lhoraire Time Management Tool

Context:

As any other student doing NCEA, it didn't take me long to understand that time management is key when it comes to achieving high grades in the exams. Each student is required to have their own order of work to finish their tasks before the deadline.

This could be the more common gradual increase in time spend per day, or the ambitious-working hard in the beginning and acing the way to the exams.

Even though it has evolved over time, I could devise a method that would keep me concentrated to my work and that was simply doing as much practice past papers as possible. The number of past papers I did eventually become my mode of comparison between exam preparation and gave me the impression of how well I was about to do in the upcoming exam.

Even considering studying for exams as plainly doing past papers, which rarely is the case, it is pretty evident that there needs to be substantial planning in advance so that the desired number of past papers could be reached as well as time allotment for different deadlines that were closing by.

As surprising as it can get, even though several attempts have been made to take off the stress and pressure off the students, exams and deadlines still come in a bunch when they do come. This leaves no room for procrastination, or even wasting time planning how to spend time during those crucial hours before the exams.

Thus, this very obvious need for a proper time management system led me in the past few years to try a couple of tools from Excel spreadsheets to IFTTT mobile automation to help me keep track of my workload and my to-dos.

Moving away from students, I also did notice the extreme levels of tension that exist in the workforce. According to the health navigator NZ[i], the stress levels at work are generally increasing, thanks to reasons such as heavy workload and long work hours. Time management would definitely prove helpful in places like this. In fact, looking at some statistics by trafft[ii], if each employee spends around 10 - 12 minutes per day planning, they would be able to save up to 2 hours of time that would have otherwise been wasted. Boss employee relationship which primarily depends on how well and quick tasks were done, would also improve with this.

Thus clearly, time management plays a vital role in both studies and work.

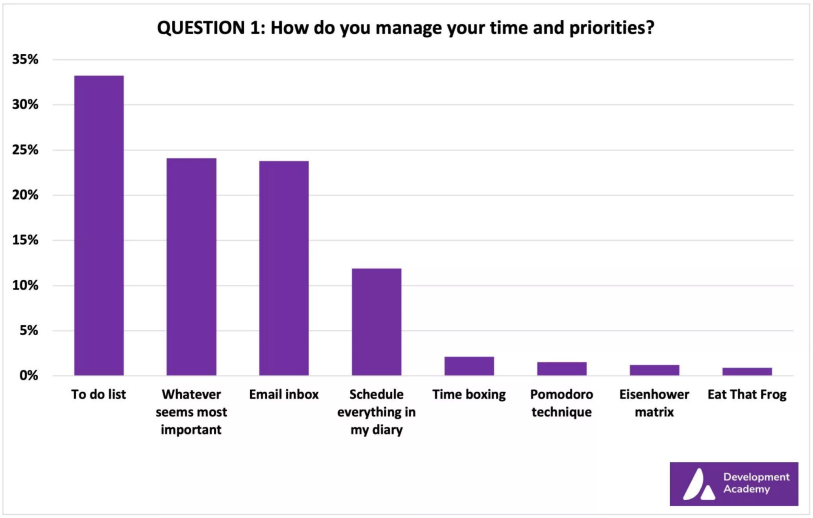
To summarize:

1. Students find it really difficult to manage their time for their exams, and this causes distress and tension.
2. Being required to finish before a given deadline, for tasks and projects can put a person in extreme pressure in their workplaces.
3. Time Management is the only feasible solution to these problems.

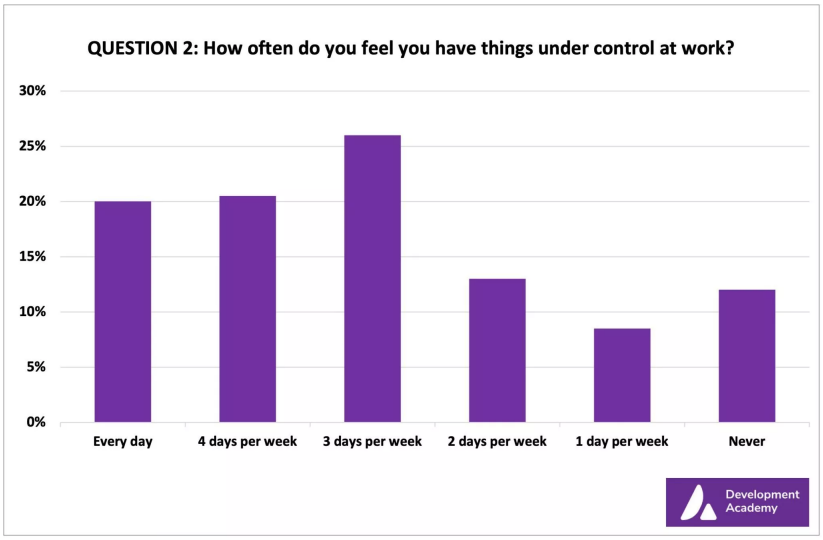
Time Management is a Problem

Time management is a challenge for almost all students and employees around the globe. In fact, according to the Development Academy[iii], 10%-15% of employees never feel things are under control.

It is also seen that only 18% of people have a proper time management system. This is less than 1 in 5 people. The 82% just use either a list, their email inbox or nothing at all.



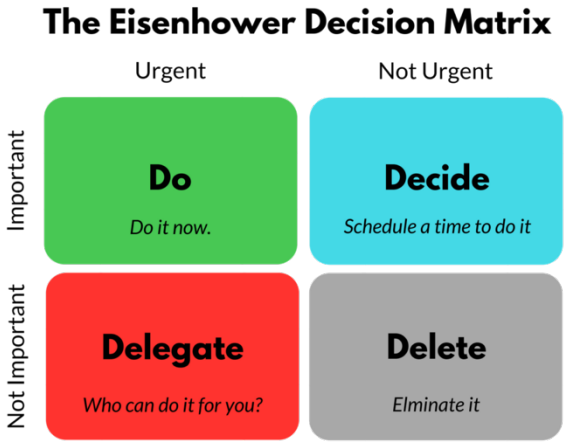
It can also be seen that over 20% of people feel their work is either never under control or only under control for 1 day per week. Moreover, 1 in 8 people feel that their work is never under control



Through this analysis by Development Academy I could realise that time management is a big issue that many face and something that many are searching a solution for.

According to the same report the Eisenhower Matrix seems to be the most successful strategy with 50% of the people feeling their work is under control every day and 50% feeling their work is under control 4 days of the week.

Eisenhower Matrix:



The Eisenhower strategy for separating and sorting tasks involve three boxes of tasks. The first is the Do: urgent tasks that have to be done immediately, Decide, non-urgent tasks although important which has to be done at a later time. A time to do it has to be scheduled. If the tasks are not important: Delegate; make someone else do it for you, and if it is also not urgent, Delete; eliminate it.

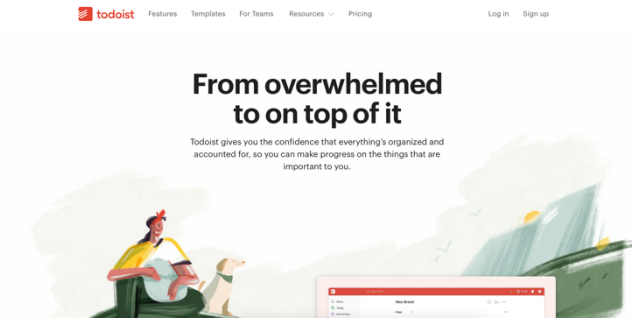
Even though this system has been a success and is being used by several people, it is up to the user to decide which task goes to which box. The Decide box requires the user to schedule the task to a different time, however, it does not present with how to schedule it. The Do box are usually the immediate tasks that are usually small such as the homework for the day. The Decide box however deals with longer tasks such as research and planning which definitely requires proper time management to be successful.

The Decide matrix is of interest to us, as this is something that a person has to do by themselves. This could also be the box that is the hardest for the user among the four. For users who might be finding it difficult to decide when to do what task, it would be great to automate this for them.

Before I start to work on any digital technologies outcome that might help with this I would need to inspect other similar tools that exist.

Time Management Tools that already exist:

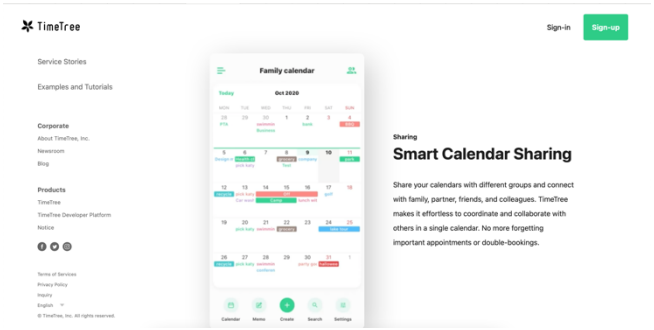
Todoist



Todoist is a time management app that allows its users to create tasks, subtasks and assign them to their team and/or make sure that tasks are done. In its essence, it shows a very user friendly list of tasks that have to be done by the user. It has boards and Kanban cards which can be used to organize and strategize what all tasks have to be done.

However, this does not come with the option to automatically assign hours to each day according to the work the user has to do. In short, here the user has to devise by themselves the tasks that have to be done on a day, and the app will act as a reminder and a sorter.

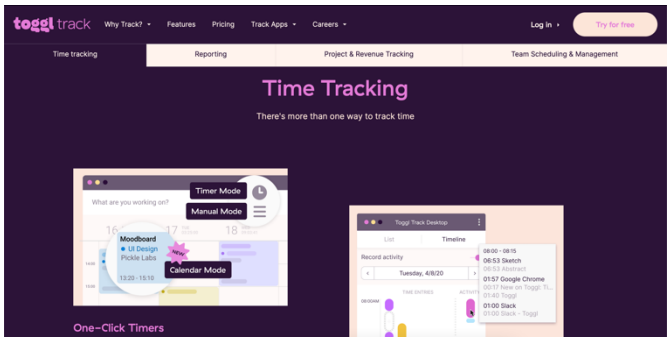
TimeTree



TimeTree is basically a calendar app that allows multiple account sharing so that the events can be controlled and seen by different users, usually in a team. It can be used for time management as it shows which all tasks have to completed by an employee for example within a due date set by the employer.

However, the tasks in this are not automatically created and require manual effort. Even though the calendar shows how much work has to be done by the user, the tasks placed in the calendar also need to be made by the user itself or any other person. There is also price tiers for this app and it might not be suitable for every use case.

TogglTrack



This tool is similar to TimeTree as it does involve in the creation and management of tasks using calendars. However, here it also creates reports for users on how they are managing their time so that they will be able to improve on their time keeping. It is more of a team / project management tool that is mainly aimed at businesses, even though normal users can use them. It has a free and paid tiers and some features can be limited to paid users.

Just like the above examples, even in this one tasks for each day has to be assigned by the user itself before or during that day. Even though there are small scall machine learning, nothing really is done to help user assign the tasks according to their needs.

Stakeholders

A project aiming to help manage the time will be useful for both young people and old people alike. It can be used by almost everyone irrespective of their work field and can be especially useful for students to spread out their work.

For students it can be used to help them prepare for exams and for important due dates. They can get prepared in advanced and not need to panic towards the end. This will also be useful in stress management in students.

Employers and employees can also find the feature of placing the exact amount of work that has to be done for them on particular days as useful. They would not need to spend much time to decide how much work has to be done on which all days.

Talk with Year Level Dean

I had a chat with the Year 13 Dean to understand their experience in understanding how time management has helped students cope with stress and anxiety.

From this discussion I could make out some points:

1. A lot of anxiety comes from feeling unprepared, and also not knowing how best to prepare for an exam.

2. This can really affect how well students are able to succeed as if they are anxious before an exam they may not get a good sleep the day before and they may not know whether they are properly prepared.

3. A key part of preparing for exams is planning how you will use your time in the weeks and days leading up.

4. If it is known what has to be studied, then dividing them across time that's available will allow to know if you are on track to cover everything required, thus alleviating stress.

The essence of the discussion boils down to the fact that time management indeed has a really strong correlation to having student stress levels decreased. Also how well in advance time is allocated for different tasks would allow for the feeling that you are ahead of all the things and will allow for self-confidence.

Another important requirement is to divide the tasks across times so that small chunks of work is done each time and to make sure that everything is covered.

Specifications

From talks, research, and discussion that I had with teachers, peers and other stakeholders, I managed to summaries what are the requirements for this time management tool that will potentially help students and employees.

1. Produce a schedule for the user. This should include all the tasks that they wish to complete by a particular day. It should also be ordered in a logical manner.

2. There should be ways in which users will be able to modify their schedule without much changes to the parts that they are not changing.

3. The schedule that is produced should be usable by people according to their needs.

4. Easily usable and guessable user interface.

5. Implications (accessibility, privacy, etc) should be addressed throughout the program.

Planning

Mathematical Models:

One of my inspiration for this project were the vast possibilities of solutions to everyday problems that mathematical modelling allowed. Specifically, it was the curves that interested me the most. I had experience from last year, when I used a relatively simple quadratic model to animate a fade-in fade-out of items as the user scrolled past them in a website.

In that project, I used this equation:

$$y = \frac{x(x - H)}{-1 * (\frac{H}{2})^2}$$

Here, y is the opacity value and x is the scroll height the element has covered across H, the total height of the screen.

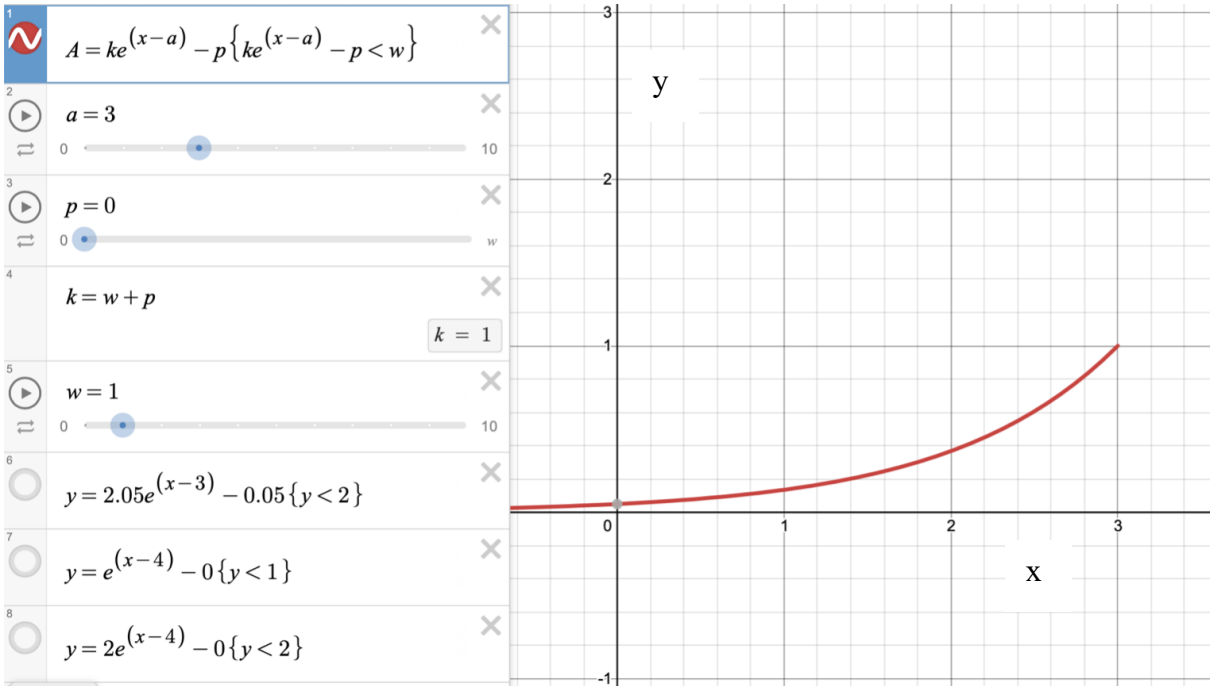
This allowed me to perform my desired animation without the use of any clumsy jQuery .animate() methods and that too being hyper light weight respecting end user browsers and resource usage.

It was after this supposedly success that I indulged myself into manipulating mathematical models for circumventing my computer science problems.

Exponential Equation:

When I started thinking about an app that would divide workload across days, the first type of function that came into my mind was an exponential function. I reasoned that the exponential curve which accompanies many humanly traits could as well be used for the progress that one would make to finish a task.

The logic worked as follows: The segment between the y-axis and $x=n$ would be used to measure the progress; where n is the number of days remaining till the due date. At $x=n$, the exponential growth should max out at 1. This would mean that 100% of the work has been done by this date.



This model, however, came with several shortcomings that skimmed its possibilities to be used as an efficient model.

1. The study patterns of many people are not exponential. In fact, there are people who would have different study patterns according to the topic they are studying.

2. The model produces percentages of the work that has to be done by each progressive day. This would not thus be an efficient way to know how many hours of work has to be done on each day. Producing a schedule from this would get really complicated.

3. The graphs do not have a 0 y-value even though the values are really small. It would be really great if I get a function that has the initial y-values start from 0 as it would show that no work has been done.

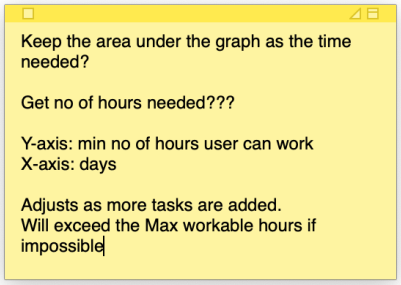
4. Multiple tasks would each require their own exponential graph. This would mean that the situation can get complicated real quickly.

5. Using exponentials in the function could also cause delays in calculations in the code and delays are never welcomed as these calculations are needed to be done as fast as possible.

These observations made me reconsider the model that I had begun to work on. I started looking for more easier and basic curves that could replicate work load of a student.

Quadratic Equation:

Exponential equations even though seemed perfect, were not easy to play with. I needed a model that would be easy to use with generic or basic mathematical operations (in contrast to *e* or *ln*) inside loops, without causing any delays in calculation. I would also want the workflow to look humanly understandable as possible.



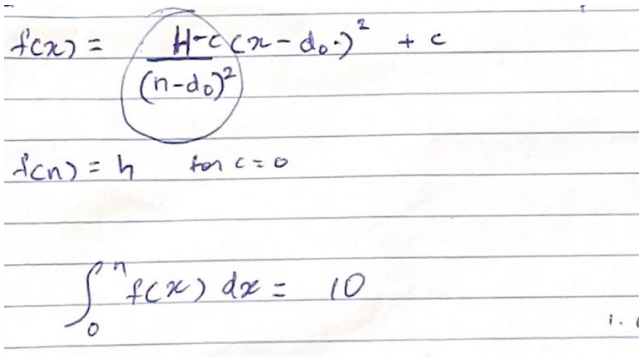
A second possibility was a quadratic equation. However, I was not quite sure how I would be using a quadratic equation to model workflow; would I just use it like the exponential showing the percent of work done by each day or should I be calculation the area under the graph for each day?

