that had just finished. Then, all the app must return is the start date of the next term. Note this algorithm does not work for term 4 holidays, and simply returns "the app will begin next year" value as there is no "fifth" term. This is illustrated in the drawing below.



Total subscribers

The number of total subscribers is just the number of rows in the database of emails. This is returned by using the universal SELECT query from the entire table of emails, and then returning the length of the list returned from the selection.

Long term statistics:

To store more long term statistics, I have created a separate SQL table to store them. It will have columns of “total_emails_sent”, “uptime”, “running_time”.

After every time the app is run, the main script will increment the total_emails_sent by the number of emails sent. It is not done every time it is sent because that is inefficient, as connecting the SQL database and querying it requires significant energy. By doing it at the end of each session, it saves computational power and speeds up the app.

The uptime of the app is incremented by 1 after every day the app is run. The entry on the SQL table is incremented by one.

The running time of the app is determined by taking the difference of times between the start of the app running, and the time after the app has sent out all the emails. This is then updated on the table.

These statistics will be important to the client as they allow for performance to be gauged, and important lifetime statistics to be easily identified.

Launch day

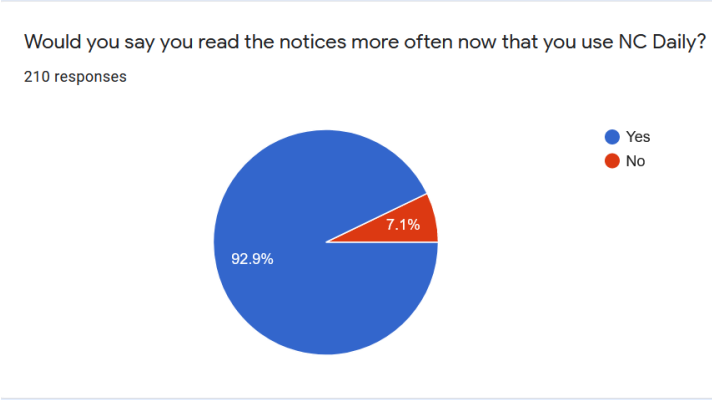
Now that the app is ready to release. I will do so via a combination of three methods: email, assembly presentation and social media campaign. I will present the app at assembly, but also email out information with the link to the website. In addition to this, release advertisements on the school social media account.

I have decided on this mixed style of release so that I reach a large proportion of the intended audience, whilst publicising the features of the app. If I was to simply email out a link, it would be expected that many would delete it as it was given no context. If I was simply to do an assembly presentation, the link would quickly be forgotten by most people. If I was only to do a social media presentation, only those on social media would be reached. Thus, by using a release strategy that utilise all three of these strategies, we had hoped for maximal success.

Within the first 24 hours, we had amassed 200 subscribers. By the end of the first week, we had 450 subscribers and the app was running smoothly. Students were reading the email newsletter all around school, and the general feedback was amazing. The client (ICT director) was happy with the results, and thought it was a smooth launch with no major issues

Reflection of the apps success

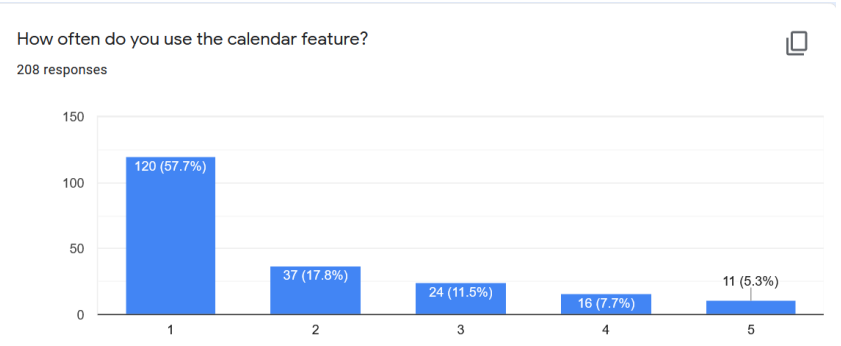
After a few months of the app running, I decided to conduct a survey on the users of NC Daily to determine the extent of success the app has achieved. While there may be many students and teachers using the app (just more than 60% of our school), whether the app benefits them is another question. The survey had 212 response in total

