

CSC 400 Software Specification and Design Document

FINAL REPORT

JORDAN PHOENIX

GITHUB LINK: [HTTPS://GITHUB.COM/JPHOEN18/GOALFORIT.GIT](https://github.com/JPHOEN18/GOALFORIT.GIT)

Introduction:

For my CSC 400 Capstone project, clearly there are quite a few changes that were made to the original idea. Ultimately, the original idea pitched had to be discarded due to data constraints. So, the new system that I will be developing is completely different from what was previously being worked on. For my new application idea, essentially the idea is to provide a simple application with a pleasant interface that allows users to plan out and formulate a particular goal. This is done by them defining the goal. Then, the application will take the key words from the goal description and compile an organized list of resources (videos, articles, newspapers, events) that will give a person appropriate background information from which to build a foundation for accomplishing, or at least planning to accomplish, a particular goal.

For example, if someone wanted to learn a language, they would give the new goal a name (i.e: learn French), and then the program would search using a few api's for relevant resources. Essentially, the program will first allow them to create this goal out of resources provided by encouraging them to form their own curriculum or list of what to read and study next. This goal setting will also come with the ability to label goals with due dates and modify the goals or milestones already set. This app will help people to take responsibility for trying to accomplish a goal/task on their own, while also helping them to pull together some helpful resources and build a plan for accomplishing objectives.

Some advantages of this system would be that instead of just displaying a feed of articles and videos for the person to just browse and save for later, as some applications do, the resources displayed will be relevant to the user's main objective. Potential users would be people who perhaps wish to be organized enough to accomplish a specific personal goal, but they lost track quickly of what they were doing. For example, many people attempt to learn a different language, or learn to cook, or learn to code computer programs. Often though, people find so many resources online which leads to them either being overwhelmed with choices, or they just forget articles they were looking at. They probably have 20 browsers open to various, useful articles. That is, until they accidentally close a browser or shut their computer off. So, people sometimes lack the organization skills necessary to propel themselves toward their goals.

In addition, people like to think that they have the self-discipline to accomplish goals independently, but we all still benefit from structure and planning. Therefore, I would also hope that the users of the program would especially be college students, or people who have a desire to learn on their own and formulate milestones and goals.

Ultimately, this idea will benefit the user by providing an interface in which they can set a goal, do some foundational research on the goal, and save these goals with resources to the database. In addition, program would allow us to save resources (links, urls, potentially full articles) to the user's self-made, self-ordered curriculum. Users can then use this list of resources as a small part of a larger goal in their goal timeline that my program will help them create.

Architecture:

Essentially, the architecture of this system is going to be very similar to the architecture of my previous Capstone idea. This will be a client-server system. I will use node.js as the backend language to communicate with the api's and database, storing and fetching needed data. This application is primarily focused on being mobile phone compatible and will probably be hosted using google cloud services for websites. So, the major inputs of the system would be from the user. They will give me their desired goals, and they will construct their own individual goal timeline. Output will include the users resources from the api's, as well as their constructed timeline which will track their goals, how much of their sub-goals are left before they complete the whole goal, and of course their goal profile which is the center hub for their goals and their respective resources. This application will use a web interface, constructed with the classic html, css, and potentially bootstrap for a simpler web app construction.

Use Cases:

For the first scenario, this is when a user has created a new goal. So, the user logs into his account and wishes to create a **brand new goal**. User is then tasked with giving the goal a name, a brief description, and a goal due date. Once this goal is created, the user must enter sub-goals that the original goal will consist of. A user can decide to either enter a stream of continuous sub-goals, or enter a set number of sub-goals. Once the user is done entering the goals, he can select done. Based off a user's goal description, a list of online resources is generated and displayed. A user can then sort through online resource snippets such as web articles, videos, and forum posts. If he sees a URL whose respective article looks promising, he can simply save these articles in priority lists.

Going a little more in depth, the exact process of creating a goal will now be elaborated. So, once a user decides to create a new goal from the main profile screen, he will be sent to the main goal input screen. From that screen, he will be asked to insert the goal type, description(notes or important points for future reference), start date, and end date or due date. In addition, there is a part where the user creating the goal is asked to "break it down." In this section, the user will be asked to divide the goal into sub goals of varying length (if desired). This is essentially a "mini goal." Once the "break it down" button is pressed, the user will be prompted to enter the start date, end date, and number of sub tasks that are needed to complete the goal. Once the overall goal's sub goal structure is specified, the screen will switch to the first sub goal input screen. From this screen the sub goal title, start date, due date, and description are entered in. Once each sub goal is entered, the user will be returned to the main goal selection page. On this page, they will press done and see the resource page, one of many methods they can use to exit the goal-making screen. This resource page is an automatic selection of resources from the internet that are collected and neatly arranged in a central resource "hub." From this resource page, a few different things can be done which will be discussed in the following paragraphs. However, the other option besides going to the resource page is to save the goal and go straight to that goal's main page. However, since no resources were selected, the resource list section of the goal page will be empty. So, I believe it will be more common for users to simply go to the main resource page upon goal creation completion.

So, after every goal is created a resource list will be generated. This resource list's purpose is to give the person some ideas, inspiration, and motivation to accomplish whatever goal they are trying to reach. In addition, it also allows the person using the application to save interesting articles and videos

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

that they would like to refer to later. From the resource page one can make a resource list which they can organize into three main lists. These lists are called High Priority, Medium Priority, and Low Priority. Once the user has simply selected resources to associate with the goal, these resources (web links, articles, video URL) will be saved in a database and displayed when the user enters the goal's main page. From the user profile, the user can scroll down to select a goal. Once a goal is selected, the user will be sent to the single goal page where the resources, goal's sub-goals, and goal timeline is viewable.

For the second scenario, a user will **edit a goal**. So, user will need to navigate to the goal main page, and from there he will click a button at the top of the goal page that says "edit goal." From there, a form will pop up that will have editable fields for each part of a goal. User will edit what he sees fit, and then click save. Once he does that, goal will be edited as desired. In this scenario, I can also include when a user needs to **edit a sub-goal**. Once a user is in the home page for a particular goal, he will click on the goal timeline button which will then display the goal visually as a graph of sub-goals arranged by sub-goal groups (sub-goals that can be worked on together). A user can click on the sub-goal div, which will then route to the individual sub-goal page. On this page, the user can edit the sub-goal with a similar form to what will be used for editing a goal. User can save the edits and return to the individual sub-goal page where they can observe their edits first-hand.

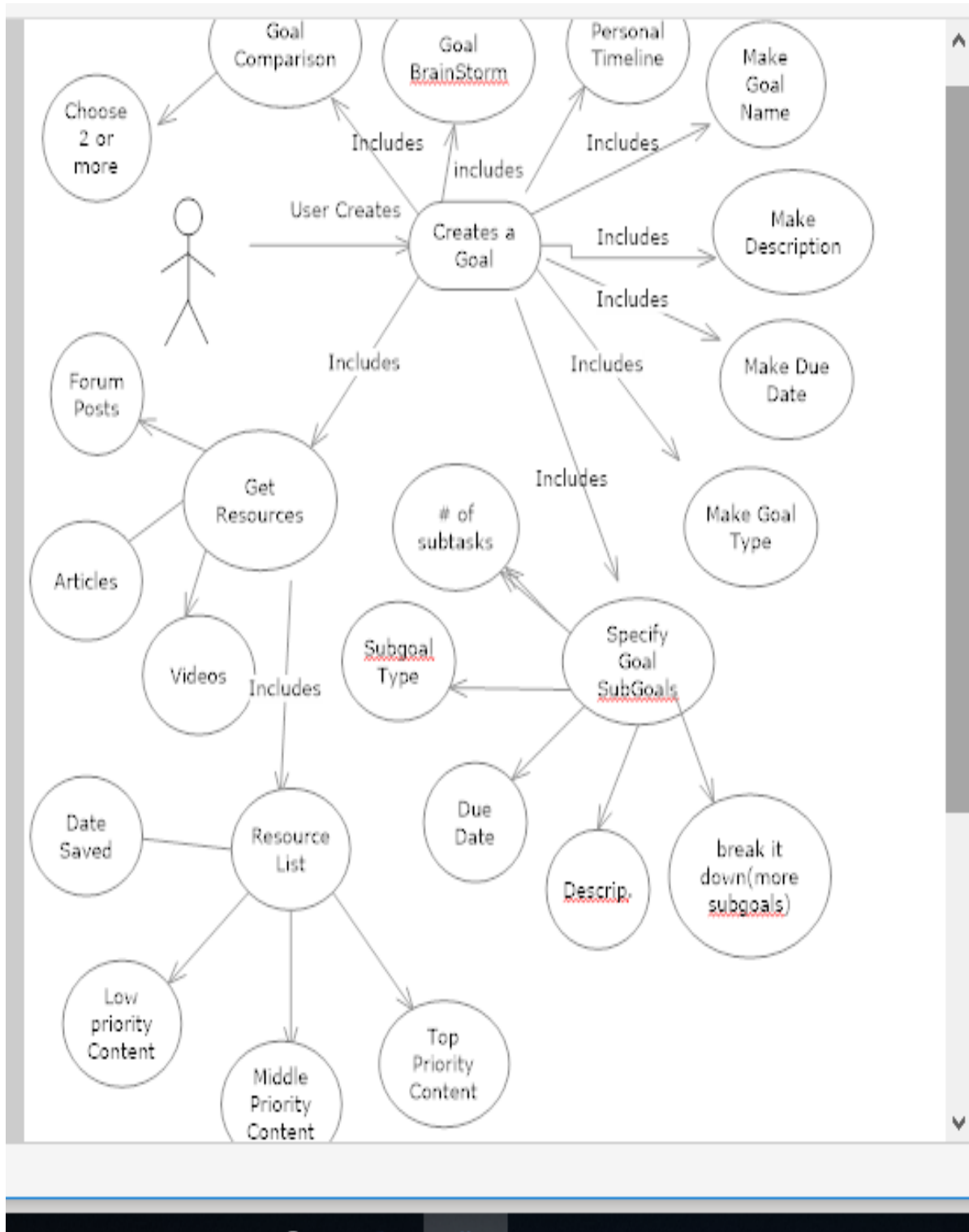
For the third scenario, a user **can delete a goal or sub-goal**. A user will start off in the main goal page, where the goal to be edited is contained. A user will click on the goal, which will route him to the main single goal page. On this page, the user can then select "delete" goal. However, perhaps the user prefers to delete one or more sub-goals from the goal. So, the user can go to the goal timeline, and for each sub-goal to be deleted user just needs to go to that sub-goal's page and select delete. Then, they will see the updated timeline in the goal timeline page.