With my prior experience, it seemed as if the premiere was not intuitive. It also looked hard to be coded. I would obviously need a proper understanding of the model before jumping into code as this could become a big risk on my time if I were to change decisions later.

Finding the area under the graph looked intuitive enough for me to code. I would just need to model a function that is increasing and the total area under the graph before the due date needs to be the total area that the user is giving. The assumptions that I am making here would be that a person's workflow would increase by time.

This led me to this equation.

$$f(x) = \frac{H - c}{n^2}(x - d_0)^2 + c$$



It would plot a function that would have reached the y-value of H (maximum work hour limit) by the due date. This will make sure that the function would have a domain less than the user's limit. n is the minimum number of days the task would take and d_0 is the initial day of the task, basically the x-translation constant.

c, even though looks like an arbitrary constant, is not in our special case. It is a 'flexibility constant'. It will take care of how much work needs to be done towards the beginning by translating that end up or down.

The integral of this function should give us the total hours the user would want to work on the task.

Because we are making the y-value to equal the maximum user workable hour in the final day and the integral does not depend on d_0, we have a very important relationship between n, H, c and the total hours (G) when taking the integral. Because H and G are constants in this equation, we essentially get to control c and n.

For the ease of calculating definite integral with initial value of 0, the model is translated horizontally such that d_0 is the y-axis.

Definite integral of $f(x)$ will give G,

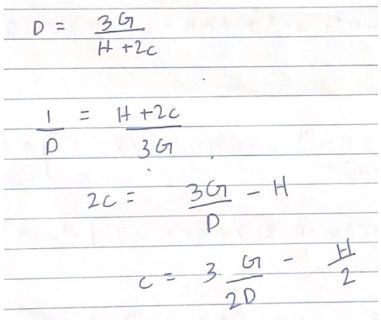
$$G = \int_0^n \frac{H - c}{n^2}(x - d_0)^2 + c$$

Since $d_0 = 0$,

$$G = \int_0^n \frac{H - c}{n^2}x^2 + c$$

$$G = \left[\frac{H - c}{3n^2}x^3 + xc\right]_0^n$$

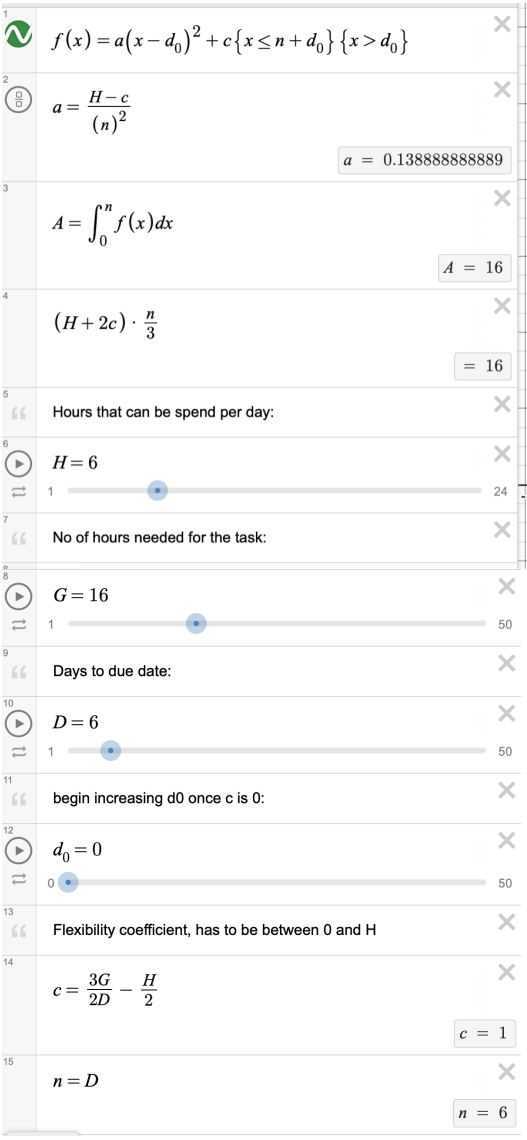
$$G = \frac{H - c}{3}n + nc$$

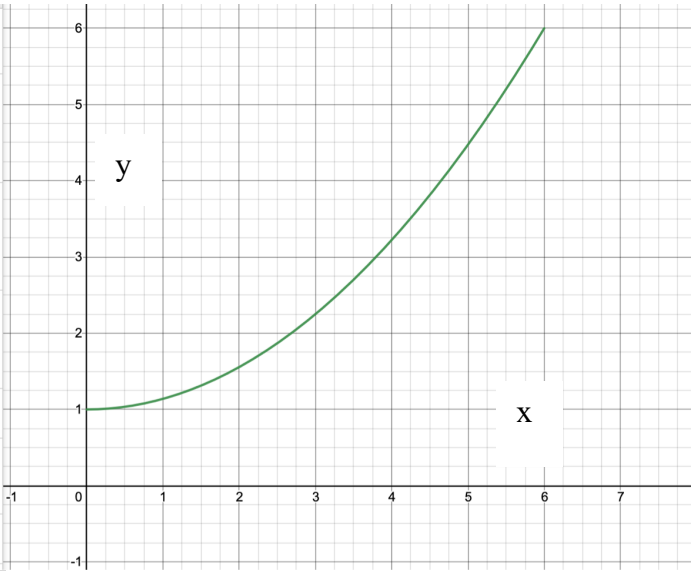


Solving for c,

$$c = \frac{3G}{2n} - \frac{H}{2}$$

This relation can be used to decide what c or n must be according to conditions by controlling either one of them.





Thanks to the relation, c is a constant when the task is really close to its due date. The closeness and the number of days required would of course require some programming and estimation.

In this Desmos graph, we can adjust the values to see how the function would change with the variables. In this situation, the graph shows a task that is 16 hours long. The user can work only 6 hours per day and hence the function approaches 6 hours by the 6th day, which happens to be the days left for due. It can be noticed that the initial hours needed is 1 and that the graph is limited by the y-axis (it cannot cross behind it). This is controlled by the *c* constant.

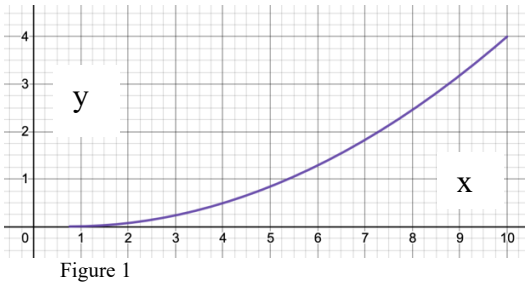


Figure 1

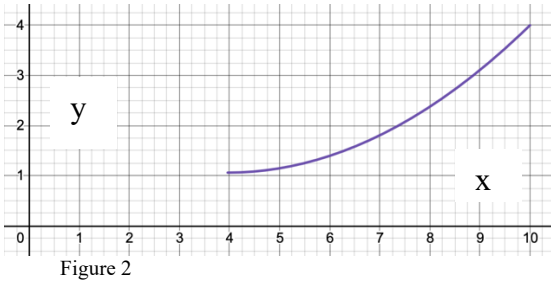


Figure 2

Figure 1 shows the graph when c is 0. In this the work has been spread out to the maximum in an increasing workload scenario. However, this might not be desired. Here days 1, 2, and 3 do not even have enough work. To the right is another possible model with just a different *c* value. With careful inspection it can be seen that this model is actually better even towards the end. The amount of work that needs to be done in the last day in the second model is in fact lesser than that for the first one. This is because the work that has been spread out in Figure 1 is from the middle days which in Figure 2 can be seen as a bit more concentrated.

Estimation of *n*

Being the minimum number of days that the tasks will be taking, there would need to be an estimate for *n*. For tasks that are set really close to their due dates, *n* would equal to d_0, the number of days to the due date and hence there is no need to estimate.

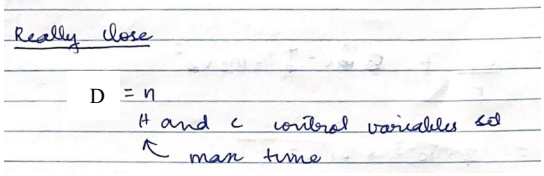
However, for tasks that are into the future, a *fair* estimation is required. It has to be noted that in the everyday use with multiple tasks overlapping others, this constant would only be needed for the model to be established. In the process of making the schedule tasks are most probably going to be stretched or squeezed according to the situation. To estimate how much time would be needed for a task that is done roughly the same each day,

$$n = G^{\frac{2}{3}}$$

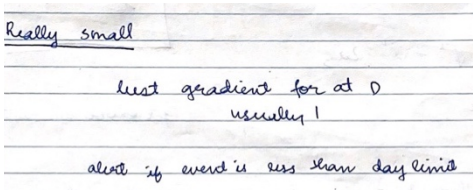
Results on trialling, work of 10 hours: 4.64 days, 20 hours: 7.36 days, 5 hours: 2.92 days.

Estimation of *c*

There are three possibilities when it comes to task models.



One would be when the task is really close to the due date. This would mean there is but one possible value for *c*, and this if found by using the relationship that is found above.



For tasks that are really small, which is defined for our convenience to be less than 10 hours, it has to be made sure that the curve is not too sharp towards the end, and that it is almost a gradual increase in the gradient. This will mean there would need to be a change in how we define the variables in this situation.

For this case, *H* is not going to be the daily limit of the user, but instead only the maximum hours that can be spent. This has to be decreased and *c* adjusted accordingly to make sure that the curve is smooth enough. To make it smooth, I have decided to have the final gradient when the function approaches *D* to be about 1.

This would mean,

$$f'(x) = \frac{H - c}{n^2} \cdot 2(x - d_0) = 1$$

When $x = D$, $D - d_0 = n$,

$$f'(x) = \frac{H - c}{n} = \frac{1}{2}$$

$$H = \frac{n}{2} + c = \frac{G^{\frac{2}{3}}}{2} + c$$

Also,

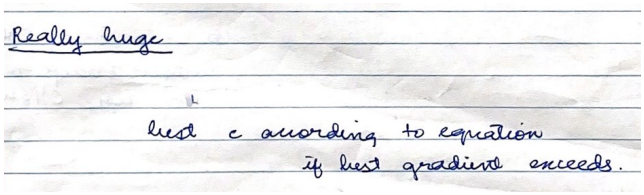
$$3G = (H + 2c)n = (H + 2c) \cdot G^{\frac{2}{3}}$$

$$3G^{\frac{1}{3}} = H + 2c$$

Substituting 1 in 2 and solving for c,

$$c = \frac{1}{3}\left(3G^{\frac{1}{3}} - \frac{G^{\frac{2}{3}}}{2}\right)$$

H can be found by substituting in *c*.



For large tasks, usually needing greater than 10 hours, *c* has to be such that the beginning is not undesirably small, and the end is not too heavy. One way to achieve this would be to find a value for *c* that would minimize the amount of time spent on the last day, i.e., optimize *c* for the minimum value for *f(D-1)*.

$$f(D-1) = \frac{H-c}{n^2}(D-1-d_0)^2 + c$$

With $d_0 = D - n$

$$f(D-1) = \frac{H-c}{n^2}(-1+n)^2 + c$$

To minimise *f(D−1)* we can find the derivative to get the *c* which will allow this[iv],

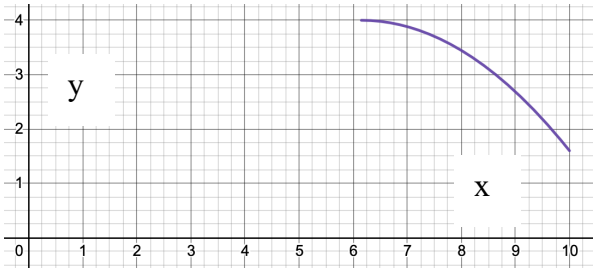
$$\frac{d}{dc}\left(\frac{H-c}{n^2}(-1+n)^2 + c\right)$$

$$f'(D-1) = \frac{-8c^3 + 16Gc^2 - 4Hc^2 - 6G^2c + 2H^2c + 4GHc + H^3 - 2GH^2 - 3G^2H}{3G^2(2c+H)} + 1$$

$$= 0$$

This has to be equated to 0 to get the values for c. The lower solution to c will minimize for *f'(D −n)* giving the model that will produce the least amount of work in the last day.

Other 'Gradients'



Upon talking with one of my peers and a stakeholder, it was also decided to include a decreasing and flat work distribution. This will allow more flexibility for the users. For example, the decreasing distribution can be used by those who has ethos of heavy work

towards the beginning, or the flat distribution can be used by those who would want to spend almost equal amount of time writing a report every day.

It can be easily noted that the decreasing gradient change slower at the beginning and gradually increase. This contrasts with mirroring the increasing distribution. That would have made the workload decrease more rapidly in the beginning itself which might not be desirable. With this model proper emphasis is being placed for the beginning of the task.

It was fairly simple to achieve these models. In the decreasing graph, the c and H, were exchanged, and in the flat graph, c and H, were made equal.

Feedback: The mathematical models when discussed with others were suggested to be too complex than it required to be. I had to reflect and see whether all of the calculations that I did was indeed required for running the algorithm that I aim to do.

The function that models the task cannot be changed as it has been modelled to make sure it follows all the requirements of the user. Most of the complexity comes from the assumptions that are being made. The steps to reach these solutions can be complicated, but these steps do not need to be made by the script. During run time the code would substitute the variables in the function to form results that can be used later in the code. The only calculation that would need to be done by the code would be finding roots and this would not be using much resources most of the time.

Thus there would not be much complexity in the code, even though, the route to reach the equations used in the code does seem to be complex.

Setting up the Workspace

My workspace would include the necessary IDE, programming languages, designing and time management tools.

IDE for coding:

I chose to use Visual Studio Code for programming and using it as my code editor as I have previous experience working with it. It has an integrated terminal which would be handy when it comes to running python and web servers if I intend to do so later. It also has extensions which will help me in my coding process.

Apart from the debugging options that it has, it's colour coding and error detection would also help me in fixing errors quickly.

Extensions that I used: Python Pylance for IntelliSense, GitLense and Prettier. The availability of extensions is also another reason for me using this application.

Project Management:

There would need to be a way in which I would be able to keep track of how the project is going and also backup the code onto a remote repository if say there was an unforeseen accident. For this I would be using git and GitHub. This will also help me share the code with others who could give back feedbacks easily. I would also not need to worry about storage and also this might help me with hosting in the future.

Kanban task management in Trello will be used to make plans and also to make sure that the project is going as planned. This will allow me to use time effectively and not miss any required feature.

Programming Languages:

To program my project, I will be using Python, mainly for its simplicity to use and also as I have a substantial experience using it. Python supports several external modules and has several web frameworks build on it that I can choose from. This reduces the number of languages that I would need to know in this project.

For the website I would have to choose between Flask and Django. Flask is simple to use and is lightweight. However, Django is used more commonly in the industry. I would have to research more into this subject when I reach the stage of web hosting.

Anyhow, I would need to know basic HTML5, CSS3 and JavaScript. This will help me design the website and its immediate front-end functionality. I would also have to use my experience using jQuery to fasten the coding and to make it more efficient. It would also make my code look clean and with lesser lines. I would also assume that I will need Bootstrap knowledge so that I would not need to design each and every component.

Browsers to debug:

I used Safari and Chrome for debugging. However into the testing sprint I have made the website run in different operating systems and different browsers with different screen sizes.

Designing:

To help with designing the planning wireframes, icons, models, and also vector art for the program I would be using draw.io and Adobe Illustrator interchangeably. Since I know the basics on how to use these, I would not find it hard.

Iterative Development:

The development of this program from scratch till production will be done in several stages. These will be called sprints. After each sprint, I will be reconsidering whether the changes that I have made in the program were worth it and what other changes that I will be making in the future.

Sprint 1:

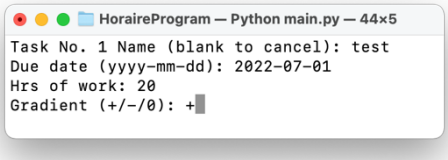
With the basic requirement of modelling a function done, I would now have to implement this mathematical model into a Python code. I will need to find the area under the graph for fixed ranges, i.e., get the number of hours of work that need to be done on each day. One tricky part in this would be to find an efficient and easy-to-use mathematical module for my

project. I would not want it to be taking really long but I would also want its code to be simple so that it is maintainable and clean.

After researching for a while, I came up with these two modules: *sympy*[v] and *scipy*[vi]. Both were similar and would do the task of basic integration and equation solving.

Sympy would make the code look neater as it was a library for symbolic mathematics. I would be able to write the code just like how I would write a mathematical equation, and this will make it more maintainable as the variables used would be mathematically obvious. However, as my initial tests showed, this came with the price of a slower runtime. On doing further research I concluded that as my algorithm would need to find the area under the graph multiple times – meaning this module would be most possibly used inside loops, I would need one that is used to analyse big sets of data. This made me choose *scipy*, which happens to work perfectly as far as I was concerned.

Starting to work on the backend, I made a dummy function to take in my inputs. This will help me in making sure that the model produced is the way I want it to work as it is worked on.



Because my project heavily depends on time, I would need a method to make time relative. This will make my mathematical model which has a range of time simpler. If I were to use normal dates all throughout the calculations, I would heavily need to depend on datetime module which might cause delays in the runtime. So I would need to somehow convert date into a form of number that I can use in mathematical operations in the model and the algorithm in the future.

One potential way to do this could be to use the UNIX timestamps which is a running count of seconds since January 1st 1970 in UTC. This would represent time as a number between 0 and 2147483647. However, the precision that is found in this timestamp is not at all necessary for our calculations. We only need to form a way to keep track of the number of days. In fact to convert the current time stamp to days, I would need to perform calculations every time and thus destroys the purpose.

By using almost the same idea but for days, I could find a way to do this. I achieved this by calling the date 2000-01-01 as *x=0*. With this being the origin, every x value would be one day after that day. This will make the date 2021-10-06 to be represented by 7948, meaning 7948 days after 2000-01-01. With this in place, I will be able to perform adding and subtracting days as well as assigning days to tasks in the future with just integers, and then converting them back to datetime form using the *datetime* module at the end. The user input would thus need to be converted into this "*date delta*" form.

