

Name: Joshua Gardner

Class: 2020

Project Title: Thrive, A Computer Aided Goal-Setting and Time Management System

Advisor: Robert Dondero

Introduction

Thrive is a web application created to teach time management skills to first generation low-income college students participating in the Freshman Scholars Institute program at Princeton University. The FSI program is a seven week intervention for these at-risk students the Summer before their freshman year. It has been found repeatedly that first-generation college students struggle with time management and goal-setting¹. Thrive was motivated by a conversation with Dean Khristina Gonzalez of Princeton University's program for Access and inclusion. Dean Gonzales requested that an application be created that will help students to break down their larger projects into manageable subgoals and then to plan and allocate their time accordingly. The project was originally worked on in Princeton University's Computer Science 333 course by a team consisting of Josh Gardner, Justin Tran, Victoria Zlatanova, and Hamza Mahmood. Initial requirements gathering took place in September and October of 2018.

Dean Gonzalez specified that the system needed to allow users to create goals and subgoals of a major task and vaguely that the failure to correctly estimate the amount of time needed for projects is a major problem that she wanted solved. However, she provided no suggestions for specific features. Since the initial requirements were relatively vague, I set out to explore

¹ Byrd, Kathleen L., and Ginger MacDonald. "Defining college readiness from the inside out: First-generation college student perspectives." *Community College Review* 33.1 (2005): 22-37.

psychological literature related to goal setting and task performance. In particular, Edwin Locke and Gary Latham's *A Theory of Goal Setting and Task Performance*(1990) and Albert Bandura's *Self-efficacy: The Exercise of Control*(1997) were both highly influential in the design of the app.

One of the most significant findings in the literature was that breaking down large distal goals into easily attainable proximal subgoals increases self-efficacy and the motivation to complete the goals, consequently improving task performance. Much of this is accomplished by lowering anxiety levels and decreasing task ambiguity, thereby eliminating much of the impetus for procrastination. In essence, the proximal subgoals seem much more attainable than the larger distal goal, and, because they seem more attainable, people are more willing to work on them ².

Additionally, the proximal goals allow people to see their progress. As each goal is completed, the person self-efficacy increases, leading to a sort of upward spiral in which they feel increasingly competent and thus exhibit higher performance as they work towards the distal goal. Similarly, Locke and Latham suggested goals must be specific measurable attainable relevant and time-bound in order to be effective³. Moreover, recent research has indicated that goals should also provide some strategy information or should make it explicit that strategies and information about tasks must be gathered ahead of time. They found that if individuals attempt to

² For a more detailed discussion see pages 100-120 in: Bandura, Albert. *Self-efficacy: The exercise of control*. New York: wH Freeman, 1997.

³ For a detailed discussion of goal setting theory see :

Locke, Edwin A., and Gary P. Latham. *A theory of goal setting & task performance*. Prentice-Hall, Inc, 1990.

complete a task prior to understanding the strategies necessary for successful task completion, this leads to demotivation and general decreases in performance. They found that when people set high performance goals without first knowing the proper strategies needed to achieve high performance, this leads to burnout and failure⁴.

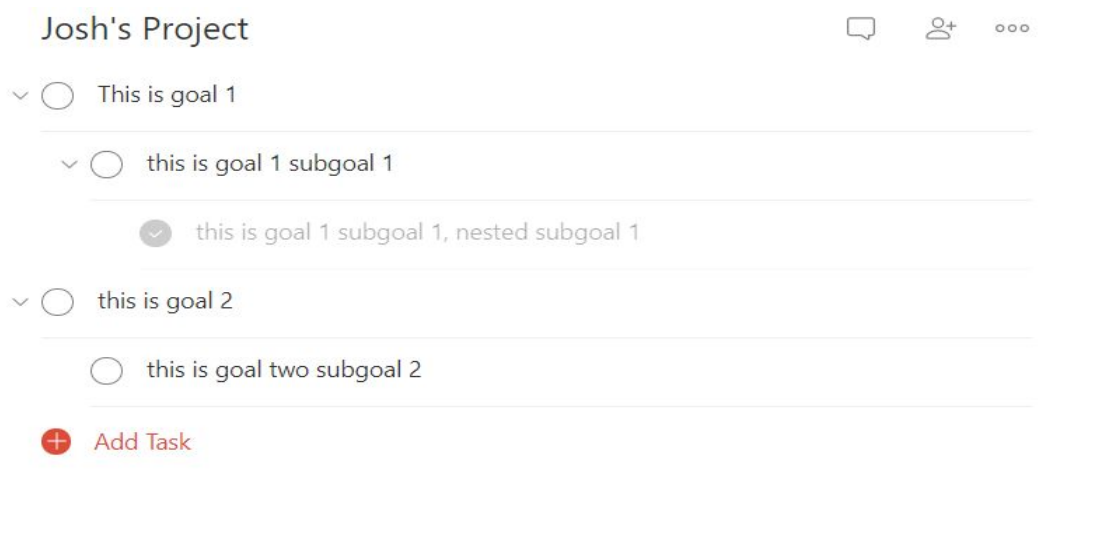
Taking all these factors into consideration, the engineering team prioritized the creation of nested distal and proximal goals, with clear indicators of goal completion, as well as the creation of time tracking devices, which allow users to estimate the time needed to complete tasks as well as the time, and to see measures of progress based on their time estimates. These requirements guided the design of the initial version of the app which was deployed in January of 2019. The app failed to satisfy certain basic requirements, thus motivating a major redesign. This is discussed more extensively in the next two sections.