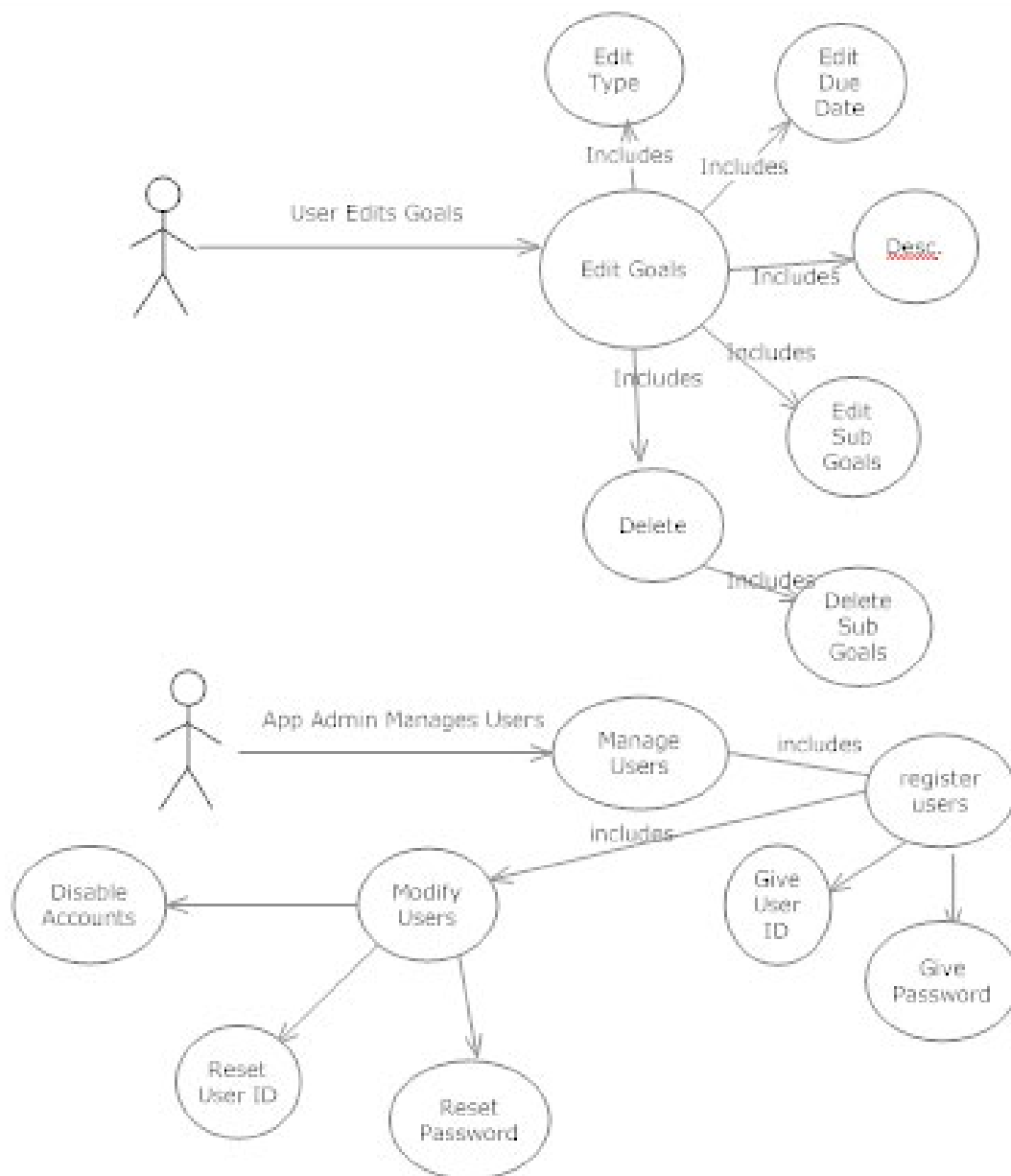
A fourth scenario would be specific to a sub-goal. For each sub-goal, the application can **create a to-do list of needed actions** that the user must compete in order to mark the sub-goal as being 100 percent complete. Therefore, from the user profile, the user can select a goal. Once in the goal home page, the user can then click on the goal timeline. From the goal timeline, the user can select a sub-goal and enter the sub-goal page. From the sub-goal page, the user can then select the to-do list. In this page, the user can define a list of tasks that he feels will be necessary to complete the sub-goal. For each task that is checked, the sub-goal's home page will update by showing the percentage completed bar as incremented (ex.: 70%, 90%, etc.).

A fifth and final scenario that will be included would be when a user needs to **mark a resource as being read or not read**. A user can do this using either one of two methods. First, the user can select resources from the user home page. Once logged in, the user can select the main resources tab. Once selected, user will need to select a resource from the array of resources. Once selected, there is an option to mark a resource as read. Once user marks the resource, the next time he revisits the resource, there will be a statement in <p> tags indicating that the user has read or viewed this particular resource (it will be highlighted in green text color).

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

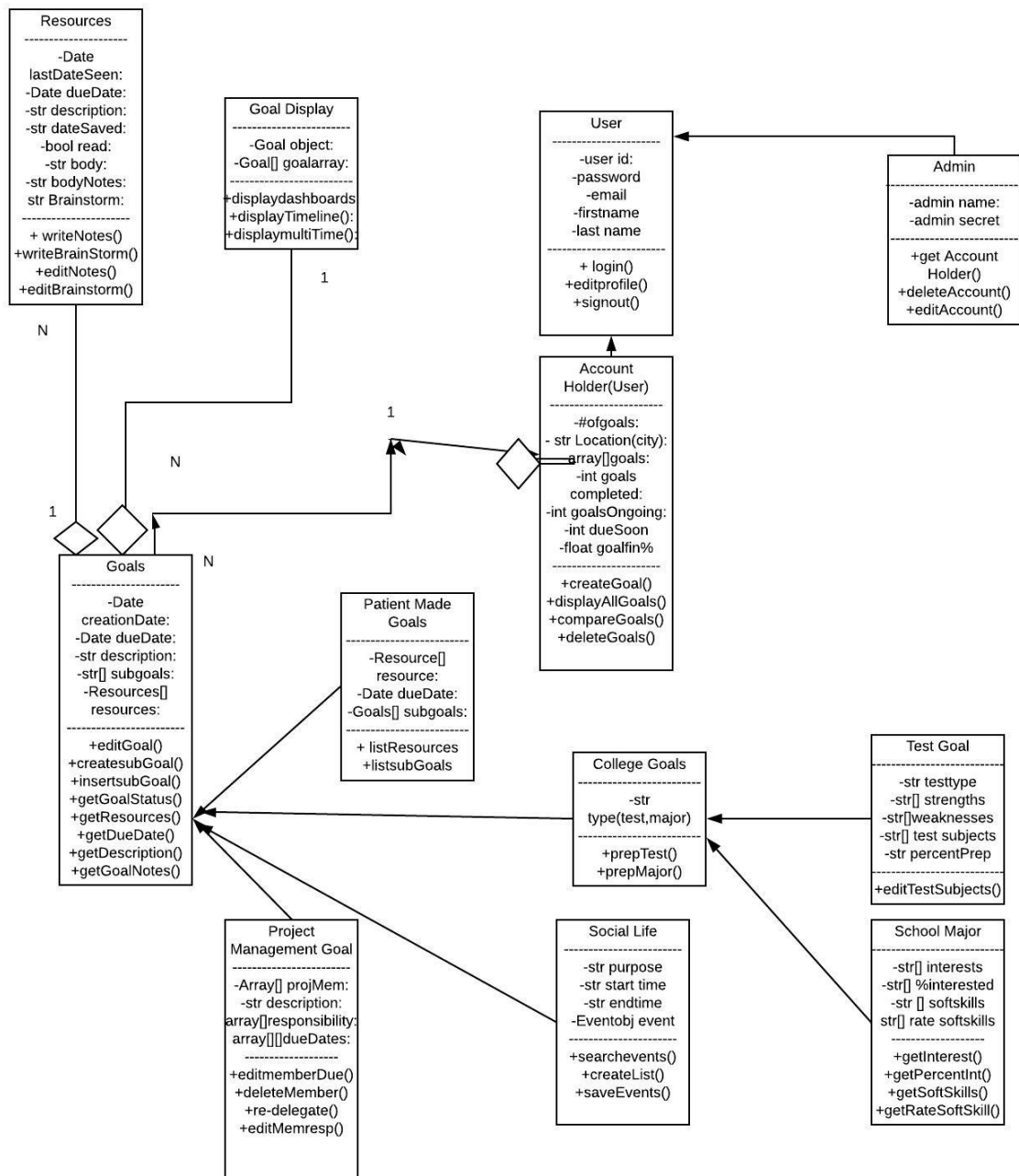
Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18
Case Diagrams:





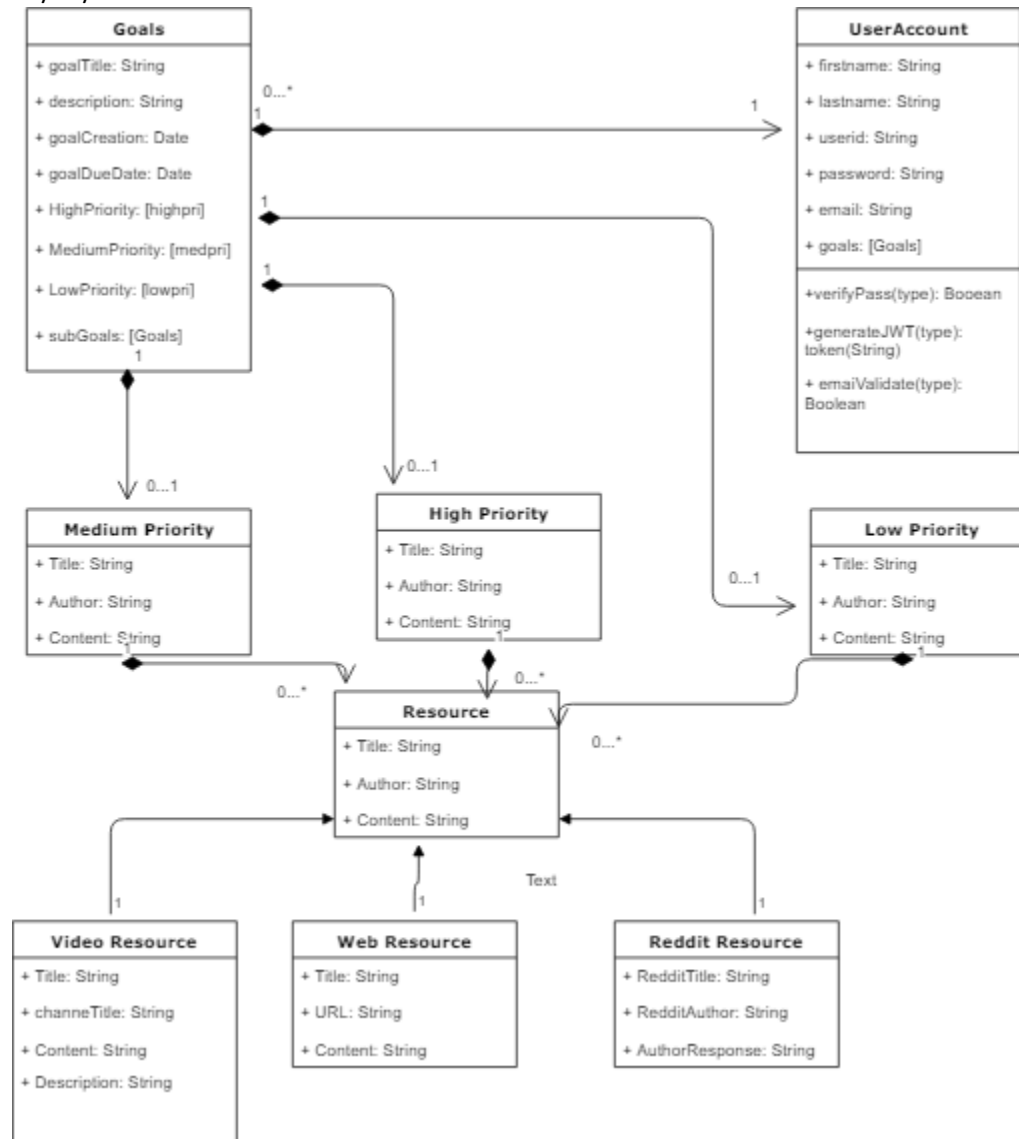
Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

5.) Structure Diagram (UML)



5b.) Structure Diagram – UML (Final):

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18



6.) Schema:

For this Capstone project, I decided a long time ago to go with a NoSQL database since the data I am storing is not extremely heavy duty, complex, or overly dependent on other data. Therefore, I saw no need to use a relational database. I am going with MongoDB, and through my research it appears that I will be using a one-to-many modeling approach. This seems ideal for my project, since I have a few nested pieces of data where I need to get to the child node, and this can be a challenge for other modeling approaches according to MongoDB documentation. So, in short this assures that when I am querying for a particular goal, if I have the ID or name(related to a particular goal for a particular user), I can search for that goal and it's associated parameters with relative ease.

Here is an example of what I am talking about:

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

```
{
  Name: "Jordan Phoenix
  Goalnumber: 32
  Location: New Haven, CT
  goalsCompleted: #
  goals: [
    GoalID(1),
    GoalID(2),
    GoalID(3),
    GoalID(4),
    .....] ..... }
{ _id: GoalID(1)
  Name: 'Learn Chinese in 1 month'
  Description: " ....."
  Subgoals: [ (), (), (), (), ] }
```

So, it appears that once I locate the user in the database, the respective goals, fields, and sub goals will be relatively easier to query and display as I see fit.

Above was a sample of what my database would possibly look like. However, as the application planning and database design progressed, the actual structure needed began to take form. Currently, this is what the database schema looks like (these are two separate examples of utilized schemas, although there are a few others. These two below are the most complicated of the schemas). :

Ex. 1 – User Account Schema

```
UserAccount = new Schema({
  firstname: {type: String},
  lastname: {type: String },
  userid: { type: String },
  password: { type: String },
  email: { type: String },
  goals: { type: [GoalsSchema] },
```