In the dummy start function, the user's input would be in the form of string which would be converted to *datetime* before getting converted to *date delta*. This would mean there is a strict input format of *YYYY-MM-DD*. To facilitate with converting to and from *date delta*, there would be the need for helper functions. I will thus create two functions *getDateDelta()* and *getDateFromDelta()* in a *helpers.py* which will allow for this.

Since each task that is created by the user would require a model, it would be best to make task models to be class objects. When the user inputs task details, the dummy start would create a *TaskModel* object with the details. This object will be defined in a separate *model.py*. It will take in an id, the due date, hours needed, hours that can be spend per day, and gradient

of work arguments. Immediately when this object is assigned, it will create a model function with the inputs. It would also have a method to produce a list of days with the amount of work that needs to be done on each of these days. For this I would have to use the *scipy.integrate.quad()* to find the area under the graph for each day interval.

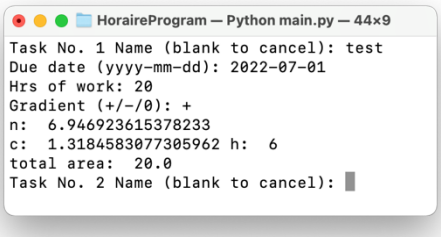
It was decided to keep the amount of hours that can be spend as the daily limit on a weekday because these will be the most common days in the week. Later limits for weekends and also maximum limits for weekdays and weekends will be introduced.

For the model, small tasks are those that take less than 10 days, for these tasks approximation of n needs to be done. With this c and H are calculated.

For larger tasks c is approximated using the derivative formula. This is done using the *scipy.optimize.root_scalar()* function to solve the equation mentioned above using the user inputs. In this case H remains to be the maximum workable hours in a weekday.

It will be over here where the start day d_0 is calculated. If it happens to be before today's *day delta*, then c and n would need to be altered, so that the task can be completed in time. In this case n becomes a constant and H becomes the daily hours that can be worked, and c is calculated. d_0 is made today.

If the chosen gradient is negative, then c and H and swapped. However, if it is flat, then a new c, H, d_0 are produced with the estimated n.



For testing purposes, the chosen c, n and H is shown.

List generation would work by integrating the model in each day intervals between d_0 and D. It has to be noted that d_0 needn't be a whole number. In this case the first integration would have to be made separate with the range of d_0 and ceil of d_0 assigned to the day floor of d_0. After this is done, definite integrals of our model should be done with intervals of days between ceil of d_0 and D which is then assigned to the respective days. The result will be a list of sets, each with the day and the number of hours that has to be spend on that day. This can be later used to produce a schedule.

As each task is an object, they can be passed onto another module where they can be processed to fit the user limitations. I have named this module to be *repositon.py* with a class of *Reposition*. The cumulation of *Task Objects* will be sent into this class which will run algorithms on it and eventually produce a schedule. Currently I am aiming to just print the output to the user without storing or retrieving anything.

The task cumulation would be a dictionary with an arbitrary task id as key and the *TaskModel* as the value. For debugging purposes, I will be converting the id of the tasks to strings, with a *t* before the number. This will help me to properly understand the location of any error that might occur in the future after connecting with a database.

This will be done inside a method called *schedule_cumulation()*. Here a barebone schedule with different days and the tasks and the hours done on these days will be produced. I have decided to keep the schedule a dictionary. One concern over here would be that in python, dictionary objects could lose its order without using an *Ordered Dictionary*. However, since the release of Python 3.7 insertion order is preserved in *dict* objects and hence should not be

causing difficulties in the code. Moreover, the only reason this is of need is for a more comfortable reading experience, since this is not a necessity for the code I do not think I would really need to rely on the new feature that much.

Along with the task and the number of hours that needs to be spend on it during that day, I would also need to know the total sum of hours that needs to be spend on that day as well as its difference from the user's weekday, weekend limits. For this I created a function to check whether a day is a weekday and used the result to decide on the limits. This will be found useful in the future when I will have to find how much free time a day has so that there can be rescheduling.

Review:

In this rather short sprint, I was able to produce a foundation for the algorithm that I will be working in. This is in the form of a python script that will accept tasks and at this stage, produce a barebone schedule. I was able to implement the mathematical model for a task properly which includes features for short, long tasks and tasks with different gradient. I was also able to create a module for fixing these tasks to their respective days. These days will contain a cumulation of the tasks as advised by the respective models of the tasks. This however would not be usable for most of the scenarios as with multiple tasks, their sum would exceed the limit set by the user. In the next sprints I would need to find a way to properly distribute the tasks.

Sprint 2:

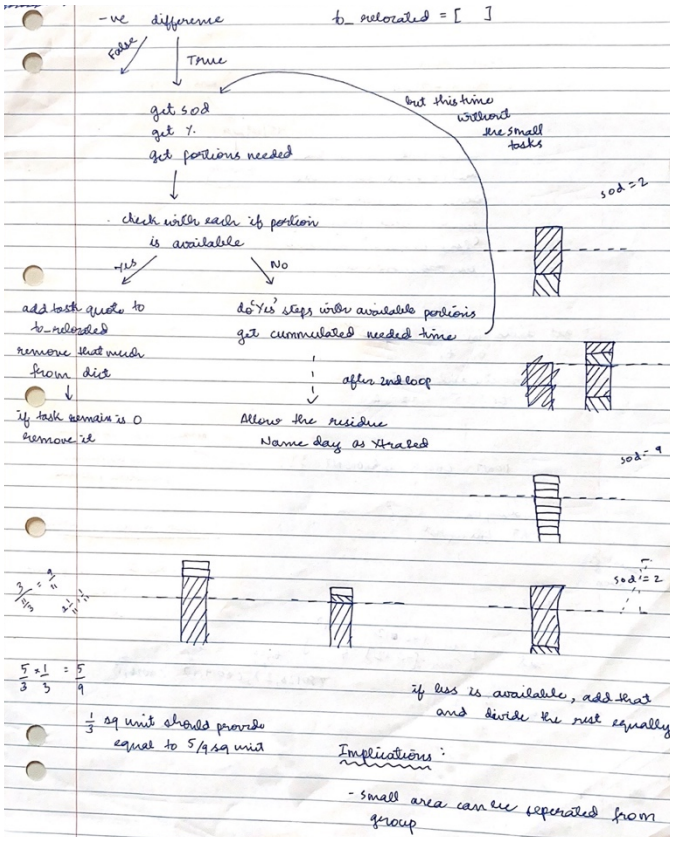
Up to this stage what we have done is implementing the mathematical model into a python code. There is the need to now produce results that users would like.

The barebone schedule that is produced can be further processed to produce ones that are limited according to the user's need. This will mean, first, taking away overflowing hours from days and later rescheduling these hours into other days according to some priority allocation.

To remove extra hours from a day, i.e., make it 'legal', I decided to use a system of ratio to remove task hours according to the closeness of the particular day to the due date. For this I would be needing the due dates of each tasks in the day, use its percent out of the sum of all due dates to get the proportion of the extra hours that need to be removed.

The excess amount of hours of tasks will be removed and then added to a dictionary that will contain the total excess of hours assigned to the tasks.

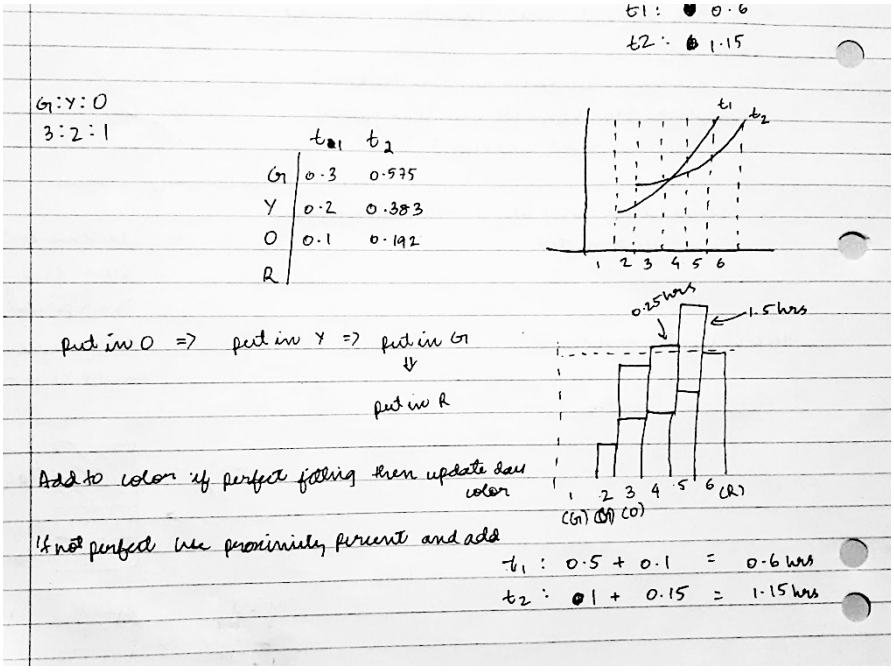
This will bring forth 2 different scenarios. One in which the proportion that has to be removed is available and when it is not available. If the proportion that has to be removed is available then we will be able to remove the correct amount. If the available amount is perfectly equal to the amount that has to be removed then its presence has to be removed from that day, which will mean we will have to pop the *task id* from the dictionary.



If the amount that has to be removed is not completely available, then the total amount that is available has to be removed. After this is done, a recalculation of ratio is done to get the amount of hours of the other tasks that has to be removed. This is then used until the difference becomes 0 or really close to it.

This will produce for us a schedule that fits the limitation of the user. However, it would not have all the hours that the user requires. These need to be added back.

One way to add these tasks back to the schedule, would be to categorize the days that are being used according to the space availability. This will allow for adding the task hours back into the most suitable days.



First I planned to categorize the days which had free spaces according to the available free spaces by Green (G), Yellow (Y), Orange (O) and Red (R). I would take the removed hours

and add them to these groups according to the gradient of the task; if it was an increasing task, I would add first in O, then in Y, G and finally in R days. The order would be G, Y, O for decreasing task as it is known which ends of the tasks have more free spaces.

However, this gave rise to several difficulties, that might affect the effectivity of the algorithm. To group the days, there needs to be a well defined method. It would be that G days are those in which there are 75% or more free space, or R where <1% free space. But, this might prevent the code to be dynamic and to behave according to the particular situation it is handling. There is also inconsistencies when it comes to group among the days inside a group. This would need more sorting methods, and there would be no specific benefit of having this grouping. Moreover, it was decided to use the ratio 3:2:1 of removed work to be put in G, Y, and O respectively in increasing tasks. This could lead to some borderline situations that might produce unwanted results.

Also the idea of having different gradient to have different sorting methods would give rise to big issues when it comes to multiple overlapped tasks as it is in normal usage.

This made me to rethink a different sorting method for the days into which tasks can be reinserted.

Another possible way would be to take every day that has at least one task and sorting them according to a method which would consider free space, and number of tasks. After ordering them, task hours can be reinserted to these days. These days will require to be within the range of beginning and final days of that particular days. For this we will need to create a dictionary to hold this data.

These days in a descending order of priority can be filled until the day limit. This is not the maximum day limit, this will happen later. Filling of the days will happen with a Proximity Percentage of the tasks as this will help with maintaining the priority according to the due dates. Tasks with closer due dates will be added in a larger ratio than tasks that have a further due dates. This process has been referred to as *day filling* in this report.

After all these days have been filled, if there is still some tasks that need to be scheduled, we should start filling days prior to the modelled start days. There does not need to be any special sorting but instead in the descending order of the days, as I have assumed that in this part of the task, it would be better to fill and finish off with the tasks.

For this, I will take previous 5 days of each task and get their union. This is then used by the same filling algorithm as the previous one, until it reaches ‘today’s’ *date delta* or the task gets satisfied.

If these were not enough, the next step would be to fill the days to the brim – the maximum daily limit. For this the union of the days of tasks that are still to be done are taken and are run through the day filling algorithm in descending order.

This would make use of all the space given by the user and should be the suitable schedule. Remaining tasks that were not been able to be added remain in the *to_reschedule* dictionary and can be used to show alert to the user.

The schedule has to be stored. In the final stages, this will be in a database, however, for testing purposes, I decided to store it in a JSON file.

Before saving there would be the need for some cleaning of the schedule. I would have to revert back the *date deltas* to readable dates. I would also be able to round off the decimals and also possibly make the numbers in readable hours form when displaying to users.

Here is the resulting JSON files: *schedule.json* and *tasks.json* however not completely cleaned.

schedule.json file has the schedule as a JSON dump of the *schedule* dictionary. I could remove the *data* key as this was only needed for calculations, but I decided to keep them to make recalculations when adding more tasks later easier. I will not be storing them in the final product, but will be calculating them.

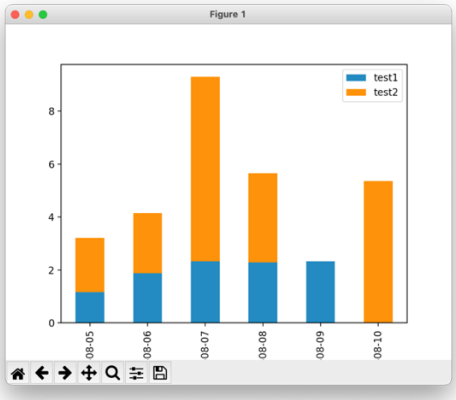
Review:

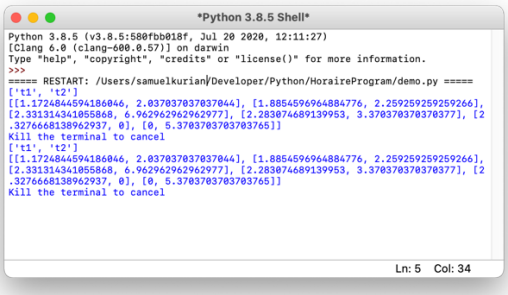
In this sprint, I was able to successfully produce schedules from tasks that are inputted. They are also saved to files so that they can be retrieved later. The algorithm is almost done and with constant testing it will be up to production quality. As of now I have been able to produce a day filling algorithm that can be used to take tasks off overflowing days and replace them into free days using priority ratios and ordering. In the next sprint I would be looking into visualising the results that are produced as well as making sure there are no errors in the behaviour of the code. Also I would need to work toward connecting the python script to a website to allow users from anywhere to access and use the algorithm.

Sprint 3:

Till now we have been able to produce a python script that will take user input through CLI and produce a JSON output. Now we would need to concentrate on developing this program to take in user details as well as to production.

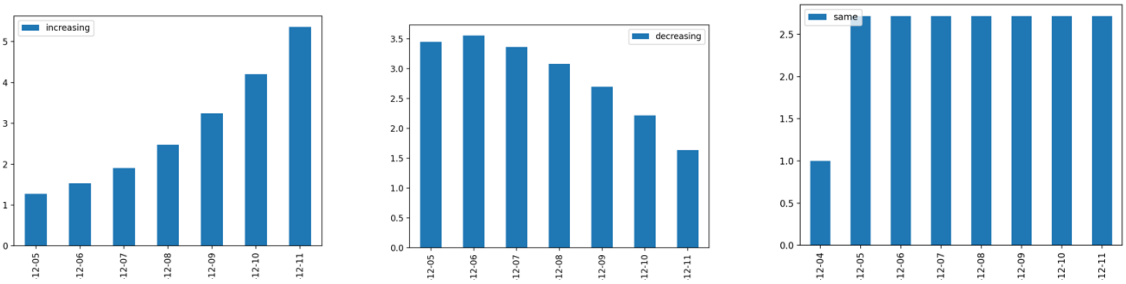
Also I would need to find a way to visualize the schedule as a bar graph as this will tell me how the final result would look. For this I will have to use *pandas* and *mathplotlib* to produce a simple collated bar graph that will represent the tasks that will be done on each of the days.





This will be the second version of this project. It contains a simple default *mathplotlib* GUI. This contains features such as zooming, panning and saving along with other accessibility options. I made the script of this *demo.py*, in such a way that closing the window would reopen the window after refreshing it. To close window, I would need to kill the terminal interface. This might go against the accessibility of the program but since this is just an adjustment for me to debug the schedule and is not meant for others to use, I do not think it will have accessibility implications.

Here is a visualisation of the three different gradients that the app allows the user to make.



Analysing these myself and showing them to peers, I could finally get to know if my mathematical functions were performing as they were built to. All the examples used here are 20 hour tasks.

From first glimpse, it is evident that the increasing graph works just as planned. The increasing curve follows a gradual increase in the gradient.

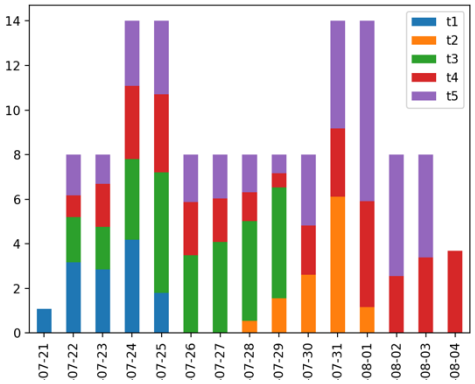
However when looking at the decreasing curve, it can be observed that the initial height is not the maximum. This when reflecting on how the math was made to work seemed pretty evident. The math behind making the decreasing curve was exchanging the heights of the ends. However, if the initial height of the task happens to be in the middle of a particular day (i.e., between two whole numbers) the area under the graph is taken from the min point of the graph and the next whole number. This could be less than the hours that has to be spend in the next day, i.e., the area under the graph between the next whole number and the number after that.

The same happens for the flat task.

To fix this error, I would need to find a new mathematical equation for the decreasing curve and make sure that the flat graph has a height in such a way that the ends are on whole numbers.

On discussing with stakeholders, I could reach the conclusion that the small detail in the beginning of the task curve isn't that important. Since all the other days followed the trend that was described there was no need to start over for the two graphs. Moreover, with real life situations, there will be multiple tasks overlapped and shifted around and this small detail would not be affecting user experience.