Related Work: Existing Commercial Applications

The related work section will start by explaining the way which related work influenced the January 2019 version of the app. Next, we will discuss the January 2019 version as a sort of “related work” to the current iteration, since it is sufficiently distinct from the redesigned app but nevertheless inspired it.

⁴ Locke, Edwin A., and Gary P. Latham, eds. *New developments in goal setting and task performance*. Routledge, 2013.

Prior to engaging in any design, the team investigated existing apps which may have some or all of the required functionality. There were two apps with functionality relatively similar to what we were setting out to create. The first of these is Todoist. Todoist allows people to make a digital todoist with time-bound goals such that the user can set specific calendar dates on which they would like to complete goals. Notably, todoist also supports the setting of nested goals for projects, as demonstrated here using their “projects” feature:



Nevertheless, Todoist does not contain substantial time tracking features. There is no way for the user to input a time estimate for determining how long the project will take. Moreover, Todoist does not contain substantial completion measures for specific projects. The app does keep track of how many tasks of user completes in a given day but not necessarily for a given project. Moreover, the indicators of progress are relatively unclear, since tasks are unweighted. Ostensibly, this can be improved by giving the user more clear visual indicators of goal progress.

We also examined monday.com, a project management app used for teams in the workplace. Monday.com included many time tracking features, however it included no goal breakdown features like todoist. Most of monday.com's tools are intended for team use, such that team tasks for a given day or week are shown in a grid layout and the completion status and time estimates of each task are also displayed in the grid. In this sense, monday.com contains many of the features that Todoist lacks, but nevertheless is clearly not designed for the purpose of individual projects.

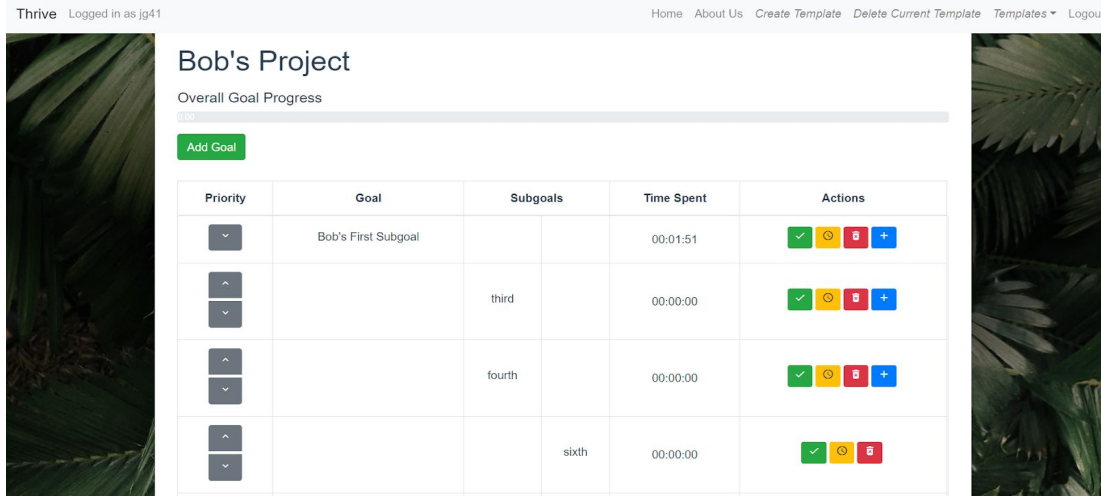
Related Work: Thrive Version I(September 2018-January 2019)

I set out to design an app which has the nested hierarchical goal-setting capabilities of todoist, the time tracking features of monday.com, and additional time estimating and completion tracking features which exist in neither application. The first priority was to implement hierarchical goal representation, similar to what is seen in Todoist. From the beginning, we knew this was going to be a major logistical challenge because of the fact that most sql-based databases do not inherently support this form of hierarchical nesting.

The project centered on the the creation of an app that supported hierarchical nesting and completion measures via a progress bar for the overall project. We encountered significant difficulties in implementing the database to support the hierarchical nested goals, ultimately having to resort to using Json strings as opposed to more appropriate database formats.

Essentially, we represented each project as a Json string which could be converted back and forth from a python object.