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

```
    admin: {type: Boolean},
    saltSecret: { type: String}
  });

  UserAccount.path('email').validate((val) => {

    let emailRegex = /^[^<>()\[\]\\\.,;:\s@"]+(\.[^<>()\[\]\\\.,;:\s@"]*)|(".*")@((\[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|((\[a-zA-Z-0-9]+\.[a-zA-Z]{2,}))$)/;

    return emailRegex.test(val);}, 'Invalid e-mail.');
```



```
  UserAccount.pre('save', function(next){

    console.log("I am about to encrypt");

    bcrypt.genSalt(10, (err,salt) =>{

      bcrypt.hash(this.password, salt, (err, hash) =>{

        this.password = hash;

        this.saltSecret = salt;

        next();

      })

    })

  });

  UserAccount.methods.verifyPassword = function (password) {

    return bcrypt.compareSync(password, this.password);

  };

  UserAccount.methods.generateJwt = function(){

    return jwt.sign({ _id: this._id},

      JWT_SECRET,

      {expiresIn: JWT_EXP}

    );

  };

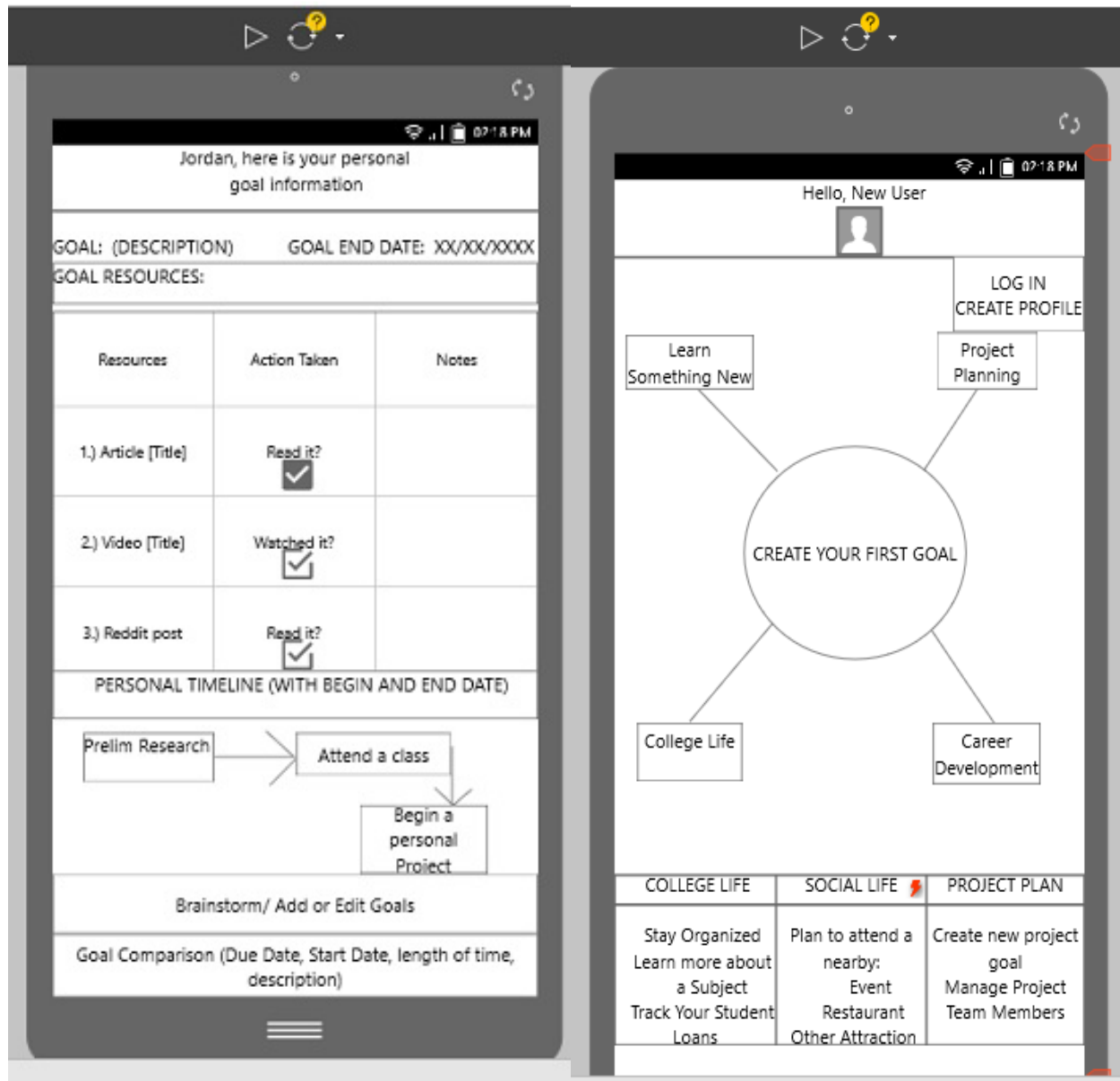
  export default mongoose.model('UserAccount', UserAccount, 'UserAccount');
```

Ex.: 2 – User Goal Schema

```
Goals = new Schema({  
  goalTitle: { type: String },  
  description: { type: String },  
  goalCreationDate: { type: Date },  
  goalDueDate: { type: Date },  
  HighPriority: { type: [HighPriorityList.schema] },  
  MediumPriority: { type: [MediumPriorityList.schema] },  
  LowPriority: { type: [LowPriorityList.schema] }  
});  
var GoalSchema = mongoose.model('GoalSchema', Goals);  
module.exports = GoalSchema;
```

7.) Mockup Designs(The program I used was the free version. Many features not available such as colors/hues. I plan to upgrade to higher version to flesh it out before implementing.)

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18



Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

The image displays two side-by-side mobile application screens. The left screen is titled 'CREATE PROFILE NEW USERS' and features five text input fields labeled 'FIRST NAME', 'LAST NAME', 'USER ID:', 'USER PASSWORD', and 'USER EMAIL'. At the bottom are 'CANCEL' and 'OK' buttons. The right screen is titled 'LOG IN' and includes two role selection buttons: 'USER' and 'ADMIN'. Below these are input fields for 'User Name:' and 'Pass Word:'. At the bottom are 'Cancel', 'Log In', and 'Reset Login' buttons. Both screens have a dark header bar with a play button and a green checkmark icon. The right screen also shows a status bar at the top with signal, battery, and time (02:18 PM) indicators.

CREATE PROFILE
NEW USERS

FIRST NAME

LAST NAME

USER ID:

USER PASSWORD

USER EMAIL

CANCEL OK

LOG IN

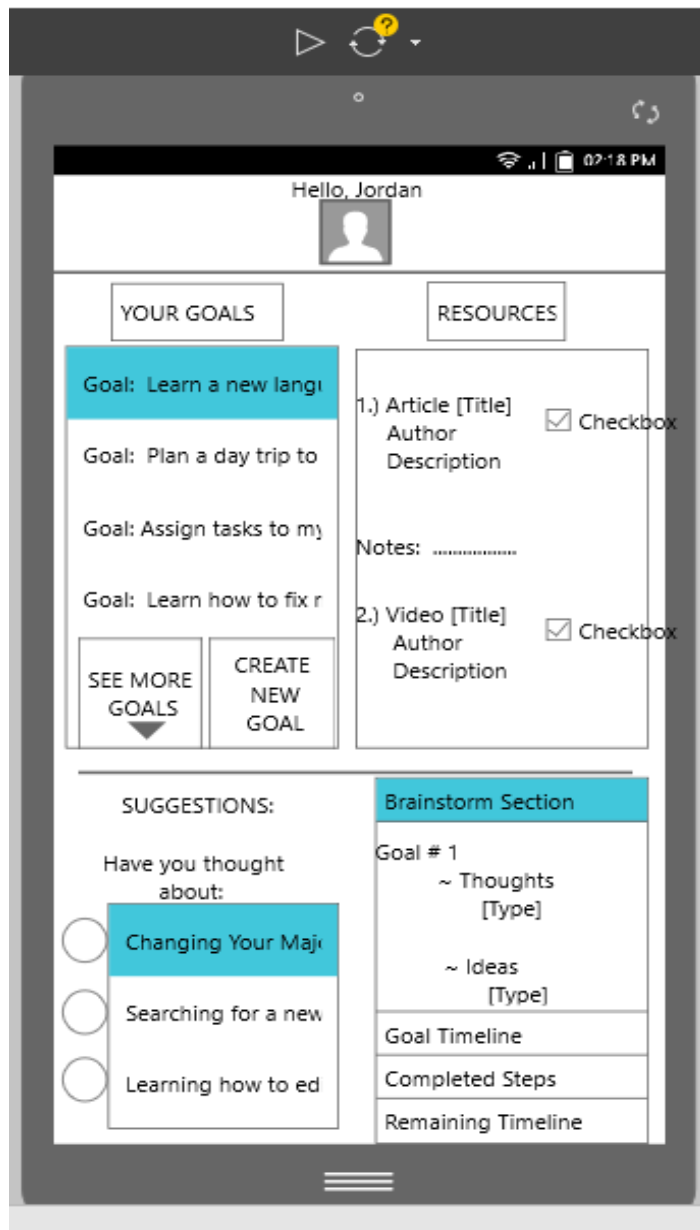
USER ADMIN

User Name:

Pass Word:


Cancel Log In Reset Login

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18



Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

Carrier 1:20 PM 100%


Hello, Jordan

What is Your Goal? >

Type Goal Title

Goal End Date: >

(Date)

Goal Type: >

Select From Picker:

-	:	-
-	:	-

Goal Specifics: >


(Provide More Information)