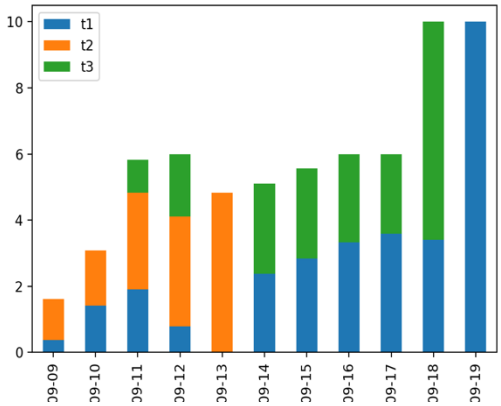
Results of the scheduling algorithm so far:



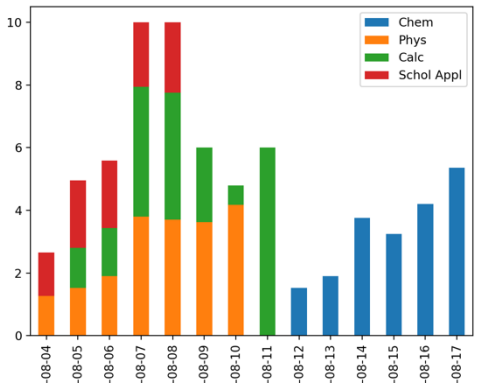
This example shows multiple tasks composed on top of each other. In this example, the weekday work is 6 hours, max weekday work is 8 hours, weekend work is 10 hours, max weekend work is 14 hours.

However, in this an error occurred in the code, and it resulted in placing some part of *t1* in the day of the testing. This is not desired as the code would not be knowing how much hours can be used on that particular day.

Also it was decided to make a feature to reduce the presence of other tasks during the last day of a task (the day before its due). This will allow the user to concentrate more on that particular task. For this I had to create a function that will see the due dates of the tasks and clear the last day from other tasks and add these task hours to the *to_reschedule* dictionary. This will happen before the day filling algorithm. This will mean there is still chance for these days to be filled with other tasks if the need arises.



This is a result after adding this feature and fixing the error. Here there are three increasing tasks, *t1*, *t2* and *t3*. They have dues at the 20-09, 14-09, and 19-09 respectively. They seem to fit like I wanted them to fit. It follows the same user preferences as the first one and thus do not overflow or show any undesirable day filling. Days 18-09 and 19-09 are weekends and have almost complete fill as they are the end of two increasing tasks. Weekdays 17-09 and 16-09 are weekdays with complete fills. Day 13-09 only has *t2* and shows that the new feature is working. The less amount of t1 in 09-09 is a concern and would be needed to be addressed in the future.



This is one of my own personal use for the algorithm, I used it for planning how much time to be spend for preparing for internals/tests and reports. In most of the days, the schedule seemed to be doable and in days like the 10-09, I feel it would be how I would want it to be. The red task was assigned a flat gradient. This happened for the most of it.

However, one shortcoming that I noticed was that there was way too much work to be done on 11-09. This task could have been spread a little more. This made me start thinking about some other ways to fill into days. However, it was perfect for now, and I will come back to this afterwards.

Our program still does not have the option to reopen tasks that were previously added to the schedule. Currently it just overwrites into the *schedule.json* file. For adding to perfectly work, there needs to be a 'filtering' system that will only produce task objects from old tasks as well with new tasks when more tasks need to be added. This will make sure that day filling algorithm would run with the appropriate tasks and not just the new tasks. After this is done these days will replace those already in the old schedule.

For this I created a function *filter()*. This will take in all the new *task objects*. It will also read the *tasks.json* and get all the old tasks. Using the start and end days of all the tasks, it will form groups of tasks which are based on the union of the days they are active in, i.e., groups in which tasks have overlapping working days. If any of the new tasks require at least one of the day that is taken by a group, all the tasks of that group is 'activated' and rerun in the algorithm.

This brings forth 2 concerns:

1. What about tasks that are not in any of the groups?
2. What will be the behaviour when 'previous' *day filling* happens in the algorithm?

Since there will not be any contact with tasks between groups, keeping non activated groups static would not be affecting the logic of the algorithm. These other tasks do not need to be activated as there are not changes happening close to them.

In certain cases, there will be the need for using previous days by the day filling algorithm in case of overflowing. For this, the old schedule will be reloaded as a JSON dump and will be used by the day filling algorithm after removing the groups that have been activated by *filter()*.

This will have certain disadvantages, during 'previous' *day filling,* such as the tasks set in schedule are not changed and the new tasks are basically added into the free spaces. If there are no free spaces close, because of due dates of other (static) tasks, the tasks that need to be added will be displaced further behind. To prevent the chances for this to occur, I decided to activate groups that are adjacent to each other by one day. This would mean that a group is defined by union of tasks that require the same days as well as adjacent days. In essence this would mean that groups that otherwise would be adjacent to each other would now be considered a single group. The big downside of this would be that more tasks would need to

be activated and it could result in slower runtime. However, this will make sure that tasks will not be displaced further back in most of the cases.

Another way that I had considered was replacing the 'previous' section of the *day filling* algorithm with making *tail* tasks, which were basically descendants of *TaskObjects*, but the amount that couldn't be added. These tails would try to follow the trend (gradient) of the main model and would have the due date as the start date of the main model. However the problem that I faced when devising such a plan were the cycles that were forming.

When tails are introduced they would activate any groups that were on their way. This would mean the tasks on the other group(s) would have to start over their whole process which might as well be involving tails. Also tails can also have subtails if they experience tight scheduling.

The amount of resources such a process would take makes it highly inefficient and thus I had to scrape that plan.

Working on the Website:

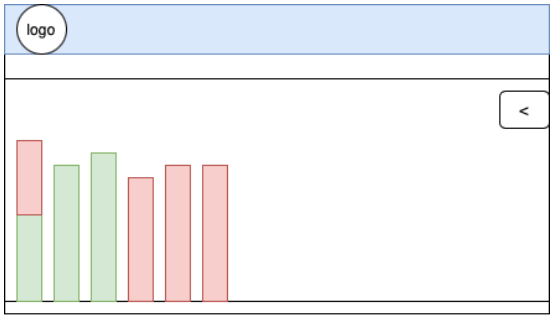
So far, we have been able to work on the working of the brain of the project, the algorithm. To allow users to use the algorithm in their everyday life I would need to produce a way in which they will be able to access it. This can be in the form of a app or a website. I decided to do a website as this will be easier to maintain and it would be accessible from any device that would have compatible web browsers.

For this I have decided to use a web framework that is written in python. After having a conversation with my friend I decided to use Django over flask. Flask even though is lightweight and simple to use does not have every option that we would require. Django also makes several instances of coding easier for me as there are features such as forms that are already available and just needs to be imported to be used.

The website would contain a login and signup pages where users will be able to register and use the service. This will take them to a dashboard where they will be able to add tasks after they have given the basic preferences such as daily limits and maximum limits. There will also be a page for editing tasks and also a page to edit user settings.

Design options for dashboard:

I used *draw.io* to produce some possible design for the main dashboard of the website. This will be the main screen that logged in users will see.



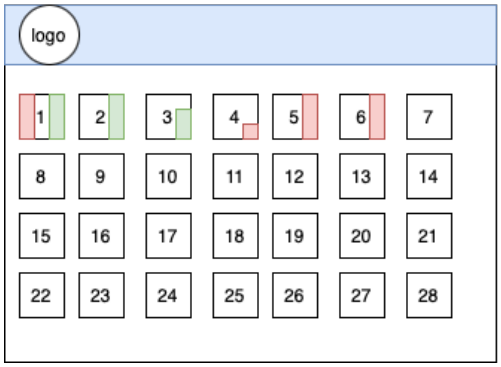
This is a pretty basic model. It has a bar graph that fills the entire screen. It is horizontally scrollable and shows the number of hours that has to be spent on each of these days.

More information can be seen by clicking on the top right button.

The bars for each task are colour coded. Also it can be hovered on them to see which task it is.

There will be the basic navigation panel in the top and the footer in the bottom.

Feedback: Looks far too simple. Taking the entire page with bar graphs will not look good in screen and will make potential users walk away. However, this might be a proper model for smaller screens where the bar graph would look better if they are bigger.



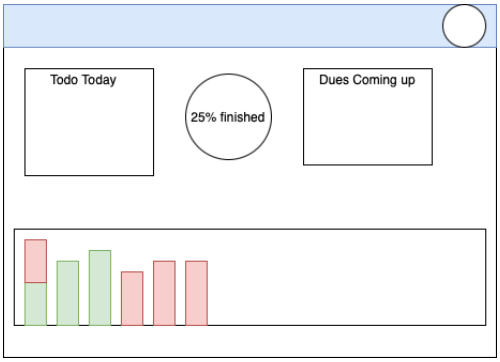
This model has a screen in which it shows a complete month calendar. It will basically be a replacement for calendars entirely and will show how many hours of each task has to be done on each day.

The tasks will be colour coded as the user decides.

Users will need to click on a button to view later months. They will be able to see enlarged and more detailed bars when clicking on the days.

This will open several possibilities for the website, such as integrating with calendar apps etc, in the future.

Feedback: This model can be really useful to the user as the UI shows a calendar and humans have evolved to make use of calendars to manage their time. Since there is already a calendar in the webpage, it would be fine to keep everything else simple. Because the display for each day is really small, there cannot be a single glimpse realisation of which all tasks have to be done on which all days easily.



This is an extension of the simple bar graph display. In here the bottom half of the screen is going to be a bar graph which shows the schedule of the user. This will be colour coded for each task. The users will be able to choose the colour, so as to prevent colour blindness factors. There is also a summary component to the top of the bar graph. This includes a to-do list for the day, progress of tasks that are being done and also a list of due dates that are approaching.

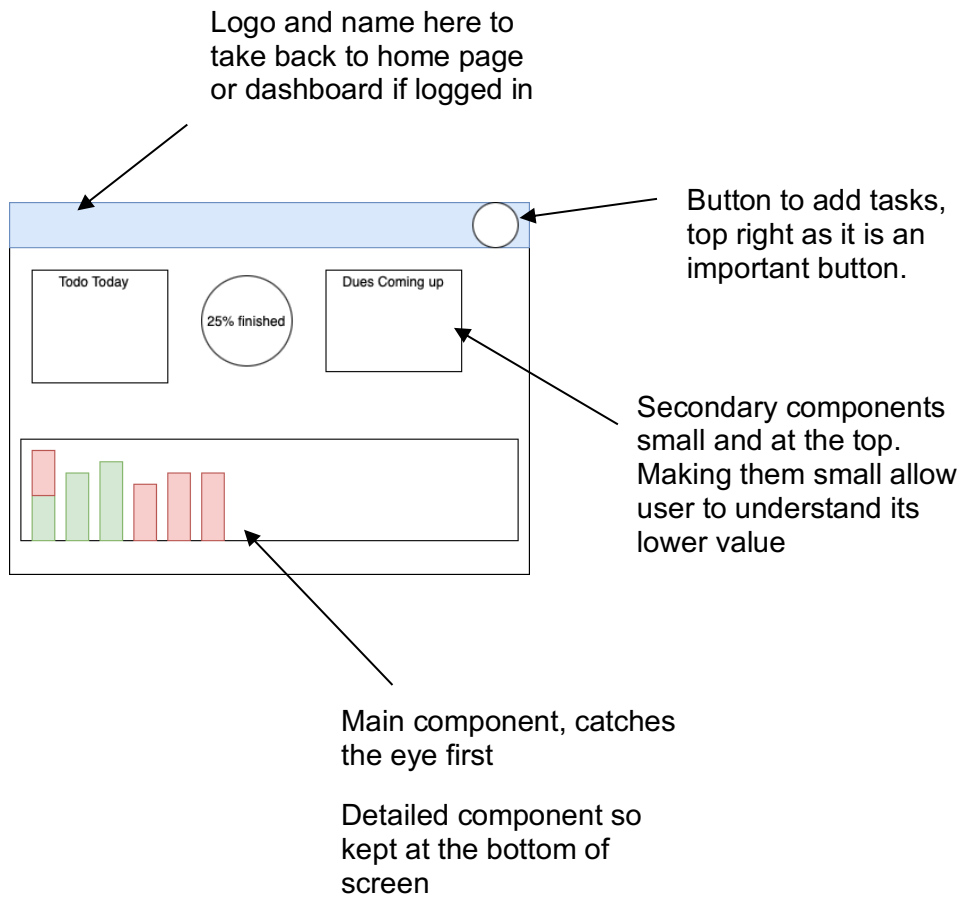
Feedback: The page looks simple with just some more summary above the bar graph. The bar graph shows the schedule of the user. Maybe adding quick features to edit tasks might make it more interesting.

Out of the three, I have decided to go with the last one as it would represent the schedule in a way that I have already done before and would not be over complicating the display. Even though producing a calendar would allow for more user friendly features, there is the risk that I might not be able to finish before due date. Hence I will be sticking with the second bar graph version.

By allowing users to pick their own colour this model can be made accessible to colour blind people. With very less text but more illustrations, this model would allow dyslexic people to use the website without much trouble.

Only minimal user data needs to be accessed from the user respecting their privacy and when it is taken from them, it will be shown as a form. If it had been the calendar I would have needed access to users calendars.

User Interactivity with Dashboard:



Further research on Django and how it needs to be implemented with a backend 'brain' that requires inputs from databases, showed me that I would need to use Django rest framework. This would produce an API for the database and would allow me to easily take data from the database in the form of a JSON dump as well as update it easily with just a python dictionary which is what I already produce.

The database service that will be used in the trialling and development of the webapp will be SQLite3. This comes in default with Django and it is simple to use. I would, however, need to shift to a different database service such as MySQL or PostgreSQL later towards the production stages. This is because of the disadvantages it presents. The following include some of these from my research:

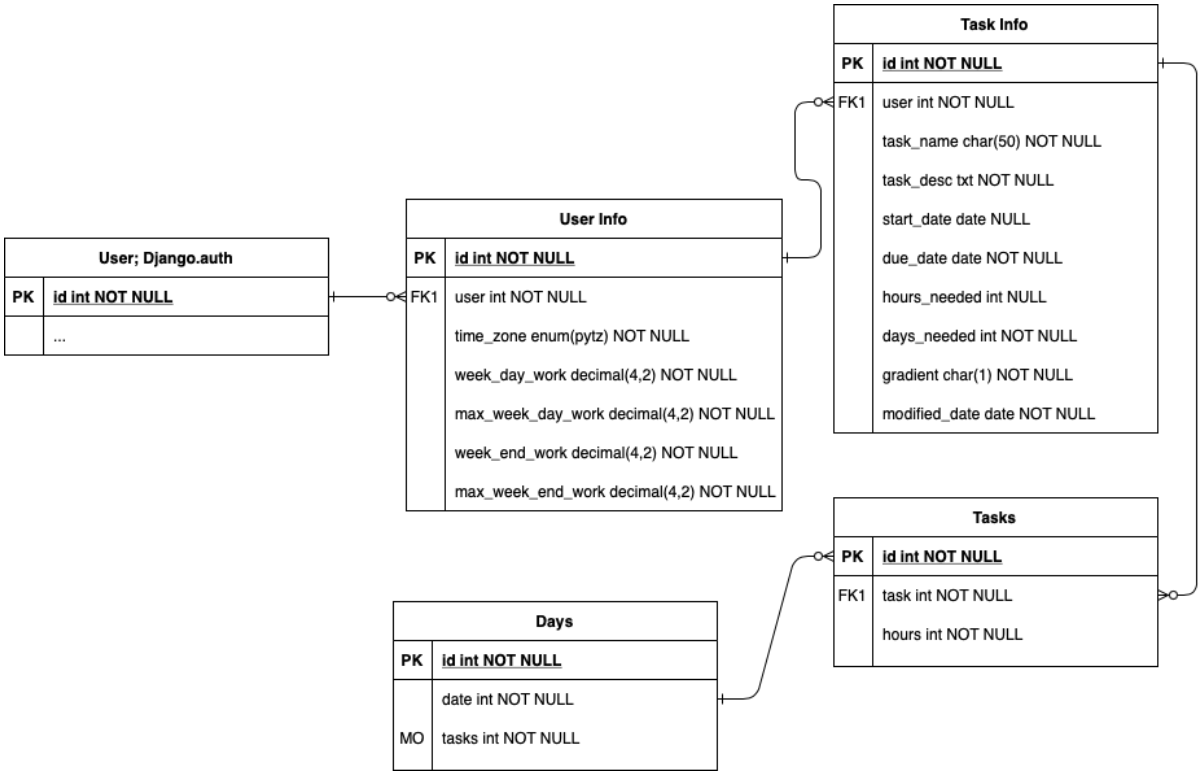
Limited concurrency: Only one process can make changes to the data at a given time, this would mean that there cannot be many users using the service at the same time.

File permissions: Because SQLite writes data onto an ordinary disk file, database permissions that might be different for different users is not possible to achieve and is often a poor choice for such kinds of applications.

Security: With a database engine, there is going to be more protection from bugs than a serverless database like SQLite. It will also make data access more precise and also allow fine-grained locking and better concurrency.

In Django I would need to create two Django apps called *accounts* and *scheduler* for the two different functions. The accounts app will take care of user authentication while the scheduler will have the algorithm.

Since it is necessary to have a database to run the program it is here that I began. First I created a user login and registration forms. These were not hard as Django comes with these by default. After this I moved on to do the Django database models for *scheduler*. For this I needed to have a relational database scheme. It has been described in this chart:



The User info model (or table) contains basic user preferences that are not related to *accounts*, such as their time zone and daily limits.

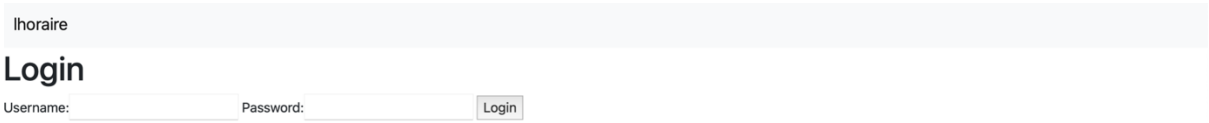
Task Info, which stores the information related to each task of the user has a foreign key to the User Info model. This table holds fields such as task name, due date, and gradient.

Tasks model, which contains information related to each *task instance* in each day has foreign key to Task Info. This model stores the number of hours a task has to be worked on in one particular day.

Days model stores the dates and the Tasks that need to be done on those days. The tasks field on this model has multiple to multiple relationship with Tasks, and allows for multiple options in each one of the record.

This type of relationship is necessary as it will allow for relationship between the task, each day and the user. The day's model would be the implementation of *schedule.json* file and Task Info model would be the *tasks.json*.

I created the login, signup and logout views in the *accounts* Django app. With this done I would now need to move to *scheduler* where I can do the views and models for the algorithm to work.



When tasks are added through a form, it would need to be added to the database as well as to be run through the algorithm. The algorithm would need to take older tasks that are present in the database as well. For this to happen I would need to run queries, take results and convert them to a python dictionary and feed it to the algorithm as this is how the algorithm currently is made to work.

However, with research I could find out that Django-rest-framework allowed this to happen efficiently and with very less code and effort from my side. Moreover, it was used by several different projects and is regularly maintained.