The python object had a variety of attributes and functions which enabled the code to interact with each goal in the goal hierarchy. The team was successful at creating something that supported hierarchical nesting. Nevertheless, the resulting code was extremely buggy. Notably, one of the core functionalities was supposed to be the ability to move goals up and down using up and down arrows respectively. However, using the up and down arrows frequently resulted in goals randomly jumping around in the goal hierarchy and other sorts of generally unexpected and seemingly undefined behaviors. Moreover, upon user evaluations at the end of the semester, Dean Gonzalez and others informed the team that the crude user interface made the app virtually unusable. The version one user interface is displayed here:



Version one used a table to represent the goal hierarchy and created indentation by using different columns of the table. This made for an extremely crude interface, which was described by Christopher Moretti as being representative of “early 2000’s web programming, nothing like a

modern web app.” Furthermore, the time tracking features were minimal at best. Each individual goal object had its own time property which was represented by a timer experienced in the user interface. Notably, there was no interaction between the times of subgoals and their higher-order goals. The expected and intended behavior was that the time recorded for each goal would be the sum of the times for its sub goals. However, the team failed to implement this functionality by the end of the semester for COS 333. At best, the team had created an app which have some of the features of todoist as well as an additional time tracking feature for each goal, but which failed to represent the hierarchical nature of the goals in a visually appealing way.⁵

Requirements Gathering

The failure of the previous semester’s project inspired me to spend this semester completely overhauling the app in order to make it distinct from Todoist and to give it the functionality it needed to meet Dean Gonzalez's requirements. In order to get the app to meet her initial requirements, it was necessary to make substantial and overarching changes to the client side user interface and significant but not overwhelming alterations to the server side.

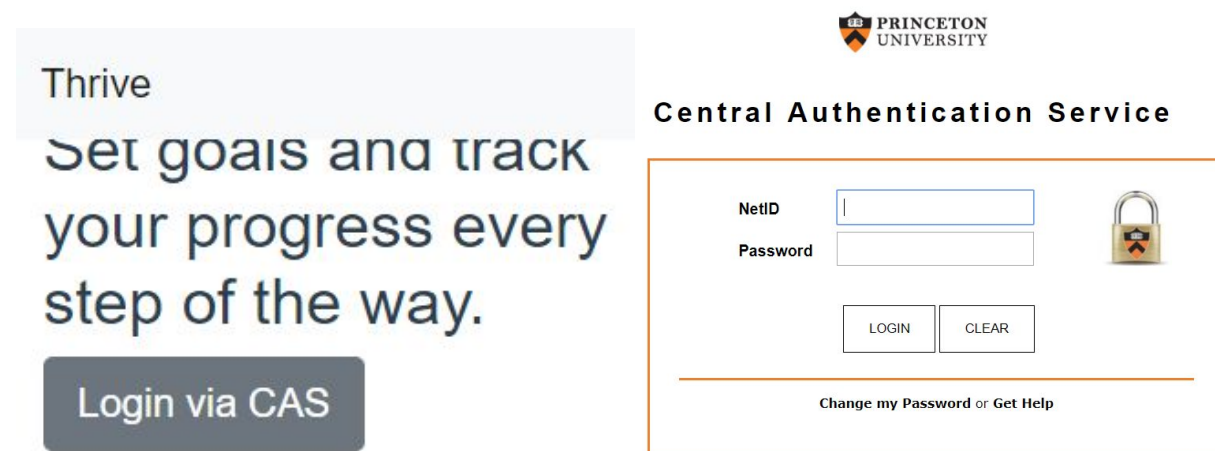
The following were identified as major concerns during user evaluations: moving goals results in unpredictable behaviors(goals moving randomly, progress statuses changing unexpectedly), moving goals while timers are active results in unpredictable behaviors, the interactions between times for goals and subgoals is unclear, hierarchy is poorly represented, there is no means of

⁵ For more details on the implementation of version one, please see the documentation created by the project team in January 2019, available at <https://drive.google.com/drive/folders/19aAJVrmw0kuhdzZ-NFr5X1aC5SIJRR7->

collapsing and uncollapsing goals like there is for Todoist, there is no means of sharing goal templates with students. Prior to attempting to address these concerns, I spoke with an adviser and software engineering specialist, Dr. Robert Dondero, to determine which issues take priority and in which order I should proceed in rectifying the bugs and redesigning app features. Dr. Dondero specified that first I should removing existing bugs, followed by redesigning the user interface to make goal hierarchies more clear and collapsible. He specified that after that I should redesign the interaction between goal times such that there is an invariant that each goal has a time that is the sum of the times of its subgoals. After that, he stated that implementing template sharing features such that students and instructors can share templates would make a good stretch goal for this or future iterations of the app.

Functionality

Upon navigating to the website, the first thing the user will notice on the landing page is a login button on the left-hand side. By clicking on the login button, the user will be prompted to enter their net ID and password in Princeton University Central authentication system.



The image shows two side-by-side screenshots. The left screenshot is the Thrive app landing page, featuring the text "Thrive set goals and track your progress every step of the way." and a "Login via CAS" button. The right screenshot is the Princeton University Central Authentication Service login form, which includes input fields for "NetID" and "Password", "LOGIN" and "CLEAR" buttons, and a link for "Change my Password or Get Help".