Break it Down: >

Sub Tasks

Goals Save & Res Pg Save Goal Save & Goal Pg Get Last Goal

Carrier 1:20 PM 100%


Hello, Jordan

Break it Down

Estimated Number of Sub Tasks >

Enter: _____

Estimated Start Date >

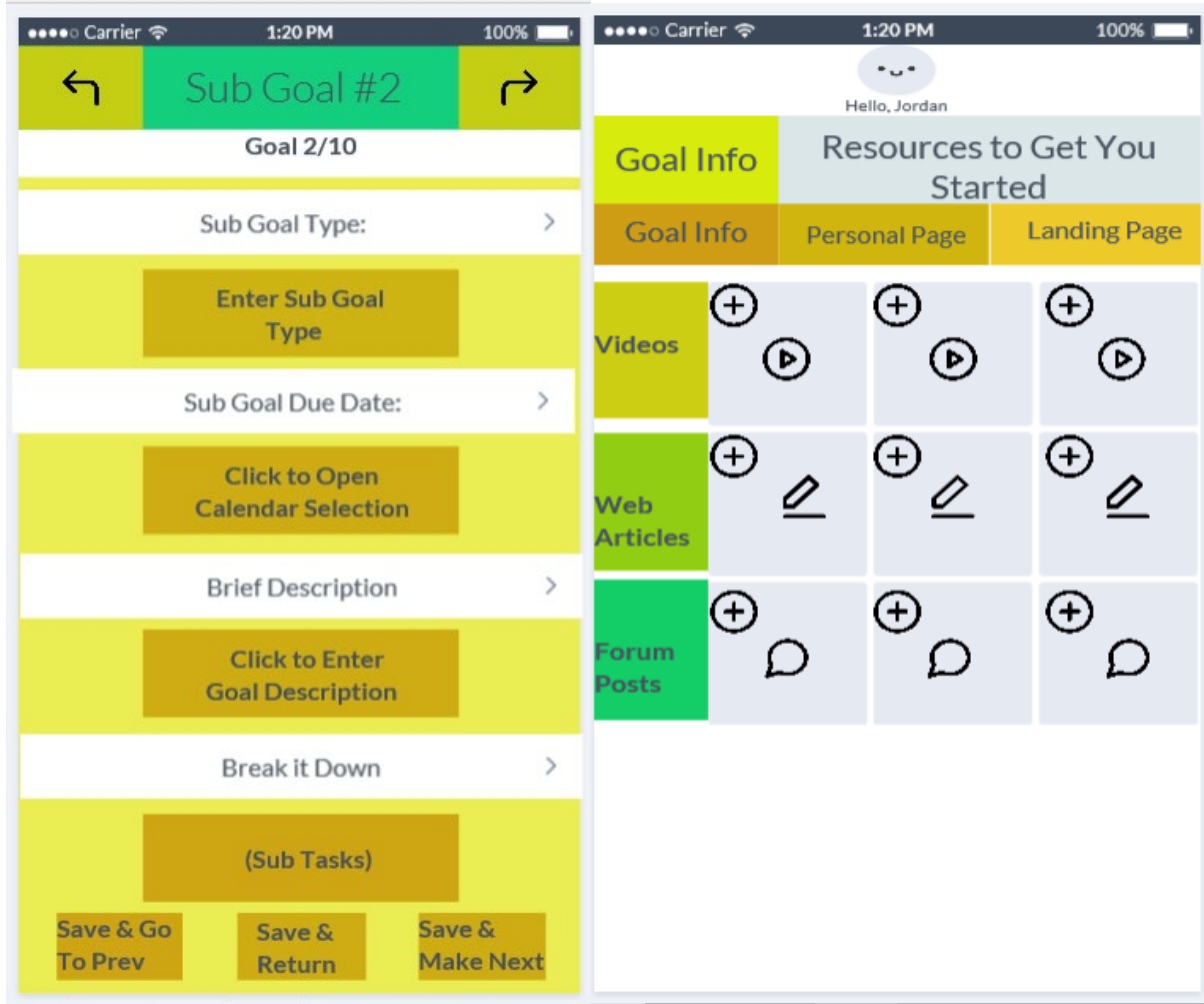
Click Here to Enter Date Below

Sun	-	0
Mon	1	0
Tue	2	0
Wed	3	0
Thu	4	0
Fri	5	0
Sat	6	0

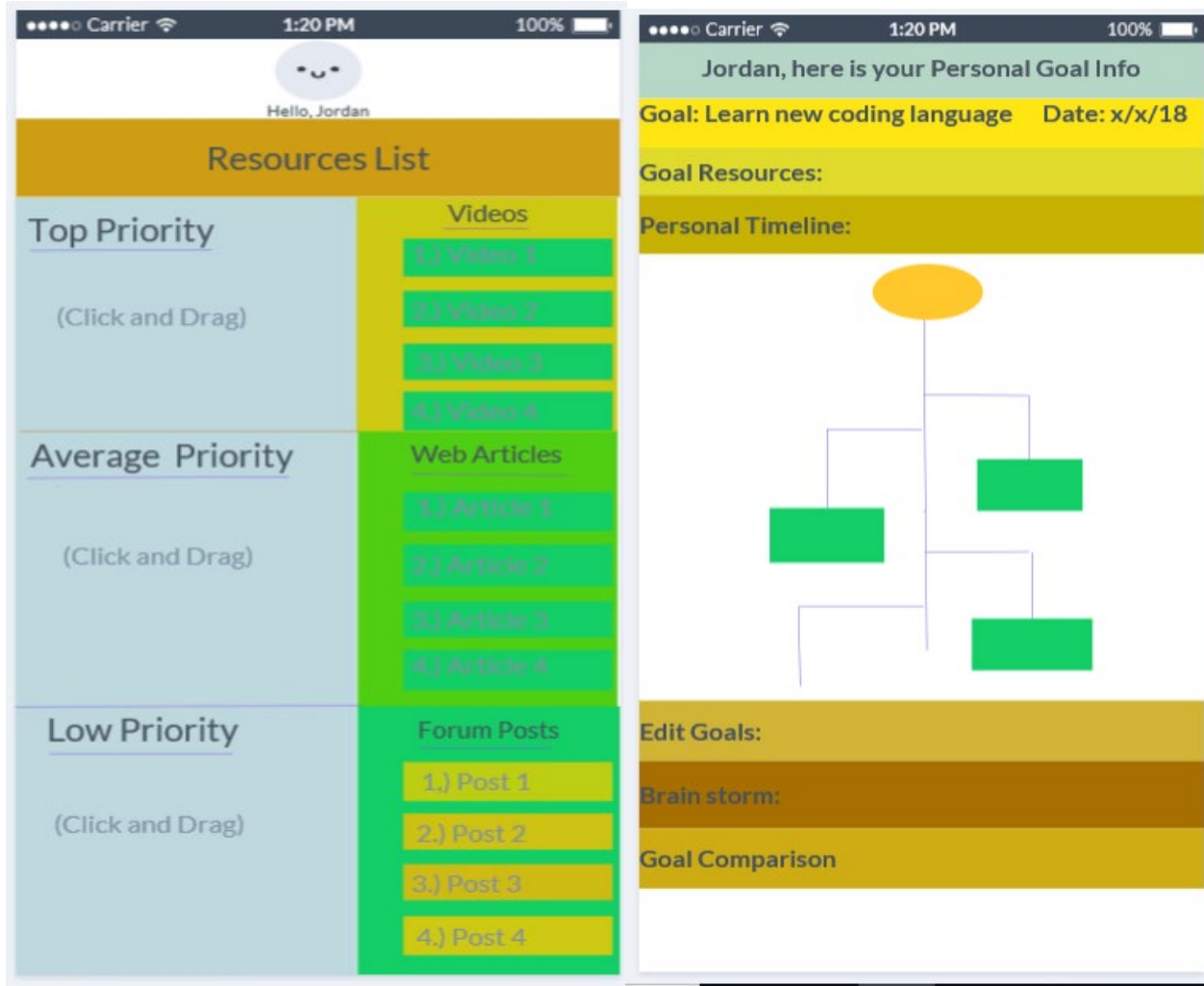
Estimated End Date >

Click Here to Enter Date Below

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18



Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18



Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

Carrier 1:20 PM 100%

Jordan, here is your Personal Goal Info

Goal: Learn new coding language Date: x/x/18

Goal Resources:

Personal Timeline:

Headline
Edit Goals:

End Goal	Sub Goal 2	Mini Goal 3
Sub Goal 5	Mini Goal 1	Part 1
Sub Goal 4	Mini Goal 2	Part 2
Sub Goal 3	Mini Goal 3	Part 3
Sub Goal 2		
Sub Goal 1		

Endgoal — Subgoal — Minigoal 3 — Part 2:

Type: Meeting Goal Sub-components:

Due Date: xx/xx/2018 ☒ Y ☐ N

Description:

Brain storm:

Goal Comparison

Carrier 1:20 PM 100%

Jordan, here is your Personal Goal Info

Goal: Learn new coding language Date: x/x/18

Goal Resources:

Personal Timeline:

Edit Goals:

Brain storm:

Goal Comparison

Select 2+ Goals	Goal 4	Goal 2
Sub Goal 5	Goal Type:	Goal Type:
Sub Goal 4	<input type="text"/>	<input type="text"/>
Sub Goal 3	End Date:	End Date:
Sub Goal 2	<input type="text"/>	<input type="text"/>
	Description:	Description:
	<input type="text"/>	<input type="text"/>
	Start Date:	Start Date:
	<input type="text"/>	<input type="text"/>

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

Carrier 1:20 PM 100%

Jordan, here is your Personal Goal Info

Goal: Learn new coding language Date: x/x/18

Goal Resources:

Personal Timeline:

Edit Goals:

Brain storm: A

Thoughts:

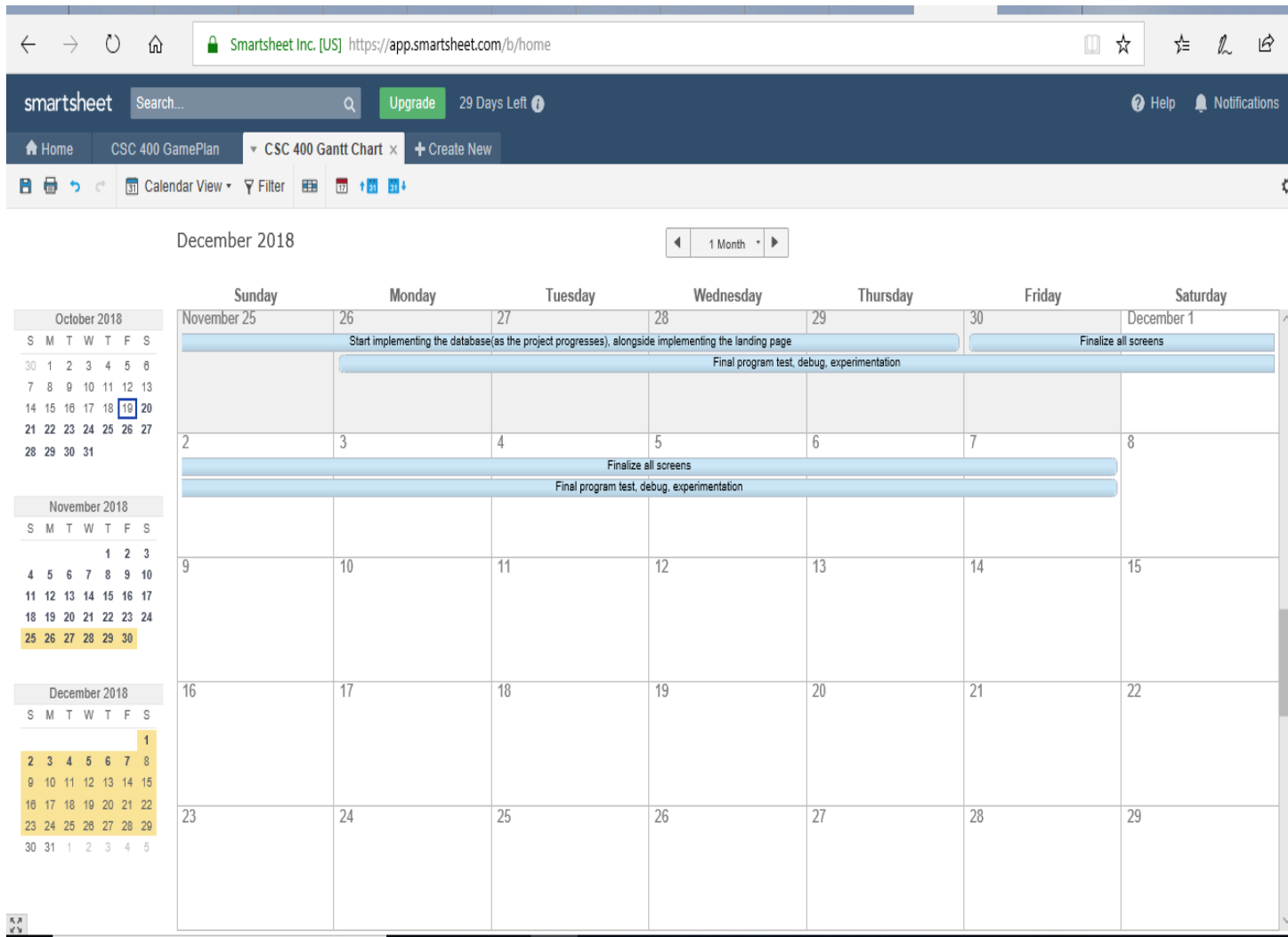
Date: _____
Notes:

Date: _____
Notes:

Date: _____
Notes:

Goal Comparison

7.) Gantt Chart



Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

October - November 2018

◀ 1 Month ▶

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
October 2018	October 21	22	23	24	25	26	27
M T W T F S							
1 2 3 4 5 6							
8 9 10 11 12 13							
15 16 17 18 19 20							
22 23 24 25 26 27							
29 30 31							
November 2018							
M T W T F S							
1 2 3							
5 6 7 8 9 10							
12 13 14 15 16 17							
19 20 21 22 23 24							
26 27 28 29 30							
December 2018							
M T W T F S							
1							
3 4 5 6 7 8							
10 11 12 13 14 15							
17 18 19 20 21 22							
24 25 26 27 28 29							
31 1 2 3 4 5							

October 21	22	23	24	25	26	27	
Give an ultimate finalization to the UI mockup plan and the UML				Start implementing the database(as the project progresses), along			
Finalize my most useful API for project test again, thoroughly read documentation to make sure I can get what I need from them							
				Start coding landing page, using Angular, bootstrap, and other tools.			
28	29	30	31	November 1	2	3	
Start implementing the database(as the project progresses), alongside implementing the landing page							
				Start coding login page			
Start coding landing page, using Angular, bootstrap, and other tools.				Start coding Goal overview page			
				Start coding dashboard page			
4	5	6	7	8	9	10	
Start implementing the database(as the project progresses), alongside implementing the landing page							
Start coding login page				Start coding goal display page			
				Start coding Goal overview page			
Start coding dashboard page							
11	12	13	14	15	16	17	
Start implementing the database(as the project progresses), alongside implementing the landing page							
Start coding goal display page							
Start coding Goal overview page							
18	19	20	21	22	23	24	
Start implementing the database(as the project progresses), alongside implementing the landing page							

Implementation Details

This application's final implementation details can be broken down into roughly five different sections. **Section one would consist primarily of the components needed for creating a goal**, mainly the forms and their corresponding html and java script components. Java script logic needed for the goal form handling was one of the most difficult algorithms to write. This is due to the fact that there were many components that not only needed to work together simultaneously (parent and child components), but also communicate to each other in order to pass data needed so the application could retrieve the information the user entered for their goals. This required a great deal of research into the various (somewhat endless) variety of ways that a component can communicate to another, as well as the many ways parent components can communicate to child components that are being displayed within the parent component's scope.

Section two of the implementation details would be **the user/password authentication logic**. This part of the implementation, being crucial for a client-server web application, was thankfully made much simpler with the jwthelper class that works with the jsonwebtoken. However, this part was far from simple in the end, requiring a great deal of code additions and deletions, trial and error, and a great deal of tutorial and forum research.

Section three would probably have to do with the code relating to **editing goals and sub-goals**. In a relational database, this would hardly be a major problem since every single element of a user's goal would be easily accessed through some sort of SQL query in the code. However, MongoDB's json-like storage although deceptively simple, requires a great deal of work when it comes to querying or editing data that is contained within a nested array. This task is made harder when, like with this application, there is an object within an array within an object whose field needs to be edited. Therefore, whenever a sub-goal or goal needs to be altered or edited, the goal or sub-goal needed to be entirely restructured in the component.ts java script files and then resubmitted to the goal schema within the User Account schema. Simply put, I cannot always target a specific field within an object that lies within an embedded array of arrays, which is what my application's database essentially is.

Section four would be the **goal timeline construction**. For this part of the application, the logic is contained within a single component that retrieves the data for a single goal's sub-goals. Displaying the sub-goals into sub-groups that are color-coordinated based off their due dates was a little tricky but ultimately not the hardest part of the application's implementation.

Finally, the **saving of the goals** from the resourcelist.component.ts code was a little tricky as well. This required a more in-depth knowledge of java script promises, callbacks, and angular services, all of which were very new to me. For example, once a goal is saved, the code then needs to effectively delete all resources from the MongoDB collection(s) that I use to temporarily store resources before the resources are selected for a goal. This requires angular

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

service calls that need to be nested within each other so that I can assure that only after a collection is deleted that the next collection is deleted. This concept of calling a service or function and waiting for the result before running the next needed code is something that was very different, but I do feel that I mastered the use of java script callbacks and promises.

Screen Shots of the Program



Figure A.: User Home Page

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

Goal For It

Logout

Create Your Goal

What Is Your Goal(Enter Below):

Describe Your Goal A Little Better:

Date of Goal Creation :

2019 08 22

Date Picker

You must enter a goal start date please

Date of Goal Creation :