Django Rest framework will allow for the Task Info and the Days to be read and updated through an API and this will permit it to be presented in JSON form. This will be really useful for us as the algorithm has been built to take in objects as JSON / python dictionary objects.

First, I would need to create rest framework serializers that will convert models into python objects. These will also be used to deserialize a python object into validated data which is updated into the table. I would need to create model serializers involving Tasks, Task Info and Days, as these are the data that is required by the algorithm and which would need to be updated. However, this does not mean that I would need to use serializers every time, I would be using the default Django methods for creating and updating models for creating users, updating settings etc.

Because schedules come as a group of days, I would need to create a list serializer for Days model that will be used to update, create or delete individual records. This will also take care of adding multiple Task model objects into a particular Day record. Task Info on the other hand required just a normal serializer.

The documentation of Django and Django-rest-framework really helped me in this stage of development.

After this was done I will now need to find a way to connect between the algorithm and the backend of my Django project. For this I created a git submodule of the algorithm inside the directory of the *scheduler* app. My plan is to import modules from the submodule into the Django views in *scheduler* and create and update tasks.

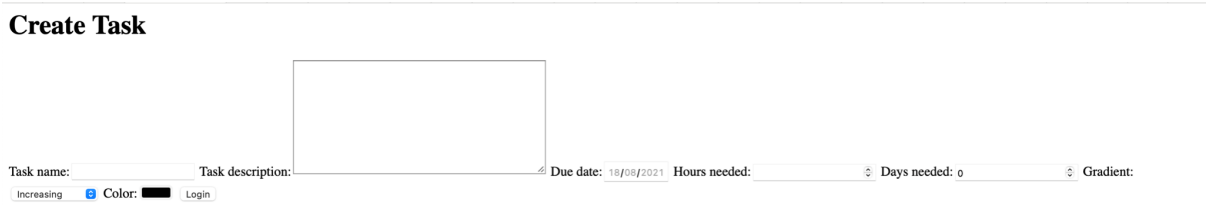
This was easier than I thought as I just needed to do almost the same I did in the *dumy_start()* function in a *create_tasks* view. This view will have a Django form into which users can add their new task and it will use the post data to create *Task Objects* and run through the *filter*.

Here I had two options: provide the user with a formset where they would be able to add how much every tasks they would want to, or show one form where only one task can be added.

I decided to make the form be only able to add one task. There were two main reasons for this:

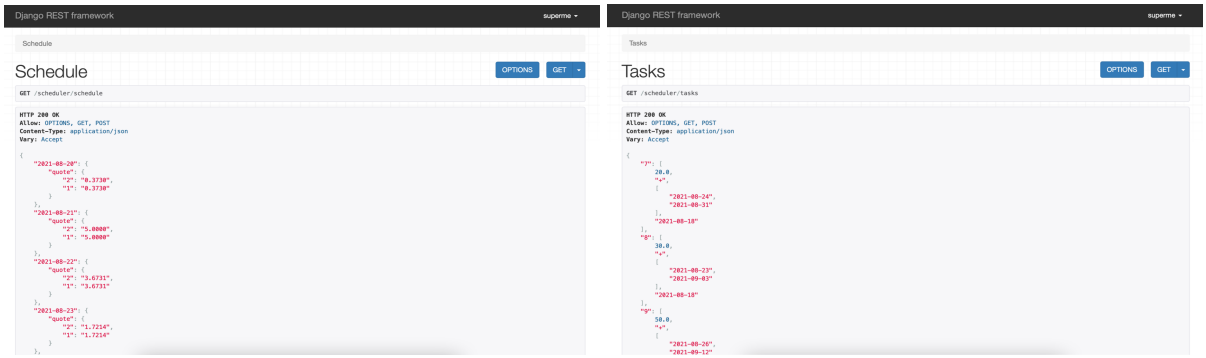
1. Adding several tasks at once could cause the algorithm to timeout if any long loops or errors happen in the code. To minimize this during the early stages of production where the algorithm hasn't been totally tested to be bug proof I would need to take this sacrifice.
2. Keeping the forms and inputs simple is the best way to make the application maintainable. There will be less chances for wrong data to come in and also lesser possibility for robots to attack the website (DDoS).

Besides adding tasks would not be hard for the user in the near future with having a way to fill the form without going to a new page. This is how the form looks right now:

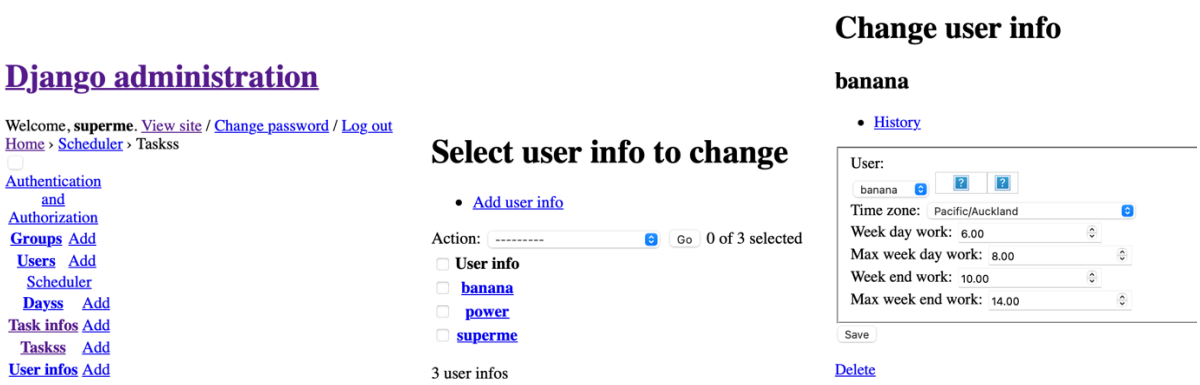


The form has all the basic requirements of task name, description, due date, hours needed and gradient. I also added the option to add any constraints on the number of days required. This is not needed for the algorithm to function. I added the required field of colour as this will make the tasks look distinct in the schedule bar graph.

Finally I was able to produce two views for showing the schedule and the different registered tasks in JSON format. This is the API by the Django rest framework. Currently it only performs the function of displaying the schedule produced so as to error-check. However, in the future it can be used as an alternative way to add/retrieve data.



Django's administration function allowed me to see how data is stored in the database and also to manage users.



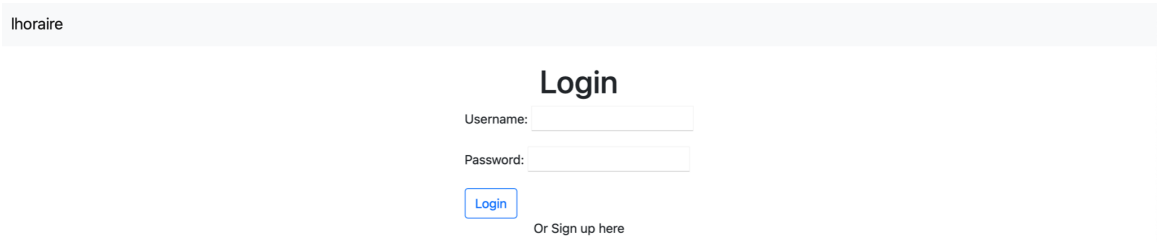
Review:

In this sprint, I was able to produce a basic website interface for the program. This meant that I was able to display the website to stakeholders. I was also able to test my algorithm to create tasks and store the results in a database. This also allowed me to ask for feedbacks from several people easily. Django administration helped me to debug the results of algorithm and also to see if the Django-rest-framework worked as desired and stored the data in the right format. Further I will look into making the website look better and also improve existing and add more features. I would also need to look into hosting the website somewhere.

Sprint 4:

The aesthetics of the web app is really important as it will decide how much users will prefer the service and also user time on the website. To make sure that the website is pleasing and also accessible without taking really lot of the time, I decided to use Bootstrap to design the webpages.

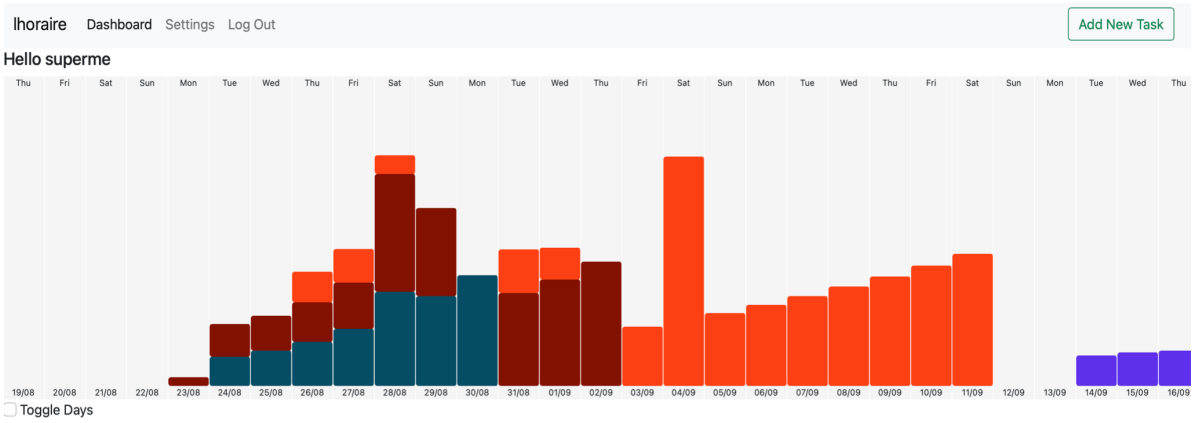
Because Django uses templates which are actually just html files, it would not be hard to insert external CSS and JS libraries. This will make my work really short and easy. Also since it is also possible to extend html files I will not need to rewrite a lot of codes that are otherwise repeated such as navbars, footers, external library imports etc.



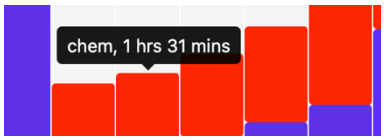
The login page becomes a lot better using bootstrap. I also needed to use flex boxes for aligning the contents. The Sign up page looks fairly similar to that of the login page with an extra input field to confirm password. The validation of these forms will be mainly done by Django Authentication, however, I will need to make sure that proper alert is shown to the user.

According to the plan, I used a div that takes the entire width of the window which has bars of constant width which will act as our bar graph. There will be smaller divs which represent

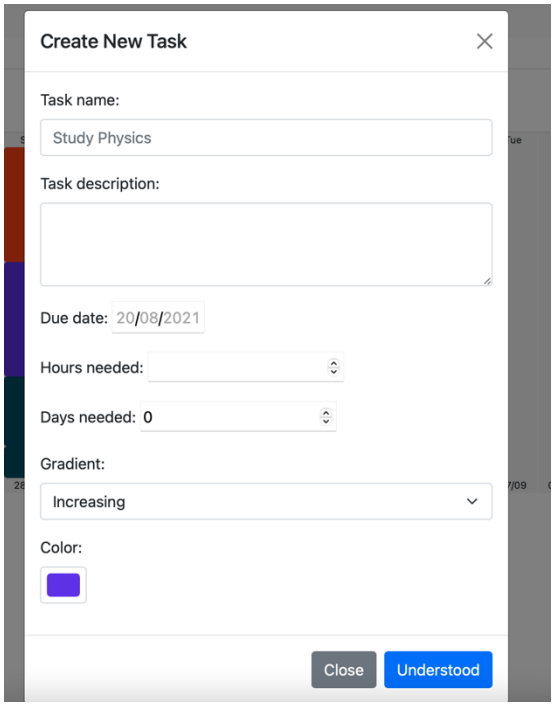
the tasks inside this div which is a flex box. I will be using the *flex-basis* style property to use percentage of time the task will be to represent it in the website. The maximum height of these containers would be the maximum of weekdays and weekend days limit.



Since the bar graph is showing the upcoming days in the calendar, it would need to have the day and the date which I have added in the top and in the bottom respectively. To improve the accessibility of the schedule, I also added pop-over text that will appear on hover on any of the task items.

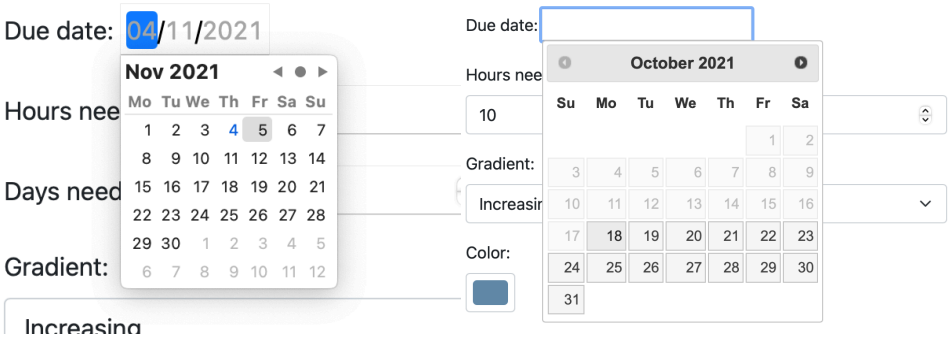


To allow users to add tasks on the dashboard itself instead of taking them to a different page, I added a bootstrap model that will pop-up when clicked on the 'Add New Task' button. This will contain the same options as the form that I made previously.



The date picker that I used initially is the default html, however, I came across several limitations that this had and had to take the decision to shift to a different one. The limitations include lack of customization such as limits on days that can be selected and that the design is too basic.

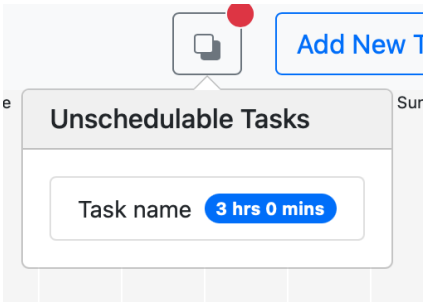
The date picking tool that I used is jQuery date picker. This allowed me to customize the dates that can be chosen and also make the selection easier for users. If it was a text input there would be ambiguity in the formatting of the date and errors due to that. This was done upon feedback from stakeholders.



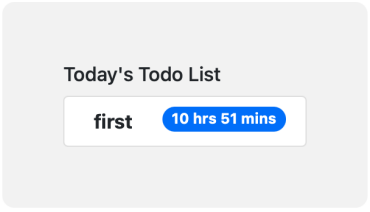
It also adds to the accessibility of the webapp. Preventing previous days from being selected would also reduce chances for errors.

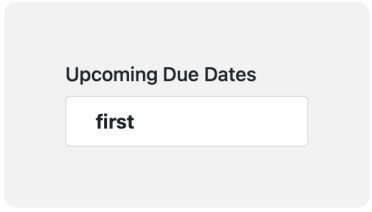
Colour selection is the default html colour selector. This will vary from browser to browser but will perform the same task.

To allow users to see the amount of hours that could not be placed into their schedule because of the limitations they have placed, I created a button which will open a pop-over. This will include a simple list of task name and the number of hours. There will be a red badge on the button when there are tasks in this list. The first time when a task is added to this task there will be an alert shown to the user.

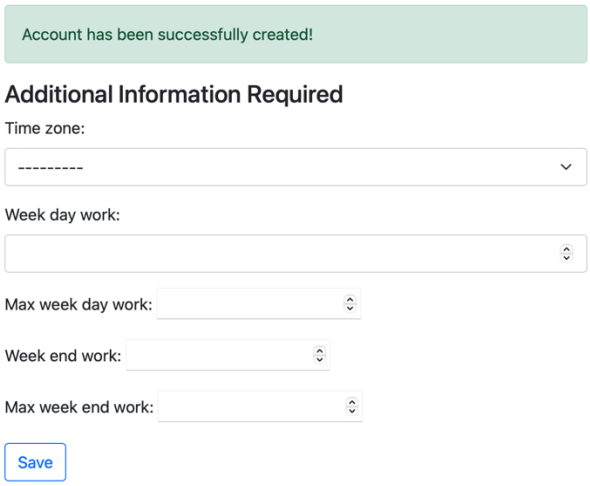


To show a summary of the schedule of the user I decided to add a To-do List for today and a list showing due dates that are coming up. These were placed in top of the bar graph display of the schedule. More details into these can be added later.





To collect user preferences just when they sign up I would need to create a form. This will fill the User Info model for the particular user. Initially this is how the form looks like:



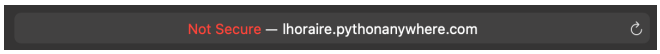
Hosting:

At this stage, the basic features of the webapp need to be tested further with stakeholders and there will be no better way than to host a pre-alpha version of the website so that it can be shared around.

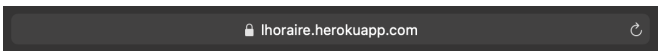
I had several hosting services to choose from. Researching around and chatting with a fellow developer, I found out that *pythonanywhere* is a good choice that people use to host python backed web applications. Uploading project files into the server was as easy as doing a git clone. However, there were certain limitations of this service that did not permit me to host it here.

First off, my algorithm ran really slow in it. This is understandable as there are several loops and nested loops, however it felt as if it was slow beyond use at some trials. Some trials even exceeded 15-20 secs. I might be able to reduce this by cleaning my code, but I was not sure if the slowness was related to the service and whether it could be reduced with a different provider.

Also *pythonanywhere* did not support SSL and showed a 'Not Secure' warning when opened. This also was a potential threat to the website when it would be hosted for production.



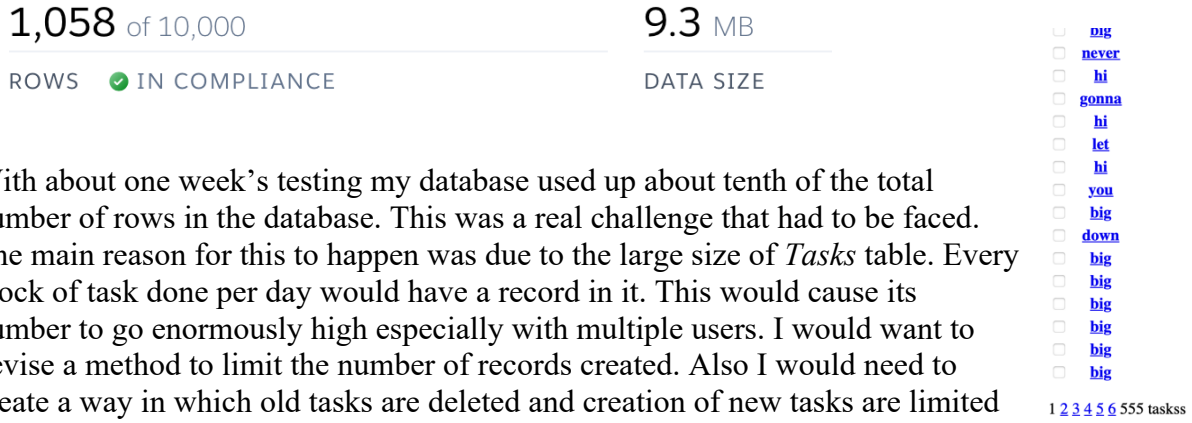
With further research and talks, I decided to use *Heroku* which also supports hosting python backend web applications. The service looked modern and the free tier seemed to satisfy all my needs. It also could be connected directly to my *GitHub* repository. However later I found out that, because I use sub-modules, I would need to create a *Heroku* git repository, which in the long run worked perfectly fine.