Thrive
set goals and track
your progress every
step of the way.
Login via CAS

PRINCETON UNIVERSITY
Central Authentication Service

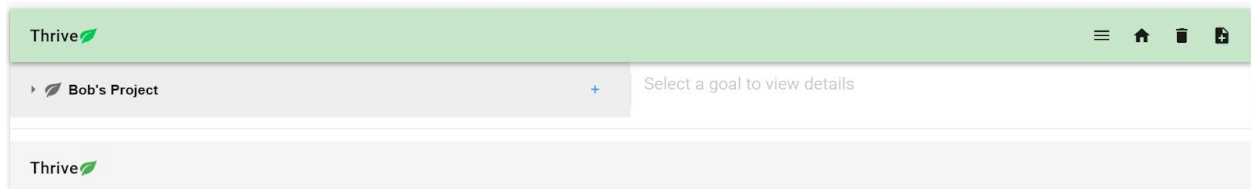
NetID

Password

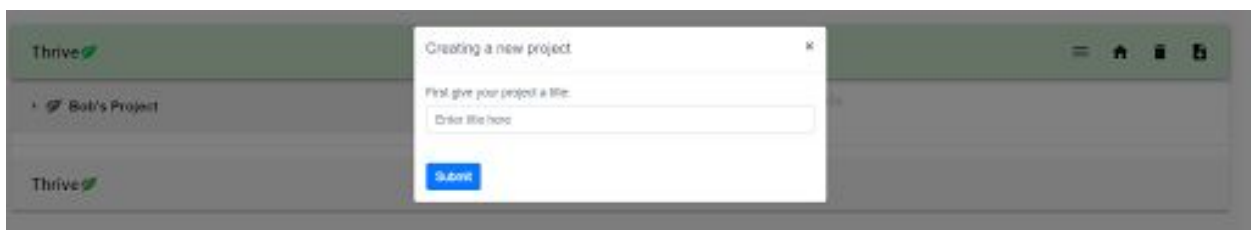
LOGIN CLEAR

[Change my Password or Get Help](#)

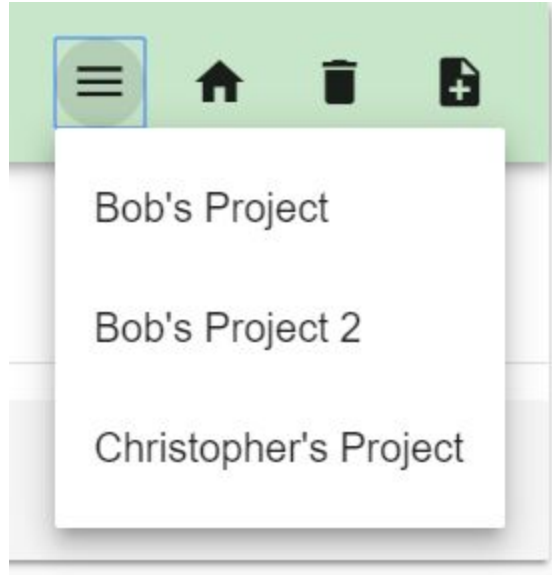
Once the user has logged in, they will be taken to the main Thrive page. On arriving on this page, they will see a toolbar as well as the first project in alphabetical order if they already have existing projects.



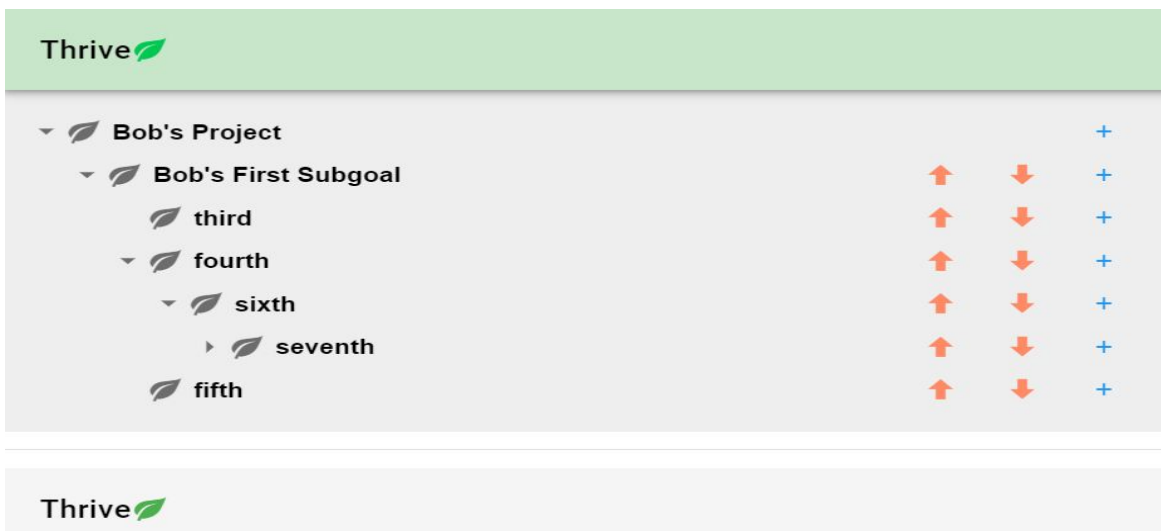
The toolbar supports various functionalities. There is a trash can icon for deleting the current project template. There is a home button which will return the user to the splash landing page(i.e. the log in screen above). There is an add-note icon(directly to the right of the trash can icon) for adding a new project. Upon clicking on this symbol, a modal will appear asking the user to give a title to their new project template.