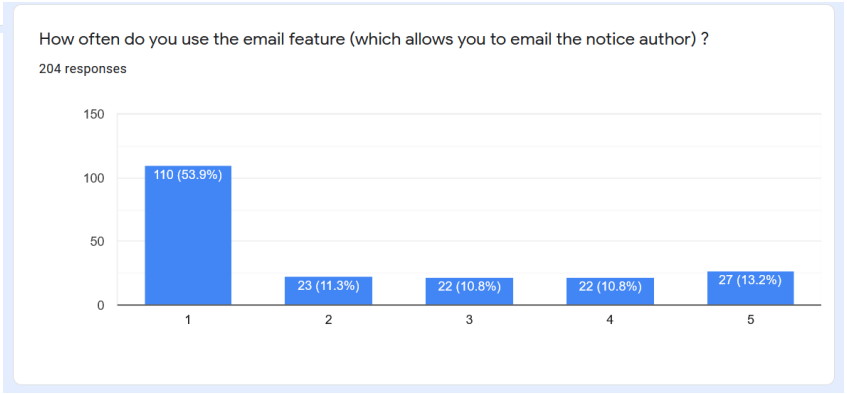


The app received an average rating of 4.5 out of 5, indicating user satisfaction with the product. 93% of users, indicated that they were reading the notices more often after using this app. This is a major success, and means the main end goal of the app has been reached.

The client also responded to the results, saying it would have a real impact on student participation and the way messages are read around school. The results may not be immediate, but with such a large increase in people viewing notices, it would be bound to increase participation rates eventually.

I have also included questions about how often users engage with certain features. More notably, I wanted to see if people actually used the mail and calendar feature. I hypothesised that most people wouldn't really use the email and mail feature, but a select few. This appeared to be the case with the results too.





Despite this response, however, I do not see it as a negative. In fact, I would argue this is an expected response. Not all students have questions for their teachers, or use Google Calendar for their event organisation.

The features will be kept in the app as they are there for the students who use them. As it appears to to be working for them

Extra written feedback was also given by some students. Some students wanted the notices to have a year level structure, and some people wanted the notices at different times sent. These are definite ideas for future improvement

Reflection of the apps future

The future of NC Daily looks promising. I have passed on the credentials of the Heroku cloud server and the admin panel onto the ICT director and he will now be in charge of the app.

The app will be maintained by programming club members, and I have walked them through the code and sections they could work on. The website will be promoted to all new students who go through the school, and issues/improvements will be worked on by the programming club members.

In terms of the technology used, the app appears to be fully scalable into the future. Utilising a modern technology stack (Python, CSS, HTML, SQL), and cloud based hosting, the only issue I can foresee is the time of the sending taking too long. However, this won't be a significant problem until the number of subscribers moves past 1000.

Final notes

While I am very happy with what I have created, and the impacts it will have on future students, I know that the app has significant areas where improvements could be made. These include additional features, optimisations and/or changes to the design. Unfortunately, I will be leaving this project behind with the programming club members, but I am sure they will continue the legacy of NC Daily and continue to improve it.

The app is also scalable to any Kamar parent portal website, and as Kamar is a very prominent school administration software, there is very real potential for a partnership with Kamar or turning NC Daily into a product that can be used for other schools too. All that needs to occur to scrape and turn the notices of another's school's notices is the link of scraping needs to be changed. Unfortunately, I did not have the time to do this, but this is definitely something that can be done.

It will be interesting to see how the app continues to impact the school, and how it will evolve. I am confident it will have a long lasting positive impact. Even within the few months I have seen it active, various teachers have mentioned its effect in students. Students are now regularly using the newsletter over the old system in their class, and teachers are beginning to enjoy its usage too. While I am yet to see any noticeable changes in participation, I believe this will come with time

Acknowledgement

I'd like to say a few thank yous to those involved in the project. These people have helped a lot, and I believe the project would not be where it is today.

- Craig Milmine, for his patience and input into the project.
- Margaret McLay, my digital tech teacher, for encouraging me to do scholarship and allowing me to pursue my ideas with freedom
- Newlands College Programming Club, for looking after the app's future and also providing invaluable feedback
- Vishek Kumar, Alec De Leon, Jethro Read, key students I had worked alongside with in the project to give invaluable feedback on design and implementation

Links to resources

I have included the link to the final iteration of the website here, at notices.newlands.school.nz

I have also included the link to the GitHub repository here and can be easily downloaded from here:
https://github.com/brandon-does-code/ncdaily-opensource.

In the event the link doesn't work, I have also included the entire repository as a zip file in the submission.