It also had SSL by default and I could run my web application with a secure connection between my clients and the server.

Upon speed testing, *Heroku* happened to run a lot faster compared to *pythonanywhere* for my use, with some instances of having about 10sec difference for the same exact scenario. This made me chose *Heroku*.

One big limitation that came with this service was that there was a limit of 100,000 rows in the database. Also the database had to be PostgreSQL. Leaving it to be SQLite3 would cause the service to wipe the database almost once every day which was alright during testing but not at all desired in production. Other restrictions included maximum 20 concurrent database sessions and 1000 *dyno hours* per month. *Dyno Hours* show how much hours the website can be active in users' browsers.

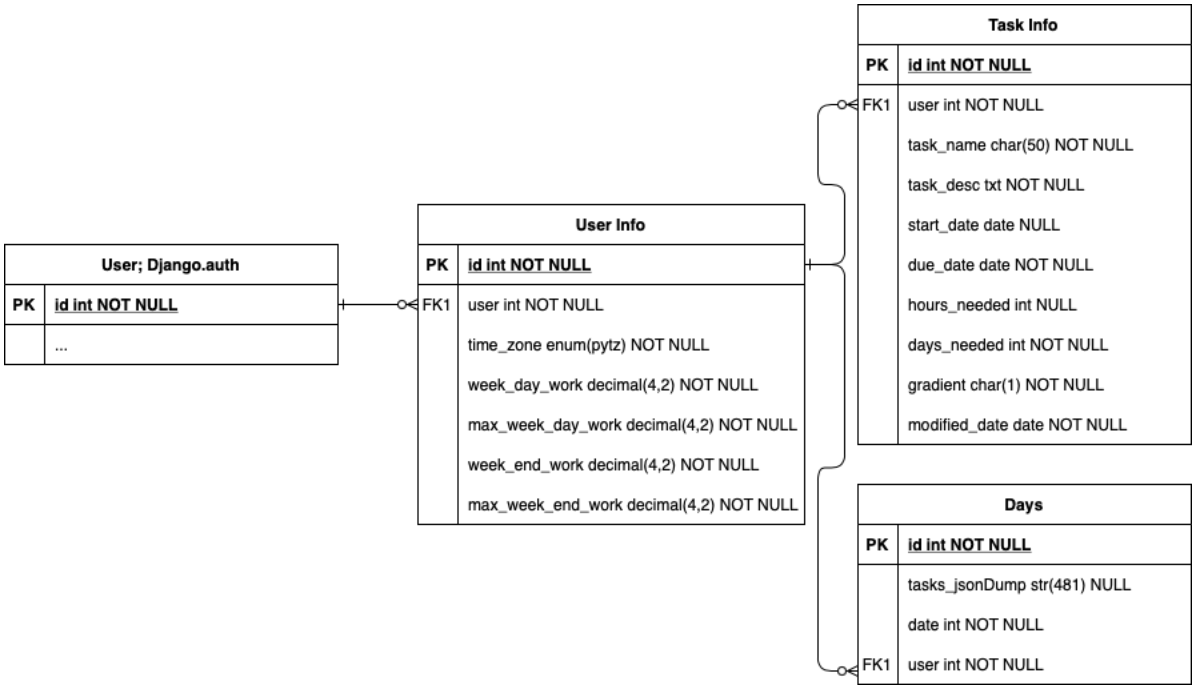


With about one week's testing my database used up about tenth of the total number of rows in the database. This was a real challenge that had to be faced. The main reason for this to happen was due to the large size of *Tasks* table. Every block of task done per day would have a record in it. This would cause its number to go enormously high especially with multiple users. I would want to devise a method to limit the number of records created. Also I would need to create a way in which old tasks are deleted and creation of new tasks are limited to a particular time window.

An idea that came into my mind was to store JSON dumps in the form of stings of the tasks as one of the fields in Days model. This would mean that every day would have only one new record added instead of a record for every single task that happens that day. This will dramatically decrease the number of records in the database, increasing the sustainability of this application. This would mean there will no longer be the need for a *Tasks* model.

However this will have the drawback of uneasy lookup and editing of tasks that are present in the JSON dump especially in the case of deleting the task. I would need to find a workaround that will allow me to perform all the necessary tasks. Talking with another developer I came to the conclusion that there was no other possible means to reduce the number of rows that are being used. The only other option was to pay for the Paid tier.

Here is the schematic of such a relationship between models.



The new relational database will include a *User Info* model that has a foreign key to the *User* model provided by *Django.auth*. *Task Info* model has the id of *User Info* as its user foreign key. The same foreign key exists in the *Days* model. This will allow *User Info* to have days and tasks related to them.

The *Days* model would have a *task_jsonDump* field which will basically be the JSON dump version of Tasks model in the old relational database. This will now help in reducing the size of the database.

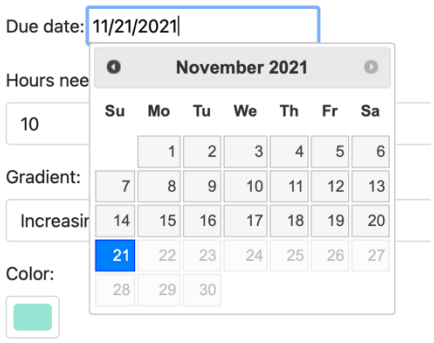
Since when a task will be updated the entire group (of union) of tasks is activated, the JSON dump is certainly going to be overwritten. The only time when task groups are not activated are when a task needs to be deleted. When this happens, the task data from *Task Info* database is deleted, this will make sure that this task is never called later. The database view which will take display the schedule could just make sure that every task in the JSON dump has the respective task information in the Task Info model. This will make sure that the deleted tasks are not shown. These tasks will be overwritten/removed when a new task is added which falls into this group.

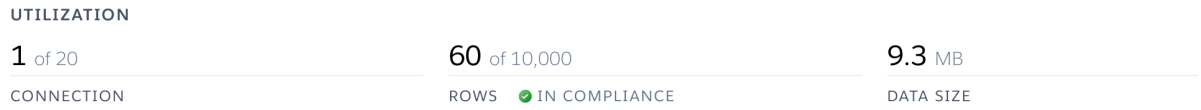
Careful calculation was done to make sure that the character width of the JSON dump field is exactly how much is needed. For this a new limitation had to be made: minimum task duration per day is 20 mins.

JSON formatting is similar to a python dictionary. This will allow us to calculate what the maximum width of this field would be. The maximum no of hours that can be spend per day is 24 hours. Which is 1440 mins. With tasks as small as 20 mins, the maximum no of tasks that will be done per day is 1440/20 = 72 tasks. Given a limit of 9999 to be highest possible task id, and two decimal rounding for task hours, each task would have a width of 12 characters. This including the comma that has to be present in the JSON object, it will be around 13 characters per task. 72 such tasks will mean the maximum possible width per day is 936 characters along with the end braces and one comma less. This will be the width of

that column. It has to be noted that this is a really rare case where there are almost 72 tasks each worth only 20 mins, and there is very less chance for such a long day field to occur.

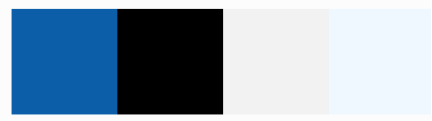
There is also the limit of making tasks only 3 months in advance and also to delete old days automatically whenever the user refreshes the page. A limit of the number of tasks a user can add could also be implemented.





Back to the website:

The colour scheme of the website was decided to be simple and light in the background with cards, navbar and the footer standing out with grey background colour.



The reason for selecting a light colour for background is the fact that the user has the choice to select the colour they want for the task bars and having a light background seemed like a sensible way to allow as much colours by the user as possible.

Feedback: When asked how this colour would fit the website by presenting an unfinished prototype to stakeholders, there were concerns whether the contrast difference between the two light colours were good enough. To fix this I made the grey that is going to be used for the nav bar and the footer darker.

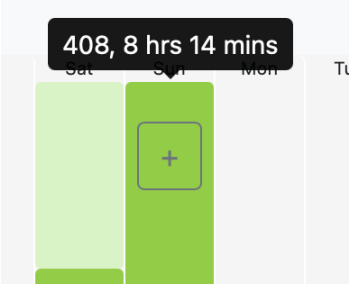
This decision was later confirmed with a student studying design technology and it was made sure that user's will not face difficulties accessing the website. His feedbacks included making sure the same colour scheme was used all throughout the website and also in the logo.

Also I will make sure the text shown is always readable with light background and dark font.

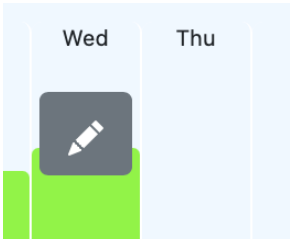
At this stage, tasks hours that could not be added to the user's schedule is listed in the non-schedulable tasks list. I would now need to find a way to re-add them to the schedule. For this the user currently basically has 2 options now:

1. Change the due date of the task that is overflowing
2. Increase the daily limits in the settings.

Upon discussing with peers, to allow users to add hours to one or some of their days there needed to be a way other than changing their daily limits in the settings. Looking at how the day filling algorithm works, I might be able to adjust the extra hours that are given to a day at the stage when its upper limit is considered.



For this I have made a button to show up when there are non-schedulable tasks when hovered over days. Clicking this will show a model form asking how much hours to be added.



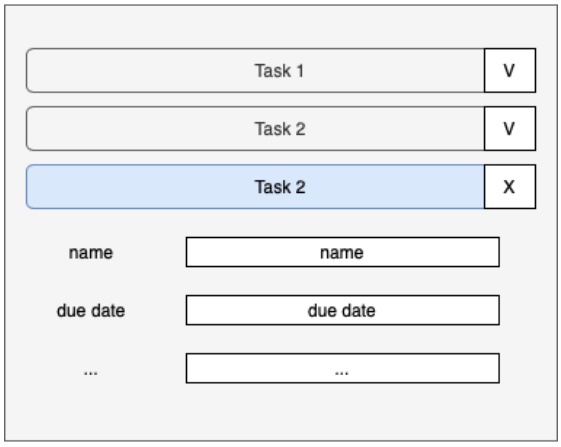
Once submitted this will change the newly created *extra hours* field of the particular day and also rerun the algorithm. The *day filling* algorithm will consider this extra hour and allow for extra tasks to be added to this day.

Days	
PK	id int NOT NULL
FK1	user int NOT NULL
	tasks_jsonDump str(481) NULL
	date int NOT NULL
	extra_hours int NOT NULL

Backend validation exists to make sure that when the new extra hours are added, the limit of the day does not exceed 24 hours.

With the way the algorithm built, even if the extra hours are given to a day that does not have the task that is been overflowed, tasks can be repositioned according to priority and cause free space in one of the days that has this task and allow it to be rescheduled. To allow this any day is allowed to have extra hours, except today and yesterday. However, adding extra hours to days in totally different groups will not solve the problem.

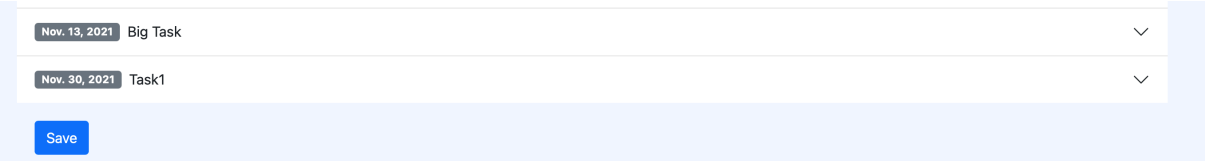
To edit the task info and also to remove tasks, I would need to create an 'Edit' page. I have named this page *All Tasks*. *Settings* page will be where users can update their preferences which they set when they sign-up.

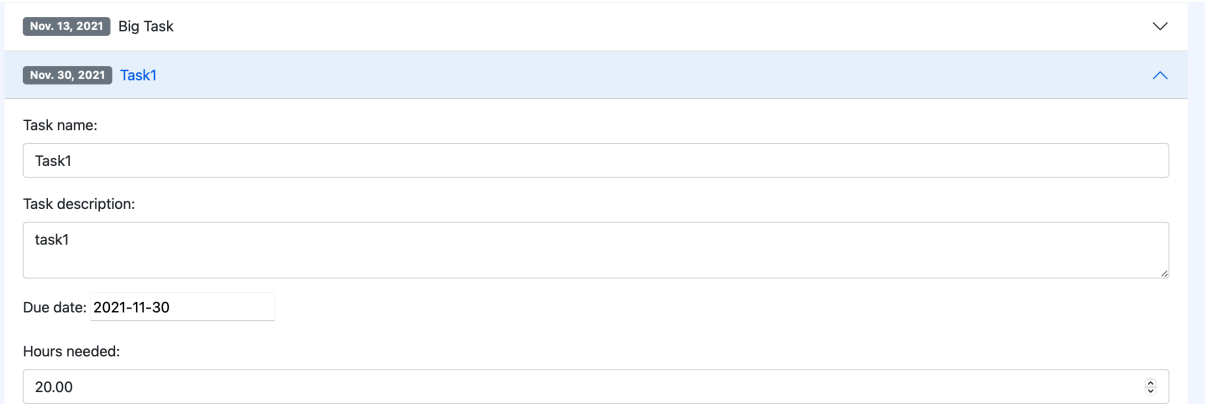


The edit page will show a Django model form set which will show the query set of all the tasks set by the user as a formset. These tasks will be ordered by their due dates. These can be edited to change the information in the database. As seen in the model, this view would have expandable titles for each task to allow the user to see which all tasks they have active and to edit all or some of them.

However before saving this data the algorithm might need to be run. When the hours that are

needed, due date, or the gradient is changed the schedule needs to be changed, or recreated.

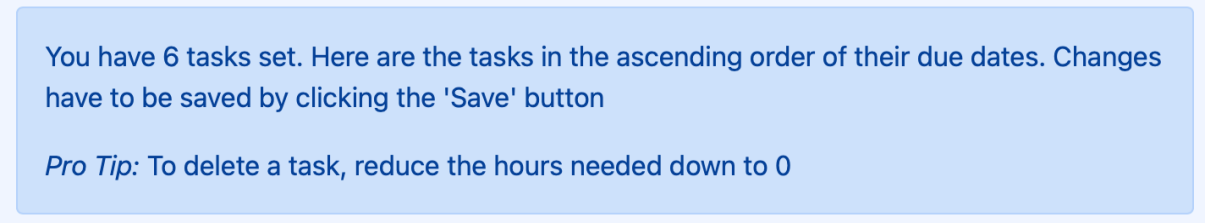




When a task due date is changed, the *filter* would require to activate the group the task was removed from and also the group it is added to. Moreover, it has to be made sure that parts of this task in today and prior are not removed.

When changing the size or the gradient, I would need to remove the task and add a new task with the new properties. The filter would then activate all the tasks that are in the old group as well as any other tasks (or group) that are in contact with the modified task.

Feedback: For the edit page, there needs to be a bubble message, instructing users how to edit the tasks and summarizing how much tasks they have. This will improve the accessibility of the website.

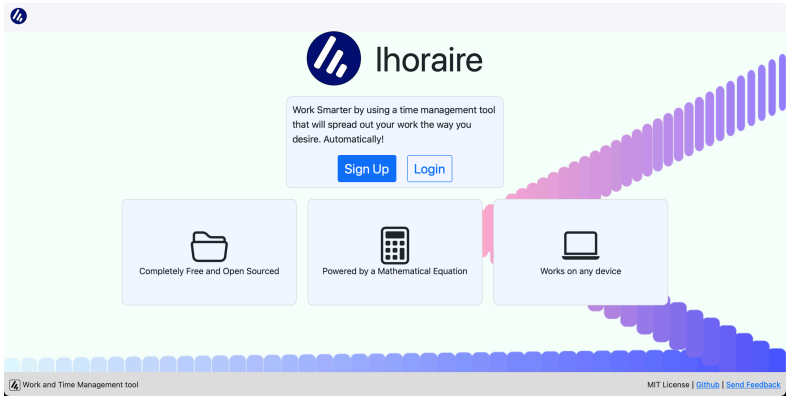


Settings page has had some changes with peer review. The time zone is now automatically picked by default when the form is first loaded. Also the labels are more clear as to what is meant and the fields with proper types and limits.

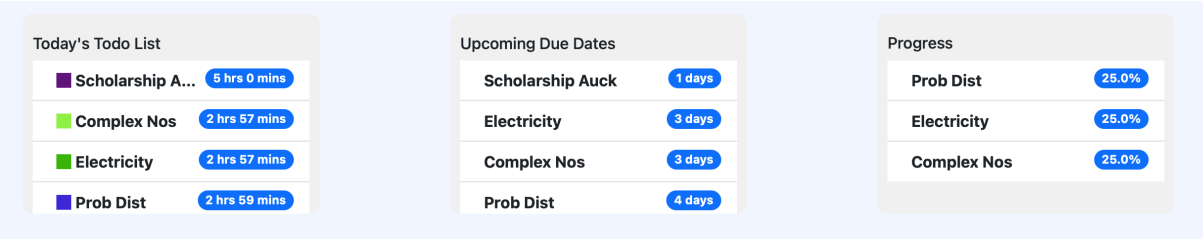
There is also backend validation to make sure that the daily limits are:

1. Within 24 hours
2. Maximum limits are always greater than normal limits

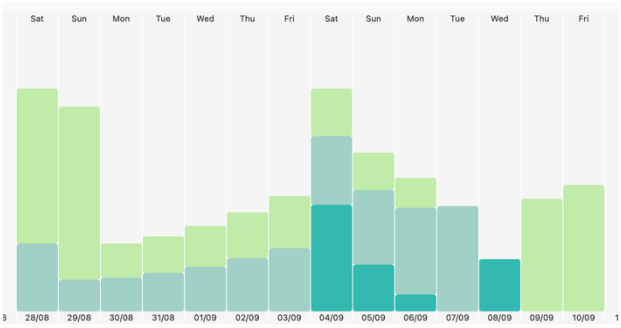
A home page to welcome new users is mandatory for any website, and thus, upon a critique from a stakeholder, I created a fairly simple home page that will briefly explain to the viewers what the service is all about.