Most importantly, there's a drop-down menu button which allows the user to select which project template they would like displayed. Clicking on this button causes a drop-down list to appear from which the user is then able to select the template they would like to work on.



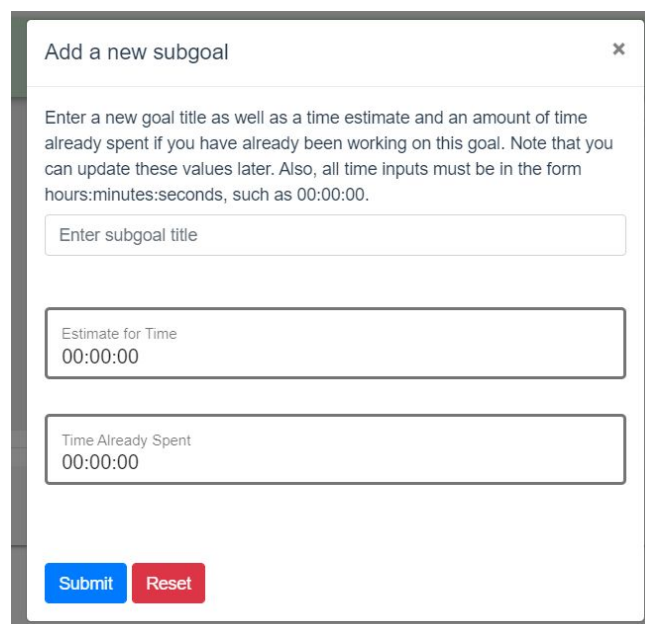
Once a template loads, a goal tree is displayed in the left panel. On opening, the tree is simply a single node with an arrow next to it. By clicking on the arrow to the left of the goal title, the user can expand the tree to see subgoals of the goal, and can expand each goal to see its subgoals respectively. The user may expand or collapse any goals they choose by clicking the triangle icon to the left of the goal title in the left panel.



Additionally, each goal has a red up arrow and a red down arrow. Clicking the up arrow the goal causes the goal to be moved up in the hierarchy underneath its parent goal. Clicking on the down arrow causes a goal to move downward. In the event that the goal is already at the top or the bottom of the hierarchy the up or down arrow respectively will not cause anything to happen.

There is a blue plus sign for each goal which the user can click on to add a subgoal for that goal.

Upon clicking on the blue plus-sign, the “add- subgoal modal” will appear:



The image shows a modal window titled "Add a new subgoal" with a close button (X) in the top right corner. The modal contains the following text: "Enter a new goal title as well as a time estimate and an amount of time already spent if you have already been working on this goal. Note that you can update these values later. Also, all time inputs must be in the form hours:minutes:seconds, such as 00:00:00." Below this text are three input fields: "Enter subgoal title", "Estimate for Time" (with the value "00:00:00"), and "Time Already Spent" (with the value "00:00:00"). At the bottom of the modal are two buttons: "Submit" (blue) and "Reset" (red).

The modal asks the user for three inputs, the first of which is the title of the new goal, the second which is the time estimate for how long it will take to complete the goal, and the third of which is the amount of time already spent working towards the goal. The user is asked to input the

amount of time already spent working towards the goal because adding subgoals to goals which formerly had no subgoals will cause the time for the goal to be reset to 0. Presumably, the amount of time spent on a goal must always be equal to the sum of the amounts of time spent working on its subgoals. Namely, the goal is strictly defined as the sum of its subgoals.

Upon clicking on the name of a given goal, the right-hand panel is changed to display the information for that particular goal.



I will start describing the right-hand panel moving from top to bottom. At the top of the right-hand panel is an input box which contains the current goal title. In the above image, the goal title is “third.” By double-clicking inside this box, the user is allowed to edit the goal title. Upon the input box losing focus, the new submission for the goal title will be sent to the back end and the goal object and corresponding front end representation will be updated accordingly. Beneath the goal title, there are buttons for marking a goal complete and deleting a goal. These do precisely what they say. When a goal is marked complete, the only effect is that its tile becomes green in the left-hand panel. Moreover, there is a text box in which the goal description is displayed. The user is able to edit the goal description simply by clicking inside of this box typing and then clicking off of the box. In other words, upon the box losing focus, a signal is sent to the back end, and the goal is updated to reflect its new description.



seventh

MARK COMPLETE

DELETE GOAL

Goal Description

This is a new description for goal number seven in Bob's Project

Moving further down, there is a series of features for time tracking. The first of which is a time estimate.