2019 08 22

Date Picker

You must enter a goal due date please

Continue Create Subgoals

Figure B.: Goal Creation

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18



Figure C.: User Login Screen

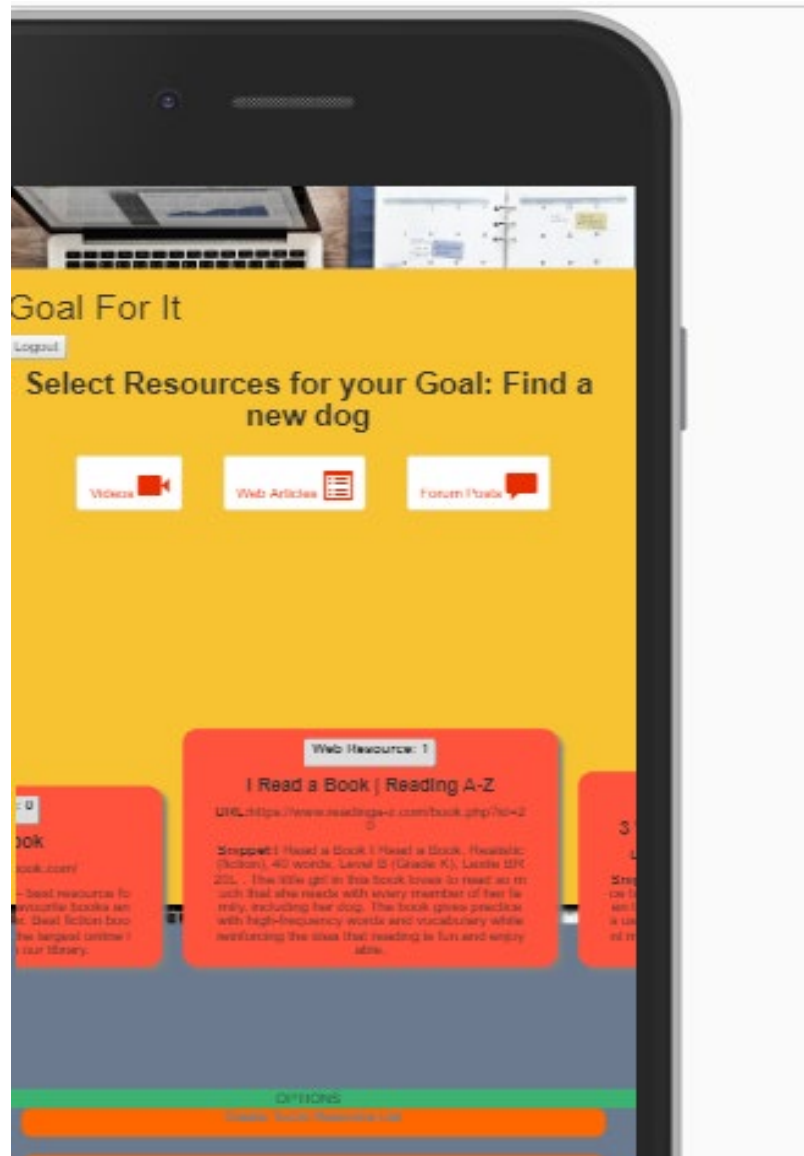


Figure D.: Resource Page

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

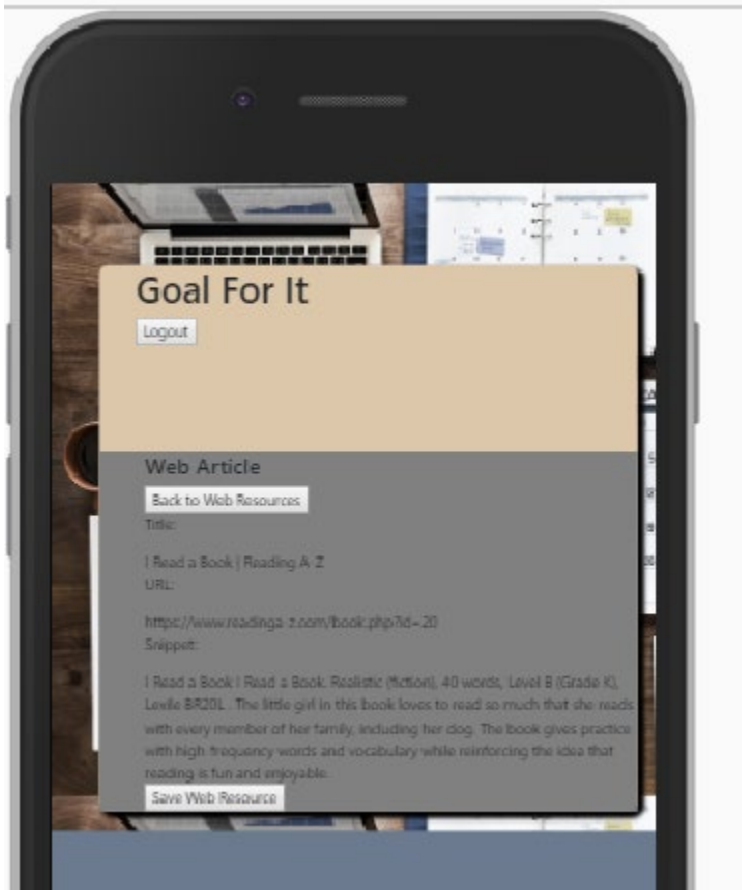


Figure E: Resource Selection

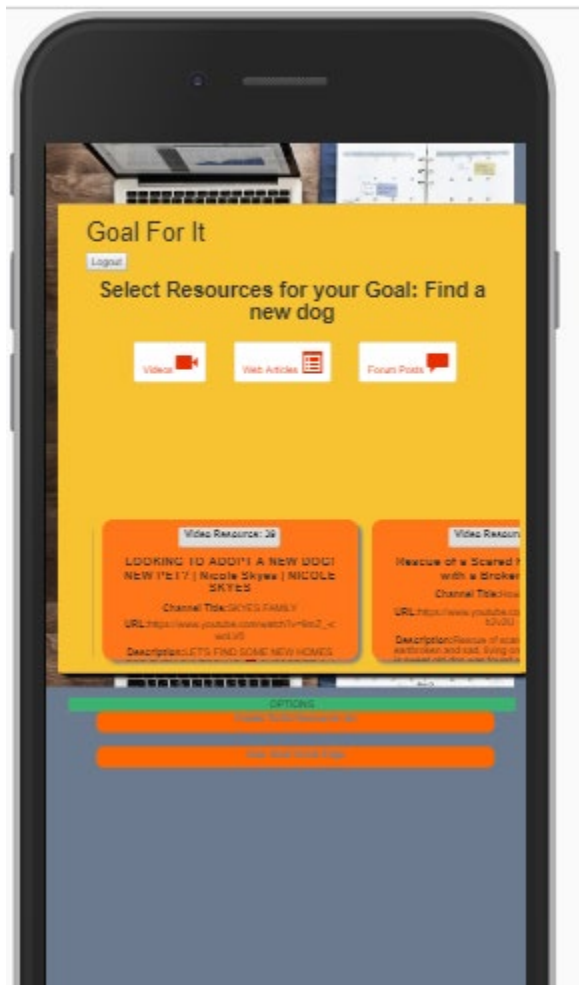


Figure F.: Further Resource Selection

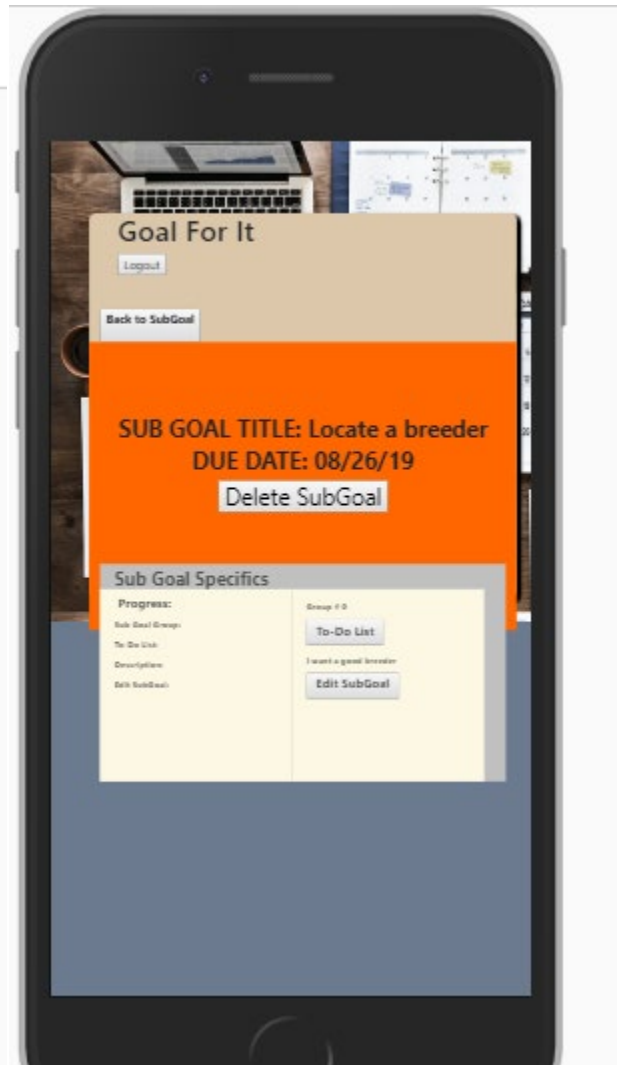


Figure G.: Sub-Goal Page

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

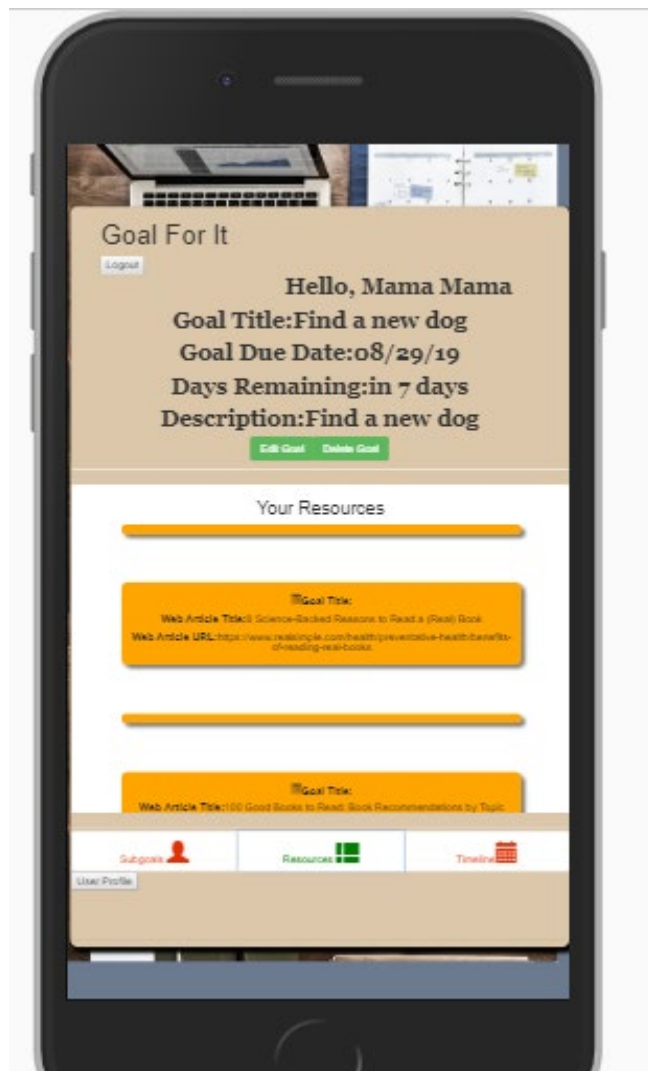


Figure H.: Goal Home Page

State of the Implementation:

Currently, the status of the implementation is complete for the time being. This application's primary objective, which is to create goals with resources, is complete. Of course, every CRUD application requires logic to edit what is created, and users are also able to edit their goals and sub-goals with ease. However, when it comes to editing and deleting content, I would say this is 90% complete since the users are not able to edit or delete resources. All the user can do is check or uncheck if the resource was read or not. In addition, editing sub-goals presented a slight challenge in the end. This is because the sub-goal to do list has a glitch that causes some occasional unexpected behavior when the list is saved or edited. This glitch presents itself in the display of somewhat jumbled, random text. However, when you navigate out of the sub-goal main page and navigate back to the sub-goal page in question the glitch is gone, and the edited information displays perfectly. After this report is submitted, I intend to fix this glitch and continue to refine other aspects of this project.

Another main implementation objective was to create a goal timeline display, which is also 100% implemented and functional. I would like to refine it to be more attractive and interactive in the future. However, the overall purpose of the goal timeline was to be a clear visual reminder of the status of the goal in question, which it turned out to be.

Deleting goals and sub-goals would be another important objective of a CRUD application, which has been met successfully. Once a user deletes a sub-goal, it is effectively erased. Similar results occur for the goals once they are deleted. They are effectively deleted and this is updated in the user home profile goal section. Finally, as mentioned earlier the resources are able to be marked as read/viewed or not read/viewed, which was a late feature to the application but nevertheless works very well.

Testing and Evaluation

Testing this application was a fairly long process, since I naturally tested each feature as it was worked on and refined. Since the application centers primarily around creating a user and then creating a goal, I mainly tested the application based off the successful completion of these tasks. For example, the last test that I did was regarding user creation and goal creation. First, I created a new user. Once that was done, I signed into the application and created a specific goal with eight sub-goals. Each sub-goal was assigned to different sub-goal sub-groups, to test if the goal timeline would display it correctly. This goal was saved with resources, and from there I checked if I could render a timeline, update sub-goals, and delete sub-goals. I would also check if I could edit each sub-goal's to-do list and I would exit the sub-goal page and re-enter the same sub-goal page to see if the information was saved. I repeated this testing

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

process many times throughout the application implementation and also once the application was deemed complete.

In hindsight, I certainly need to more thoroughly test certain aspects of the application, especially the code having to do with the angular services and the http calls they make to the server. Often, whenever user information is retrieved or when resources are retrieved, it seems that the angular service subscriptions return a value multiple times over, slowing down execution time. Other times, it just takes a while to fetch information needed from the backend for no apparent reason. I need to thoroughly investigate these aspects of the code that is making the application runtime occasionally slow.

However, testing and evaluation of my system convince me that I have met my objectives. In the end, users can accomplish what the application intended for them to accomplish with relative ease. As mentioned above, the system can at times run very slow due to multiple backend and api calls, which can be frustrating. In addition, the user interface is not entirely cellphone-use friendly in certain parts. However, these are all things that can be improved. Regardless, overall application objectives were met successfully.

Lessons learned and reflection

I learned very valuable lessons throughout this programming project, which will greatly impact my future coding projects and will hopefully aid me in avoiding certain pitfalls in the future. Foremost among these lessons is the fact that I need to plan my application as thoroughly as possible. This means not only making sure that the overall concept is viable but ensuring that the user interface is planned out and has logical, effective design. Of course, planning thoroughly can also be applied to the actual logic of the application. For example, I should really have structured my app so that I had separate angular services for each group of components that had to work together to achieve a certain objective. In addition, I could have planned out the exact amount of html pages or components that I needed in order to make an application that was an effective, simplistic single page application. Although I did not plan the app as thoroughly as I could have in that area, I did come close.

Another lesson would be related to the final look of the user interface. This would be that I need to construct the user interface with the user in mind. Next time I create a web application, I will ask others to test the app out and see what they feel works and what does not work.

Finally, this coding project has impressed on me the need to make sure that I have all the desired application features firmly in mind. This way, when it is time for implementation, I am not wasting time trying to figure out what other interesting features could be added at last minute. Those features will have already been vetted and determined in advance.

Final Thoughts on CSC 400 Capstone

Upon embarking on this CSC 400 Capstone project, I knew that this project would require a great deal of learning on one's own. After all, I chose a project that would force me to become comfortable with technologies and concepts that I was not strong with before. Thankfully, many of the technologies, frameworks and languages that I was initially not familiar with are currently tools that I know well enough to apply effectively to my next project. However, reflecting on the learning process required to complete this project, I realize that I followed a simple formula for developing the skills and knowledge that I needed. While creating the original design specification, I created a well-defined list of core application features that would be included in the final product. Having these features already determined allowed me the ability to better discern what technologies would be best to use for the project, and what skills, knowledge, api's, or frameworks would be the best fit for the project. So basically, I would look at the core features and then ask myself "What languages and frameworks would be best to implement this application and these features? What related skills are required? Since I need to become familiar with things that are fairly new to me, what tutorials, blogs, books, and videos are there that explain these needed frameworks and concepts?" This is the general method I followed in order to learn as I progressed throughout the application.

Since I opted to go with a mean stack application, it was obvious that I needed to familiarize myself with the four main components of the mean stack application. I needed to become better acquainted with the Mongo database, Node, Express, and java script. Before I even touch on those individual technologies, it is important to mention that I first needed to understand the angular application framework. Learning Angular was a pretty big learning curve. I have never used a web application framework like React or Angular before, so the concept of creating components, services, and using the Angular CLI to set the project up were all things that were very new to me and required a great deal of research. I utilized the many articles available to me on sites such as Medium.com, various blogs, coding tutorials and YouTube channels. Using these sources, I was able to have a thorough understanding of concepts such as angular services, observables, model schemas, and the overall angular project structure.

Learning angular was crucial to the project's success early on, however this was needed just to set the project up and have a good foundation. Once the project was set up, my knowledge of other aspects of the Mean stack would be tested. Therefore, I had to make sure that I first and foremost had a strong understanding of java script. Contrary to many applications that I have developed, this application would require a deeper understanding of advanced java script concepts. I would need to understand java script promises, callbacks, observables and behavior subjects, subscriptions, and an endless list of other features. Along with this java script knowledge, I would need to really understand typescript, a language that appears to be a derivative of java script and is needed in angular components. Although a daunting task especially considering the sheer amount of resources available, I found the best helps in online classes and books. First, I would read a few articles from an online coding blog or tutorial to get a basic idea of the concepts. After I did that, I would then move onto other resources that could fill in the gaps in my understanding, such as online classrooms like udemy or team tree house. I also utilized some actual coding books that were available online, one of my favorites being a book entitled "You Don't Know Java Script," which is actually a series of books aimed at making sure the reader has a true understanding of the Java Script language. Since I was learning as I progressed, I often turned to these resources when I was stuck in my code and needed some background knowledge and inspiration that would assist me in brainstorming an alternative route to solving my problem. This method served me well and is something I will always do for future projects.

Since this is a client-server system, obviously the server-side code was something that needed to be implemented in an organized, thoughtful manner. In order to accomplish this, I needed to make sure that I thoroughly understood Node.js and Express.js. I invested a lot of time and energy into learning these two languages, using resources such as udemy, youtube, and endless tutorials. This truly paid off, since my application is constantly making calls to the server-side code in order to access various api's and retrieve user data. A challenge that I confronted in learning these server-side technologies would be that many of these tutorials and online classes do not always address the type of problem you are trying to resolve. However, this usually just required either a more in-depth google search or perhaps paying a few dollars to get the extended version of a udemy or team tree house course. Often times I needed to

Jordan Phoenix
CSC 400 Project Design(Late)
10/18/18

combine knowledge and advice from several different resources before I could solve my problem, but more often than not a solution was found.

Not to be neglected in this final review would be the various api's that I was able master and use. Most prominent of these api's would be the jwt-helper api, which helped me create the user authentication system, bing api, youtube api, reddit api, and other plug-ins that helped me to do things like install an interactive date picker. Installing these api's successfully was a huge challenge since the documentation on them is either too vague or far too dense and packed with information that is not related to what I want to accomplish with the api. This part of the implementation required patience, research, and a little creativity in order to make them function as I intended within my application.

There were very few parts of this project that I did not find challenging. However, learning these tools through research, trial and error, and implementation really gave me a thorough understanding of technologies and skills that I can re-use again to either improve on this project or begin my next application.