The dashboard components have been updated. A progress tracking component has been added. This will calculate how much of the task that has been started has been done. Also the layout of the cards have changed. Now the title takes a top-left of the card with the tasks having almost the entire width. The tasks colour have been added in the to-do list. To prevent over complication with too much detail the colours were omitted in the other two cards.

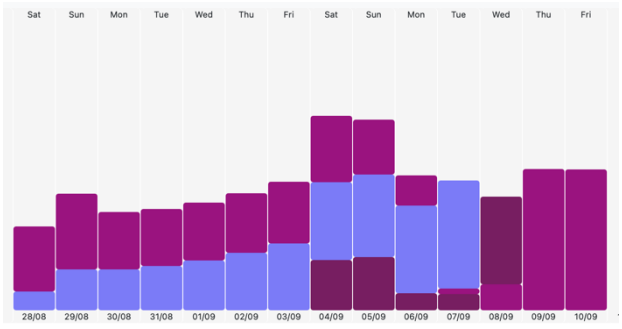


Algorithm Corrections made:



It was brought to my notice that day filling algorithm produces some undesirable results such as having distinct tall peaks during weekends or during an individual day. This is due to the way days are prioritized while *day_fillling*.

This could be not wanted by the user as there are other better ways to organize their work so there is no pressure in some specific days. For example, in this schedule, the weekend days 28-08 and 29-08 have a lot of work to be done compared to the following weekdays into which task can easily be transferred to in real life.

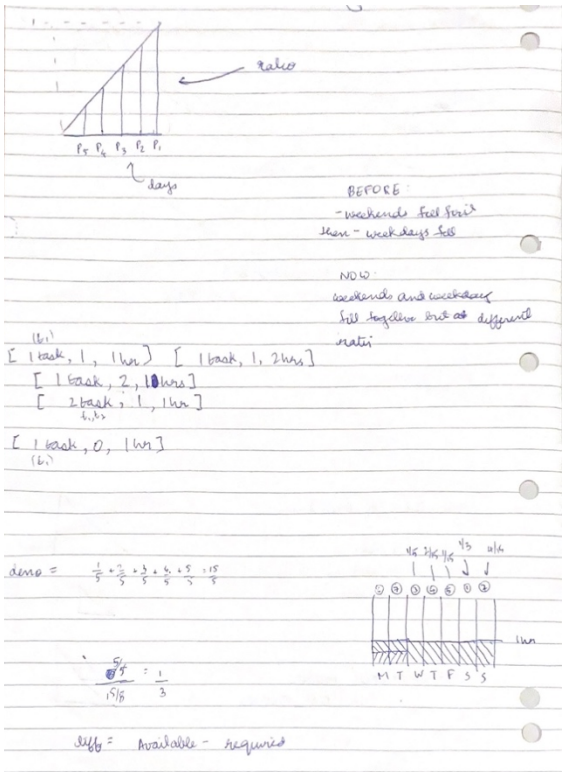


To overcome this issue, I would need to find a way to spread tasks hours across small groups of days or all the days that has the task going.

I decided to spread tasks onto groups of 5. These groups would be made based on their position in the priority list. Tasks on *to_reschedule* are iterated through the group of days which has that task.

Till now tasks are added to a particular day based on the proximity ratio. However, there is also the need to have a minimum amount that needs to be added this will make sure that hours of the task are spread out in the whole group. But the task cannot be spread out in equal measures as the group has days with decreasing order of priority.

For this I formed a different proportion system where each day will be allotted a particular number of hours of that task irrespective of whether they will or will not be able to fill. If



they are able to fill it, it is checked with the proximity ratio and if it is less than the proportion set by it, the amount will be added. However, if it is not able to fill it, then how much it is able to take, whether the proportion by the proximity ratio or the available space, will be filled in it.

This proportion system works as follows. Each day in the group of five is arranged in ascending order of their priority along the x-axis; with the least priority at 1 and so on. A straight line is drawn from the origin to y=1 at the x of the highest priority day. The proportion of the task that a day will be assigned will be the height of the straight line function at the respective day, divided by the sum of all the respective heights of all the days in this setup.

$$R = \frac{g * p}{5 \sum_{n=1}^{5} \frac{n}{5}}$$

Where g is the task hours that need to be scheduled, p is the priority of the day (1-5, ascending), in a group of 5. 5 can be exchanged with any other number.

Using this modified minimum hour requirement makes sure that the schedule is more spread out and more likely to make users accept the result.

Feedback: “I would personally prefer the second as it shows consistent hours per day.” As this change was a substantial change to how the algorithm would be acting, I had to take several opinions. Almost all of them were positive and supported this change.

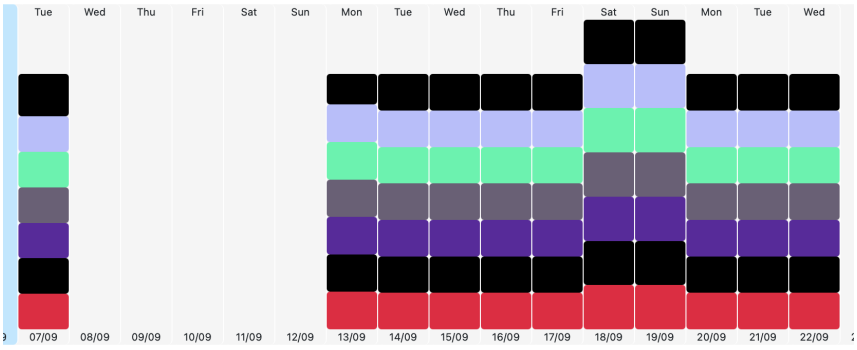
The only concern that I had was that this could potentially cause the code to run slow than normally. As it was grouping different tasks and iterating over and over, it could run into errors that might cause the loop to either go endless or to produce a biased schedule.

However, seeing the positive reviews I decided to go on with this change. I would need to regularly maintain this part of the algorithm, test and debug to make sure things are working as required. This would be a constant process along with stakeholder feedbacks, to produce the ideal error free product.

Moreover, to make sure that there is fair order of adding tasks, the order of filling days previous to the start day and till it reaches daily maximum limit has been changed. Now they will be done one loop of each after the other. This will mean that for example, after filling past 5 days, the algorithm will fill to the maximum for all the days including these 5 days before going for another 5 days back.

Further Debugging:

Really small tasks are being placed in the schedule. This can occur if the user has several repeated tasks of the same size and due dates. This will give it the same priority throughout the schedule producing such results towards the end. To prevent this from happening the code should deny placing tasks if the total hours is less than a specific length. This was decided to be 20 mins. If a task on a day is less than 20 mins, it will not be placed and hence removed.



This error was caused due to the new day filling code to group days by 5. Because the code will now try not to include tasks less than 20 mins and since task is now spread across the group, if the remaining hours is less, it will be divided causing them to be less than 20 for individual days and hence making the code skip them. The last group in this case is just 1 day long and hence there is no dividing of hours and the total happens to be greater than 20 mins.

To prevent this from happening, I modified the code to iterate through each group, but each time it will drop the final day. This will make sure that the minimum requirement of each day will reduce until it is down to one day where all the task hours have to be displaced into. If still it cannot be filled, the loop would move onto next group, until either there is no group or the task gets over (less than 0.001).

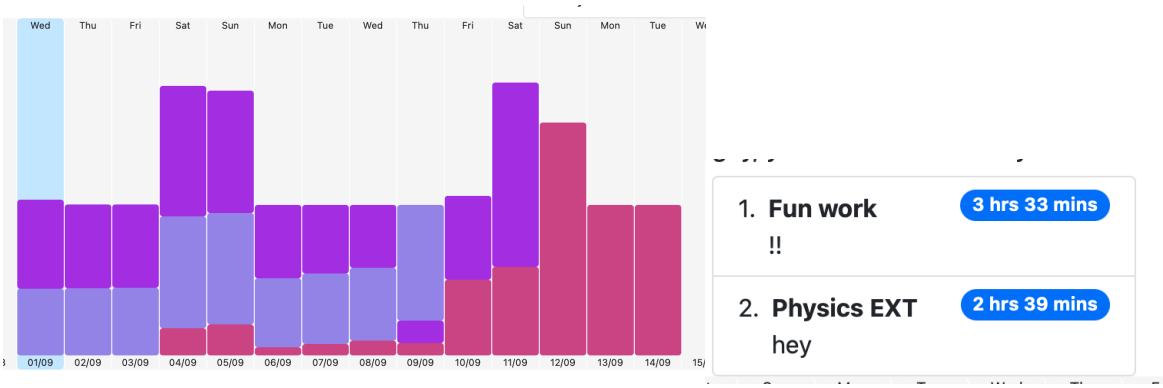
This usually occurs when a lot of tasks have the same due date and same hours needed. To reduce the chance of this occurring, I limited the number of tasks to have the same due date to be 8.

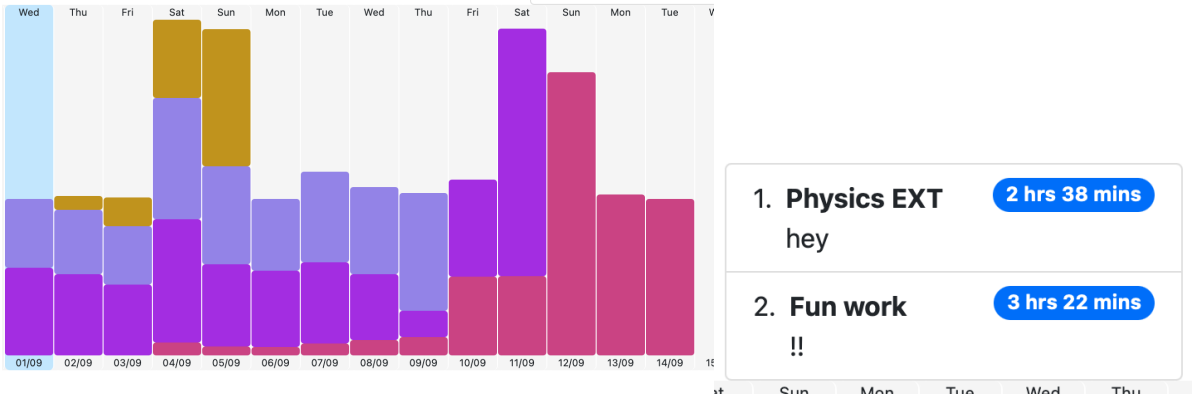
A: There are a lot of free gaps in the schedule. These have to be filled first before taking into precedence.

B: Chunks of tasks separate out from the main flow and is really away from the due date.

This error was due to not keeping a record of which all tasks that were filled on a particular day and mismanagement of proper filling of all tasks. This was a major mistake in logic when converting from individual day level filling to group filling.

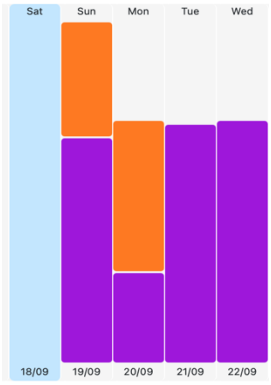
To fix this I had to create a *taskfiller item* object. This will be a collection (list) of all the days with the tasks that are done on these days. When a particular task amount has to be added, the proportion added would be based on the information present in one of the day objects in this collection; the 'right' of this task on that particular day is removed when the task hours have been added. This will make sure that the next task in the iteration would have proper time allocation.





When a task group is updated by for example adding a new task, it was observed that if there are tasks in this group that has to be done today, they too get updated. This is never desired as in certain scenarios this can cause the user's current to-list to be adversely changed if tasks are displaced into today.

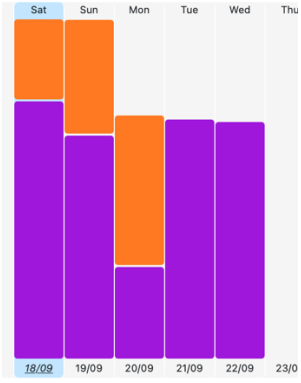
To prevent this from happening, I would need to make sure that days that are today and before are read-only days. This will prevent them from being updated. To make this happen the amount of hours of a task (that has a part to be completed today), should be reduced from the total number of hours the task needs. If this is not done, then when the task is refreshed later, all the task hours will be included in the calculations.



Task Info	
PK	id int NOT NULL
FK1	user int NOT NULL
	task_name char(50) NOT NULL
	task_desc txt NOT NULL
	start_date date NULL
	due_date date NOT NULL
	hours_needed int NULL
	total_hours int NOT NULL
	gradient char(1) NOT NULL
	modified_date date NOT NULL

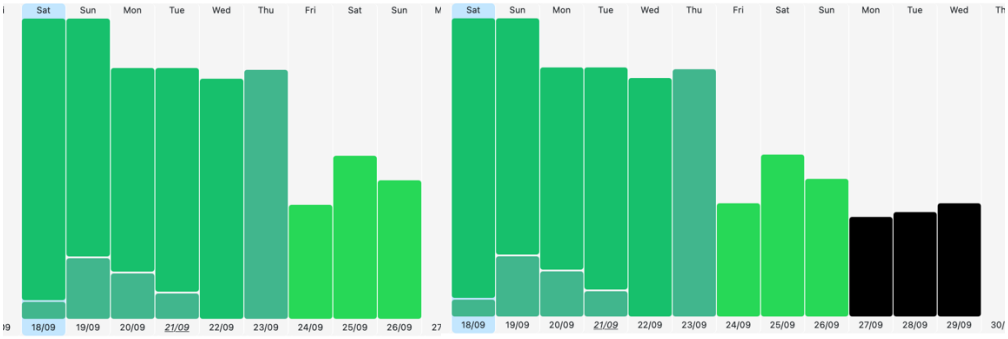
To allow for this to happen, I had to include a total number of hours field in the *TaskInfo* model. The hours needed field will be updated each day, on page refresh, to avoid this error in usability.

The *Progress Percentages* dashboard component was a use of this new implementation.

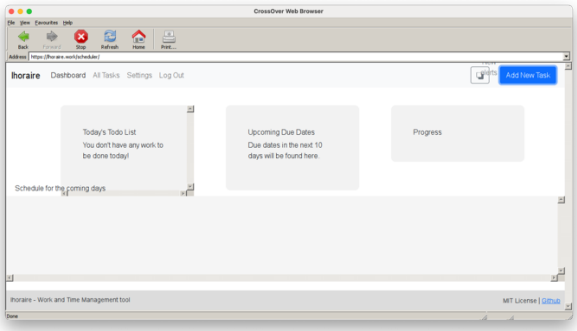


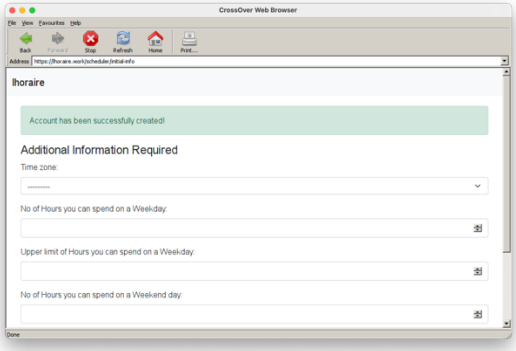
Adding a task next to a task that is happening today made that particular day's work along with older work if exist to disappear. This was because the code was activating that task and since it would have a start day of tomorrow (as no work is given to the same day as the day the task is added/refreshed), it would be overwritten and delete.

This was fixed by updating the start date of the tasks in the *Task Info* model every day. This will make sure that the part of this task that is happening today and yesterday are disconnected when it is activated due to adding a task. Also it was coded to ignore previous days from being activated in the *filter* module. Here it can be seen how adding tasks does not cause read-only days from being removed.



Browser Compatibility:





Browser compatibility is something that is really necessary to be taken care off when creating a web application. I ran this website through different web browsers. Even though most of the desktop web browsers worked with the website, the older ones does not support JavaScript and is not usable. This would mean that users who would be using for example really old versions of Internet Explorer will not at all be able to access the website. This, however, will be a really small amount of people and I would believe that it would not affect a large portion of our stakeholders.

	Chrome (Version 95.0.4638.54)	**Safari (Version 14.1.2)**
Speed	Reasonable	Reasonable
Image load	First time there was a delay	Delay in first try
Cached login session	Works	Works
Styling	Works. Flex box had styling errors at first but after adding some specific styles attributes it worked.	Works but it require web kit

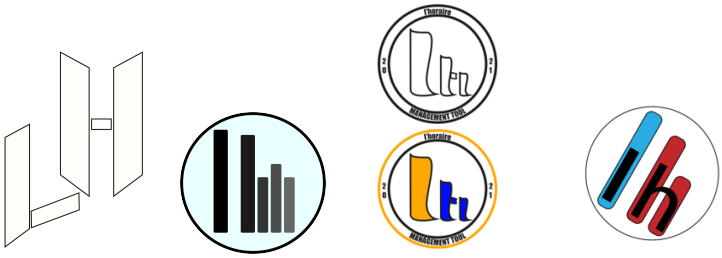
I was able to test with Chrome, Safari, Brave in macOS, and Chrome and Edge in Windows. The difference that I noticed was that the scroll bar was visible in Windows when it was mostly hidden in macOS.

When testing, it was observed that there is a delay on loading the website the first time. This is due to auto-sleeping “feature” that is present in the free tier of *Heroku*. *Heroku* has a maximum number of dyno hours that a website can be active during a month. This is around

Designing:

I would need a logo for my website. The logo needs to be simple and desirably show bars to represent the progress and schedule.

For this I asked help from one of my friends to give me some ideas for the logo. Here is some of the initial logo plans which were created from this:



These were the requirements:

- Should make a resemblance to the purpose of the app - tasks, bar graphs, work
- Should be within the colour scheme and design of the website - flat, blue, white, grey
- Should try to show the first few characters from the name of the website.

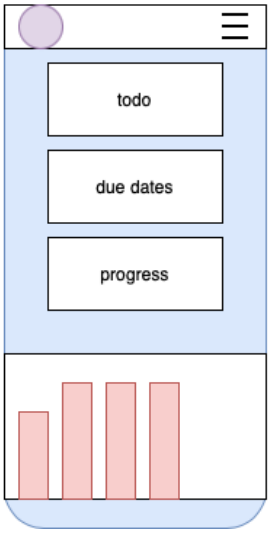
From these plans this final logo was created. The logo shows a simple flat circle filled with deep blue with white rectangles. These rectangles have rounded corners and represent the tasks that are added. It is also supposed to represent the first two letters of the application, *l* and *h*.