Time Management

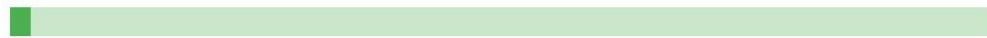
Estimate for Time
00:30:30

Timer

Time Already Spent
00:00:39

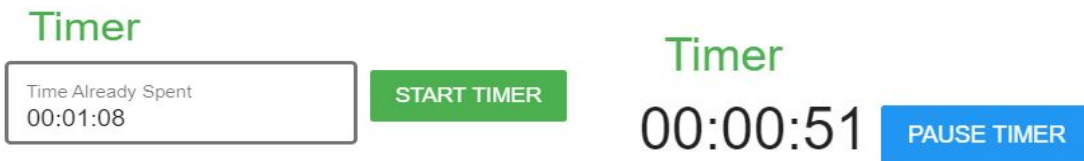
START TIMER

Goal Progress



The time estimate box appears as a simple text input box which contains the current time estimate in a string format. By clicking inside of this box and editing the text, the user is able to update the time estimate for the given goal. Note that their input undergoes string validation first and invalid input will result in no update to the goal.

Beneath this, is a “timer component.” When the goal is actively being worked on i.e. once the timer has been started, the user is not able to edit the input box and instead text is displayed such that the user can see the timer incrementing second-by-second. Once the user pauses the timer, the user is able to directly manually edit the time contained in the timer.



After making edits, once the user clicks outside of the timer input box, a signal will be sent to the back end updating the time for that particular goal.

Note that time estimates and time spent or only editable for leaf nodes. In the event that a goal has child goals, the user is not able to directly edit the time estimate or time for that particular goal:

Time Management

Estimated time needed to complete goal 00:30:30

Time already spent towards achieving goal 00:01:08

Goal Progress



This is because of the fact that there is a tree invariant each parent node must have time estimates and times equivalent to the sums of the times and time estimate of its child nodes. Therefore, in

order to edit the time estimate or time for a parent node, the user must edit the times for its' children nodes.

Finally, each goal contains a progress bar at the bottom of its right hand panel, such that the progress bar reflects the ratio of the time spent so far to the time estimate for the goal. When the time spent is equal to the time estimate the progress bar is completely full and in the event that the time spent surpasses the time estimate there is an error condition and an error message renders directly beneath the progress bar which warns the user that they must update their time estimate since the time spent has surpassed the time estimate.

Design and Implementation

Much of the key infrastructure of the current app was already in place as of the first version which was released in January of 2019. Notably, there have been no major changes to the structure of the database or to the choices of frameworks used. The database is postgres SQL, all server side code is in python, the RESTful API uses Flask, and the front end is html and javascript using the Vue framework.

A postgres SQL database is being used to store Json strings representing “goal objects.” Each goal object essentially represents a goal, it's attributes, and a list of subgoals. In essence, goal objects are a recursive data type, which can be converted back and forth between Json strings and python objects. The database itself merely contains a single table in which each row maps a user ID to a json string. A substantial API was built for interacting with “goal objects” and

performing operations such as swapping goals in a hierarchy or deleting a goal and all of its subgoals. All of the operations defined in the API automatically update the database, reducing the workload necessary for programmers downstream. Many of the features of the underlying representation were essentially abstracted away, such that it is possible to change the underlying representation in the future without having to make any changes to the RESTful API or to the frontend.

Every goal object is essentially a tree. A project template can essentially be thought of as a recursive tree data structure. In essence, the root node contains all other goal nodes. This makes certain operations more inconvenient, however generally most tasks can be accomplished recursively.

The server-side RESTful API contains a variety of functions which use the python object version of goal objects. Each of these functions maps to a corresponding JavaScript function on the front end. notably, the server-side needs to convert each python object into a specialized Json object which also includes dynamically computed features. As an example, times are only stored in leaf nodes. The time values for all of the parent nodes must be computed by summing over the times for their child nodes. Moreover, the vuetify treeview component has very specific requirements for the format of Json strings that it can represent. There's a special mapping function on the server side that converts goal objects into the required format.

Additionally, it is simpler to perform operations such as converting integer times in milliseconds into human legible string time formats on the server side as opposed on the client side.

Accordingly, such computations are performed and the results are stored in the Json strings which are sent to the front end.

Once these Json strings are sent to the front end, they are fed into a vuetify treeview component⁶. The values sent from the backend are then easily accessed as properties of the treeview component.

On the front end, updates are handled by event handlers. In the majority of cases, I've designed the system such that whenever a field loses focus any updates to that field are sent to the back end. This applies for altering goal titles, altering goal descriptions, and altering times. When values are altered, function calls are made to the RESTful API, sending the new values to the back end and causing updates to the underlying goal objects stored in the database. Whenever there is an update, the server side is designed such that it forces the front end to re-render by sending a new Json string and updating the treeview component.

Evaluation: Cognitive Walkthrough

In order to evaluate the app, a cognitive walkthrough was performed done with Dr. Dondero. He was asked to complete the following tasks:

1. Delete the current project
2. Create a new project

⁶ For more information on this component please visit: <https://vuetifyjs.com/en/components/treeview>

3. Create a subgoal
4. Create a second subgoal with time estimate 02:00:00 and actual time 00:40:00
5. Create a second level nested subgoal
6. Mark that second level subgoal complete
7. Swap goals up and down
8. Give one of your goals a detailed description
9. Start the timer for one of your goals
10. Pause the timer for one of your goals
11. Manually change time elapsed for one of your goals
12. Delete a goal
13. Change the title of an existing goal

The instructions were deliberately vague so as to determine whether the user interface was straightforward enough to figure out how to complete the actions.

Dr. Dondero noted multiple concerns with the app. First, on loading a template, the new project goal tree didn't show until he refreshed the page. He remarked that this was confusing.

Moreover, once the goals had loaded, he commented that he didn't know where to click to modify the goal description. He said that the background color for the text input box for the goal description was too similar to the surrounding background colors and moreover that the floating title of "Goal description" and the line directly beneath the text input box made it confusing where he should click to modify the goal description itself. He suggested that in future iterations

of the app the background color for the goal description box be distinct from that of the surroundings and additionally that upon clicking inside of the box the background color change in some way to indicate that the user has focused into the box.

Moreover, he expressed substantial concerns about the timer components, particularly the manual updates. He noted that under some circumstances the manual updates are not persistent and get overwritten by the previous times. Dr. Dondero stated that he would prefer for every keystroke to hit the database and moreover that he would like it to be much more clear when the user is actively modifying the time input box. He said that there is no color change when the user clicks into the box and that this is a major issue in terms of ease of use.

Furthermore, and potentially more significantly, he discovered that since the timer submits time updates when the user clicks outside of the time input box, it is possible to create strange behaviors by clicking on another button that sends a signal to the server side prior to making any other clicks away from the time input box. Ostensibly, this is introducing some kind of race condition in which multiple signals are being sent to the back, roughly simultaneously, and these lead to inconsistent states. In order to alleviate this problem, Dr. Dondero recommended creating a special modal for updating times. Effectively, whenever a user wants to update a time estimate or a time they must click on the time time, thus causing a modal to appear. They will then be prompted to update a text box within the modal, and a signal will be sent to the back end only when they click a submit button. This completely prevents any possibility for there to be race conditions.

Aside from these problems, he noted that the user interface appeared to be much cleaner and more usable than in previous iterations. Furthermore, he noted that the tooltips used for each of the various buttons made the app relatively straightforward to use and easy to understand, with notable exceptions for text input box issues.

Evaluation: Nielsen Heuristics

Additionally, I evaluated the system using Nielsen heuristics. The following ten heuristics were used: visibility of system status, match between system and the real world, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency of use, aesthetic and minimalist design, error handling/assistance, help and documentation.

The system satisfies the first heuristic pertaining the visibility of system status. Since there are never any loading states, system status messages are generally unnecessary. The only times when special statuses are necessary when there are conditions such as the time spent surpassing the time estimate. And this is handled by simple error message.

There's a strong match between the system and the real world. Most notably the use of tooltips for all buttons makes it easy for the user to determine the functionality of various buttons.

Moreover, the choice of icons with relation to the functionality of each button makes the system

intuitive: adding a goal is a plus sign, going to the home page is a house, deleting a goal is a trash can, moving a goal up is an up arrow, moving a goal down is a down arrow. Furthermore, the use of colors typically associated with certain functionalities such as completion(green) and deletion(deletion) makes the app visually intuitive and easy-to-use.

Nevertheless, the app fails to meet certain basic standards for user control and freedom. Most notably, there is no way to undo the deletion of goals or goal templates. However, there is a special modal warning the user when they are about to delete a goal. In general, there are no actions which have a corresponding undo or redo button. This should be improved on future iterations of the app.

The app has good consistency and standards. There is generally no ambiguity as to functionality. all functionalities are made clear through tooltips. There are no cases in which similar appearing buttons, icons, or text boxes are associated with drastically different behaviors.

The app makes appropriate use of error prevention for certain scenarios such as when users want to swap goals with timers currently running. In this case, an error modal appears telling the user to pause the timer prior to swapping the goal, and they are only allowed to swap once the timer has been paused. Nevertheless, there are other error states which only result in an error message. Most notably when the time spent surpasses the time estimate, an error message appears beneath the progress bar, however there is nothing in place to prevent this error state from occurring in the first place. It is somewhat questionable whether it is even possible to prevent this error state. Potentially, a special modal could appear when the time spent first

equals the time estimate. This modal could ask the user whether they would want to mark the goal complete or update their time estimate and could force them to complete one of the actions prior to closing the modal.

Throughout the app, there is no need to remember substantial amounts of information. There are no lengthy dialog boxes or particularly complex processes. Everything is generally short and straightforward.