There are also some other logos such as:  ,  ,  which can be used for other locations within the website itself such as the footer.

Mobile Compatible:

Most of the internet traffic, around 58%, are from mobile phones. This will mean that I would require to take extra precautions when it comes to making the website pleasing to the majority of users. Currently the website looks like this, it does not support small screen sizes and is not properly accessible to users. I would need to make it usable from any screen sizes so that no matter how the user accesses the website they would be able to use it.

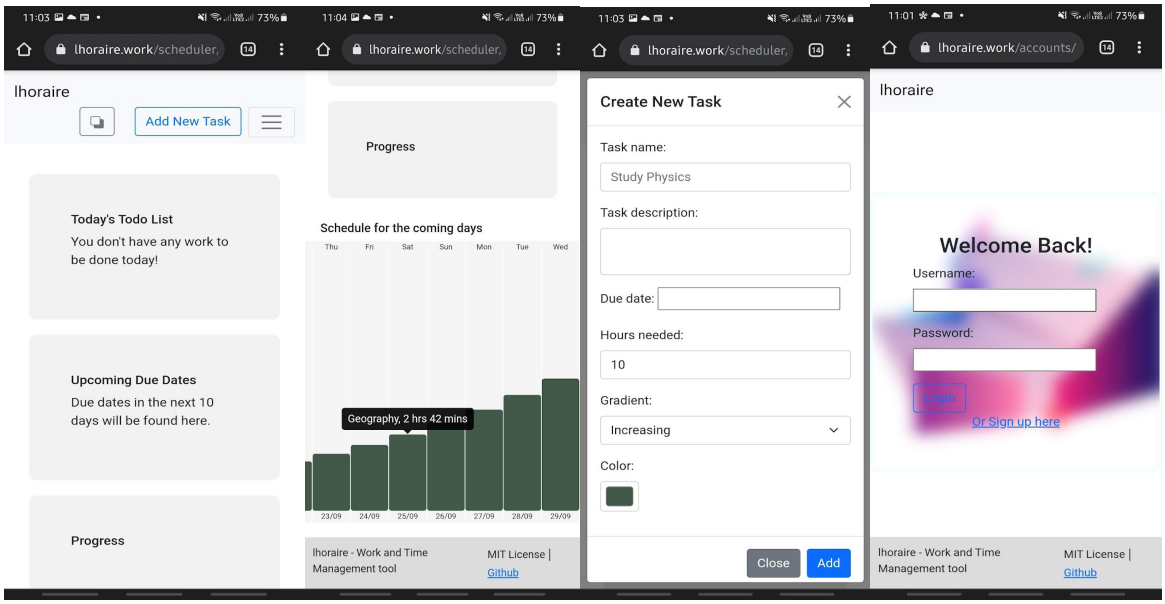
Here is a wireframe of how that would look like:



It would have the summary panels one underneath the other taking almost 80% of the width. Then it would have the bar graph showing the schedule of the user. The navigation bar would be simple with just the logo and the name with the burger button. I might also add the Add Tasks button and the reschedule button to reduce the number of clicks a user would need to do.

The bar graph would have the same features as the desktop version, just that it would be slightly larger. It would need to be scrolled horizontally to view the days that are overflowing.

Results:



Review:

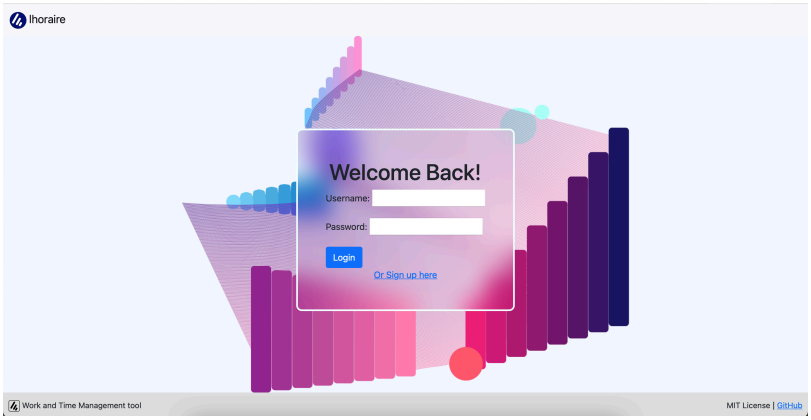
In this sprint, I was able to host the website as well as add components and webpages that will support the functioning of the website. Because of storage limitations in the free tier in *Heroku*, I had to change the database relationship as well as add new fields to certain fields to make sure that the number of records is reduced. I was also able to do important debugging in the algorithm where I discovered and fixed broken logic and code. I could do a complete renewal of the *day filling* algorithm in terms of groups of days as it will help in better dispersion of the tasks onto multiple days than onto a single day. Apart from testing and trialling the algorithm I was also able to make sure the website worked in other web browsers as well as mobile devices.

Final prototype:

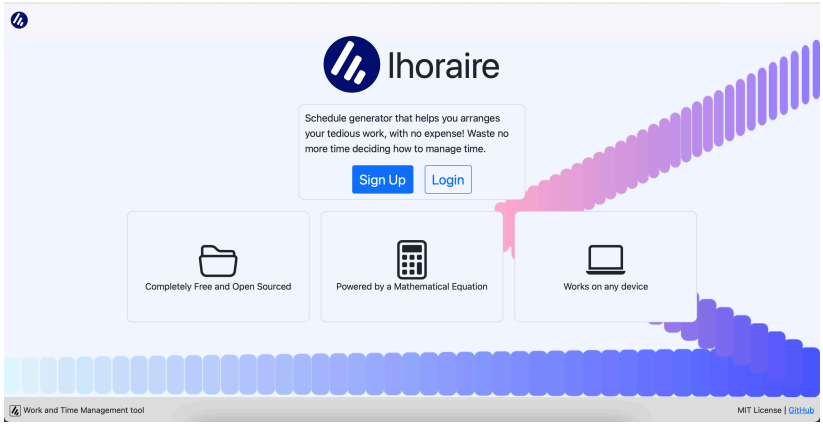
This is the final version of the website.

Dashboard Page:

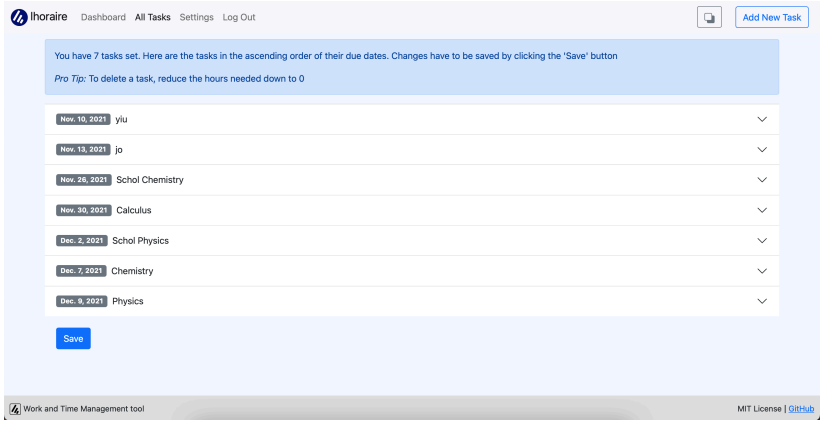
Login Page:



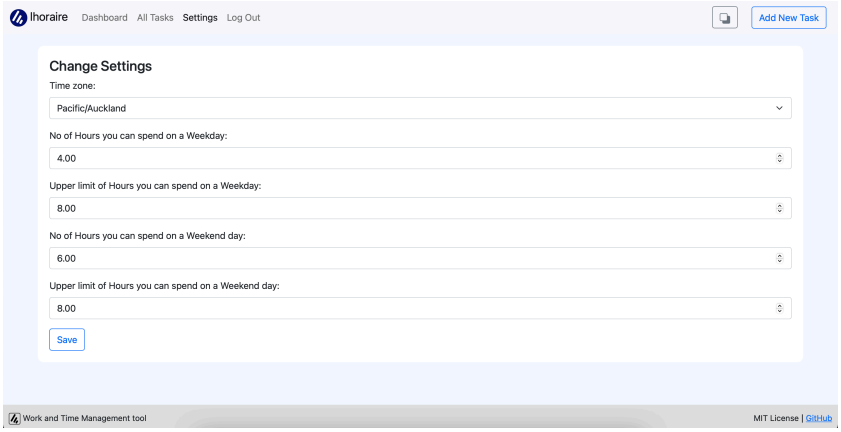
Home Page:



Edit Page:



User Settings Page:



This is how the website looks now. It has been tested by several stakeholders and myself and with several feedback I was able to finalise this might be the end of the iterative development for the first production version.

Further Challenges:

There are several challenges that this project will have to face in its developmental path.

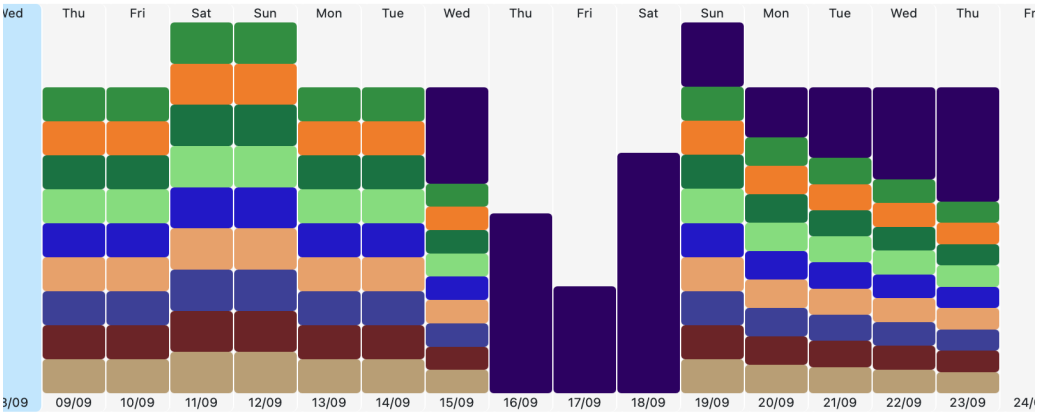
Decimal Significance

```
DAY REMOVED required_area >= available_area
Portion Used 0.0010009379999999994
```

Even though the time that is present in the schedule can be rounded to at least 3dp, doing this while performing the script can present some issues. For example, when checking if *to_reschedule* has hours of a particular task, it is smarter to check whether this amount of task is greater than say 0.001 hours instead of if the task is not equal to 0. This will prevent lengthy loops and allow the program to run faster and smoother.

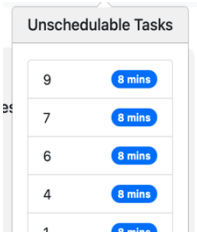
However, in some cases this could be an issue where a task receives lesser priority than others for just being later iterated in the algorithm. For example, in day filling where the remaining amount of hours is checked, if it just happens to be lesser than 0.001, then the program would consider there to be no space remaining in that particular day and hence drop that day from the list of iterables. This could result in that last task from not being shown in the same day with other tasks with same priority.

Priority



The priority system that is present in the algorithm can be made better in several ways. For instance over here, days 16-09 to 18-09 seems to be really high priority days for the purple task, this happens only because all the other tasks have equal priority and when tried to be placed in these days they do not result in more than 20 mins worth of work. This shows a shortcoming of the program as it would not allow for randomization except if user condition is so. Over here if there are tasks that could not be scheduled and they user added hours in these days, then there can be more tasks in them

Specificity



Because the algorithm tasks the numbers literally there are cases where the results are too specific for the user. This includes showing accuracy to the mins in the schedule which would not be what users want. One possible solution would be to make sure that the algorithm keeps each task item rounded to a perfect value while filling the days. This remains to be a challenge for now and needs to be individually inspected in the future.

Amount of Work in Hours

This was one of my initial feedbacks on the application; getting to know how much hours is needed to complete a work is harder than I first presumed it to be. In NCEA, it is an assumption that each credit of a standard is a result of around 10 hours of work. However, this is not in the official definition and would need to be used with maximum care. The number of hours needed always depend upon the student who is learning. If they find the subject hard, they might take longer time than average or vice versa. It also depends on which grade they are aiming to get.

Moreover, this application is not limited to NCEA students but is aimed for anyone who has to produce a schedule.

Upon peer discussion it was understood that it was indeed real hard to understand how much hours a user would need in a task. However, suggestions could be made by the application to the user based on previous collected data from users doing similar tasks. This does give rise to several privacy and ethical implications that would need to be addressed.

Final Analysis:

Like every other software, there is no end to the development or ceasing to the addition of new features. For a healthy program to be continuously and easily accessed by its users, there needs to be regular maintenance and debugging. Just like all other applications, even lhoraire is not complete. It has reached its first main prototype, but continuous debug and feedback collection along with users will define its future prospects and what it can turn into in the future.

As recognized in the previous section, there are several challenges that lhoraire still faces that could limit its usage to the wide range of requirements of users. With constant revision of algorithm and refining of code this can be achieved.

Because the algorithm can be considered as its own project, it can well be integrated with other project management tools such as *Trello* or *Notion* and can be used natively in those platforms. This will allow users to make use of lhoraire in their favourite application/website.

The API that comes with Django-rest-framework, can be used by developers or tech savvy users to build their own apps that use the algorithm. This would remove the burden of storing and/or transferring user task data.

A possible future for lhoraire would be to produce schedule in the form of calendars instead of just JSON objects. This will open up more options to integrate, such as with Google Calendars. With such a service, the backend could read the users calendar, detect any manually assigned work and use these as constraints for that particular days' workable time. It could also further develop to be able to create tasks in user calendars where each event is properly spread out and not conflicting with others and within the limits set by the user.

Privacy

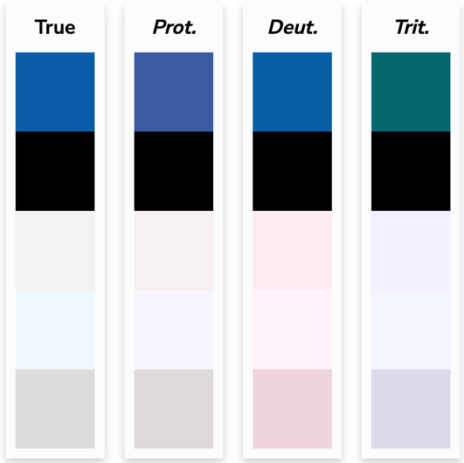
Privacy of users was considered really seriously in the making of this program. Collection and storing of only necessary information have been done and these too after complying with NZ Privacy Act.

There are only two level of rights for this app. One is the admin level and the other user level. Users are allowed rights over all their data, while admins can manage all the data. However admins are still not allowed to view personal information that are hashed such as passwords.

User Accessibility

User accessibility is key when developing any piece of software. Therefore to allow comfortable use for every users, I have made sure that the website is made properly compatible with needs of everyone.

Colour Blind: Users are free to choose the colour they want to even though the webapp randomly chooses by default. Users are also allowed to change the colours of the tasks. This will make sure that the website is suitable for every kind of colour blindness.



Testing for colour blindness showed that there is very less difference in the main colour palate of the website in such a way that would impair its use.

Contrast: Colours of backgrounds are light with appropriate contrast given to fonts and logos. According to WCAG requirements. Moreover, between task items there are white gaps, this will make sure that the colour contrasts are not a problem.

According to WCAG 2.0 level AA, the contrast ratio for normal text need to be 4.5:1 and for large text 3:1. The latter also applies between graphics and user interface components. These have been followed and tested with external software during the development and selection of colours.

Dyslexic: Font size is responsive according to the window. Font is also made sure to be sans-serif. Titles are not underlined but bold. This will allow users to read text easily. Also the contrast of font has been decided properly.

ARIA: Using Bootstrap components with ARIA attributes will allow it to be understandable by assistive technology such as screen readers.

Security

User data has been stored in properly secured databases that are stored on Amazon servers and would not potentially be involved in breaches. More so by using official Django authentication systems that are evolved over the years and has minimized vulnerabilities. Moreover there are regular updates to Django itself that take into account the security of its backend data management.

Using a HTTPS web connection between the webserver and the client would reduce the possibility for man in the middle attacks and any other types of sensitive information theft.

The passwords that are stored in the database are hashed and hence there is no way it can be accessed by others even the admins. This will reduce the chance of exposing passwords that are used elsewhere.

PEP8 Standards

Conventions and standards have been followed in the making of this project and thus the code would be readable and its maintainability is increased.

Maintainability

The code has been documented almost throughout and can be easily maintained with the proper understanding of the python libraries. I have also followed code conventions to make sure that others can understand my code properly. Just with any other program with database linked to it, this app would have to be maintained regularly in the future and corrected from

bugs that have been noticed. Because user's information are at stack at times, these actions would have to be done in a short notice and I think the organization of the code would allow for this.

Future Proofing:

The app can be used to model tasks which could have several applications in the everyday life. This would mean that there is going to be uses for such a program in the future as well. To allow the app to be used in the future, I would need to make sure the code uses up to date libraries and modules. This has been taken care of as of creation of the project. Also with database management that reduced the size of the records dramatically, I think the longevity of this app has been increased.

However, to make sure that this app runs and functions as it is supposed to in the future, there is the requirement of a proper maintenance strategy. Making the algorithm and the Django backend open sourced in GitHub allows for others to view the code and give feedback as well as provide help. This will be allowing this project to be caried forth in its development path.

Even though there still exist several areas in which the app can improve, which it would certainly in the future, the app has shown its potential to individuals who were testers and stakeholders:

"*This project well and truly did above and beyond my expectations. The time and effort put into the forming and developing of the formula that creates the time management itself is truly a piece of work. In terms of improvement, there could be more user friendly information, as well as an improvement in the design of the page. However for a small project, the design is simplistic and very usable at this current stage.*
The project could very well be used in the future. This project has many different avenues that it could be used in such as, libraries, Mobile Apps, or a time management tool for a big firm. I believe it definitely meets the requirements of the targeted audience as it has versatility of uses." – Jethro

From inception in paper to the first prototype, this project took more than 5 months to reach what it is today. In this process there were several people who supported me, including:

1. Mrs. McLay – for the directions
2. Mr Lander – for the opinions and feedback
3. Jethro – for beta testing and feedback
4. Daniel – for beta testing and feedback
5. Victor – for design ideas and feedback

---

[i] https://www.healthnavigator.org.nz/health-a-z/s/stress-at-work/
[ii] https://trafft.com/time-management-statistics/
[iii] https://development-academy.co.uk/news-tips/time-management-statistics-2021-research/
[iv] Steps can be found using Symbolab.com
[v] https://www.sympy.org/en/index.html
[vi] https://scipy.org