There are no features in place for flexibility or efficiency of use. There is no way for an advanced user to customize the interface of the app or to speed up certain routine processes. In future iterations of the app, it may be desirable for the user to be able to customize the appearance of the interface, or to reuse project templates they have already created.

The app uses aesthetic and minimalist design for the most part. Generally, the messages and dialog box is a relatively short, with the exception of the message for adding subgoals which informs the user that adding a sub call may cause the time for the apparent goal to be altered. Ostensibly, tooltips may be overused, however they do make the functionality of the app extremely clear and easy to understand.

The app helps users to recognize, diagnose, and recover from certain basic errors such as time spent surpassing time estimate. However, certain types of system error such as failure to connect to the database cause no message to render on the front end. In general, there is no messaging

system when server-side errors occur. In the event of a server-side error, a typical user would have no way of knowing what is going on. This is an aspect that can be improved in future iterations of the app.

There is no help or documentation present for the app. Nevertheless, the app is relatively simple and compact and it is questionable whether it has any features sufficiently complicated as to merit more detailed explanations for the user beyond the tooltips already provided.

Overall, the app represents a major improvement over the previous iteration, however still leaves much to be desired and leaves room for work over the Summer and potentially in future semesters.

Conclusion: Next Steps

The app is going to be tested on the students of the Freshman Scholars Institute this Summer. . Accordingly, it is of the utmost priority that the app be an excellent working order prior to the arrival of the FSI students on campus. They're currently slated to arrive on July 9th. Therefore, the suggestions made during the user evaluation must very quickly be incorporated into the app, and if possible the application must undergo further rounds of user evaluation and redesign.

The most significant next steps all around improving the features which Dr. Dondero pointed out as being problematic in the current iteration of the app. First and foremost, it will be necessary to create a new modal for entering time estimates and spent times. Particularly, this modal must be designed in such a way that there are no major issues with user input verification. In an ideal situation, the modal would be designed in a way such that it is impossible for it to be in an error state. As an example, many time selection features an existing timer apps, such as the one used in the timer on the iPhone, force the user to select values for hours, minutes, and seconds from a drop-down list of integers. This completely eliminates the need for any special forms of string verification. Moreover, the use of the modal completely obviates any possible race conditions, such as those that plague the current iteration.

Once this is completed, slightly less significant aesthetic details can be focused on. Most notably, text input boxes can be modified such that they have darker backgrounds than the surrounding areas, and moreover that they change colors when the user focuses into the text boxes. This will hopefully be a relatively quick fix.

Next, a new overall progress bar will be implemented such that regardless of which goal is currently open in the right panel, the progress bar for the overall project will be visible near the top of the screen directly beneath the toolbar. The overall progress will be based on the time estimates for each subgoal and their completion statuses. Effectively, the estimated time for the completed goals will be divided by the total estimated time for the project overall to compute progress.

After these changes have been made, the app should undergo further user evaluation. Notably, Dr. Dondero had mentioned that he did not find the time tracking component particularly useful, while he nevertheless found the time estimates to be highly useful. One of the key focuses of this set of user evaluations will be to determine whether other users agree. If it is the general consensus that the timers are not particularly helpful, the feature will be removed prior to the start of FSI.

Moreover, at this phase, users, most likely instructors in the computer science department, will be asked about their preferences in the design of separate accounts for students and teachers. In the next phase of requirements gathering, one of the key focuses will be to design a system for teachers to provide templates to students.

Conclusion: Lessons Learned

One of the key lessons this semester concerned debugging. In particular, I discovered that sometimes it is much easier to simply prevent the user from doing something which causes errors as opposed to trying to correct the underlying cause in the code. One of the most difficult bugs involved swapping goals while timers were still active. This led to race conditions, which I failed to eliminate even after a week and a half debugging, trying various techniques involving threading and locking. Ultimately, the solution was simply to not allow the user to swap goals until all timers were paused. It had not even occurred to me that this was an ostensible solution. It was only after Dr. Dondero had mentioned it that I even considered it. Solving the bug in this

way took a matter of minutes, whereas I was completely unsuccessful even after weeks of trying to solve it the other way. Consequently, in the future I will generally use the heuristic that it is often better to simply prevent the user from doing something that causes an error condition by modifying the user interface rather than trying to eliminate the error altogether.

Additionally, I greatly underestimated the difficulties which can arise from package incompatibilities and poor documentation. Simply finding a compatible package which could support the functionality I was looking for in representing hierarchical goals was frankly much more frustrating than the actual process of coding or debugging itself. The process of trial and error with multiple components found on the internet was frustrating. This process could have been avoided if I had simply decided in the beginning to use the vuetify package which I ultimately ended up using. The app already contained many vuetify components from the previous semester, and therefore already had many of the required packages. Had I understood this in starting out the process of looking for a package, it might have been much simpler. Rather than play around with other packages and try to work out their compatibility issues, I should have just simply went for the vuetify version, even if it were slightly less convenient or feature-rich than many of the other components available online. Essentially, the time wasted on dealing with package incompatibilities vastly outweighs any potential gains from having a slightly superior component.