Senior Project 2

Zachary Torretta

Computer Science: Business Software Development Concentration

Lawrence Technological University

**Abstract:**

This project, like senior project 1, was brand new to me. Initially I was not sure what to work on, but I decided to have something to do with modular arithmetic and API’s. I hadn’t worked with databases before this semester, so I learned a lot as I was working on this project. There are a lot of behind-the-scenes functionality with this project that may not be apparent to the user.

This is because it has to do with the database functionality. I have my own database I used for this assignment. Each user that submits an assignment to track is added to the database. I also have some extra functionality to allow the database to be stateless. To allow for more growth in the future.

The assignment has people enter their credentials, an assignment, and a due date. It then takes them to a page with a count down timer until the assignment is due. It uses modular arithmetic to calculate what day this assignment is as well. The main purpose is to use this API and modular arithmetic to allow ease of use for people with due dates coming on a web page. The actual detail of the web page is minimal of course. I did not spend any time at all to decorate it and make it look pretty. That was not the point of the project.

**Introduction**:

At the beginning of the semester, I was not sure what I wanted to do. It took me a while to even figure out what I had planned. So, with the modular arithmetic idea presented to me, I decided to just go with that. However, I did not know what I wanted to do exactly. Out in the real world, I would most likely work with databases to some extent, so I wanted to incorporate that into this project. With the help of the API, database, and modular arithmetic, I brought this app to life. It allows people to easily see when something is due, and it can be tracked at all times.

**Review**:

This website helped me learn how to create the API I used in my application. I was not familiar with API’s at all prior to this semester, so it helped out tremendously. It helped me to create servers, routers, services, etc. (Node.js - RESTful API n.d.).

This source has to do with the famous zeller congruence modular formula. I used this to help come up with how I wanted to incorporate modular arithmetic into my API for the app. It was very helpful to give a clear understanding of how this helps with dates. (Zeller's Congruence 2009-2021).

This website was extremely crucial for this assignment. It basically is an entire website where I am able to access libraries from to then use in my API. Each library has its own tutorial that I would follow to setup for this project. (npmjs n.d.).

**Main Application**

When I first worked on this project I was working with C++ in visual studios. I had no decided on developing an API into a little bit into the semester. I worked on a bunch of different options with the modular formulas to be able to extract information from a user and use that information to display helpful information to them.

I found it best to allow this application to be setup on a database. It allows people to fill out credentials and sign up in a way. You submit your information, it adds you to the database, and it redirects you to a new page displaying the application’s results.

Some important information is that I have a lot of extra functionality in the application. For example, each person in the database is unique. As in their id number. So, if you were to try and input a duplicate into the database, it would give you an error.

To start off in the main server file, it requires in a bunch of important libraries, packages at the beginning. I then use an engine in this portion for my html files. I need to be able to take information from the model schema in the database to then use and display on the screen. To do this, I needed to set an engine for the html, and use it as a render for the page. Here is an example of what this code looked line for this portion:

app.engine('html', require('ejs').renderFile);  
  
app.get('/', (request, response) => {  
 response.render('C:\\Users\\Zack\\Desktop\\SP2\\SeniorProjectTwo\\html\\index.html');  
});

I then connect to the database via mongoose. It is mongoDB, a website that is used to setup databases. I just have a basic one that is free. A note here is that I do not actually have my credentials displayed in this server page. That would be terrible for security, so I brought in a library called dotenv. It allows me to save a text file inside this application with my actual credentials, and it reads those credentials for the database sign up.

The server then sends you off to the router once it checks what has been typed in the URL. Once you are on the localhost website, it shoots you off to the router. My router has extra functionality that I added to allow this to work with API applications such as postman. Postman allows you to perform different methods onto an application like PATH/DELETE, etc.

This application only allows for posts and gets. This means that you post your info, and you get submitted into the database, and you get your information. I have inserted a middleware inside this router file. A middleware is a type of intermediary function that runs a verification of sorts to determine if the application can continue and the test passed, or it must end and send an error. I implemented the middleware to assure that the user is only working with gets and posts. For example, you don’t want the user to be able to delete other users. Here is the example of the middleware function.

***router***.use((req, res, next) => {  
 if (req.method === 'GET' || req.method === 'POST') {  
 ***console***.log('good to go');  
 next();  
 } else {  
 res.send('Please enter a GET or a POST.');  
 }  
});

If they have a get or a put, then it continues to the controller. This is where a large bulk of the code is. I created two functions. One to post a user, and one to get the user information itself. Each of these functions calls into the database service, where it creates a new object to send back to the controller to use. When something is sent to the database service, it usually is the incoming request body. For example, say the user says their name is Joe, that would be inside the request body.

This body is sent to the service, and inside the service it imports the model file. This model file is a schema or a model built for the database. For example, it would simply the information in the database. Like “users” schema and there is a field for “name”. The database would create a new object with the request it received (with name joe). The controller for the post function would then take this new user and all the information they typed (request body), and save it to the database.

The other function in the controller works very similar to the post function I explained above. The difference is that instead of saving the request body to the database, it is retrieving that information from the database. It finds the information related to that person by the id they submitted. It then pulls out the date information they entered for their due dates.

It sends this information off to the main html file with the logic. So, this app has two main notable html files. The first is the index.html file where it displays right when the application loads. It is an html file with a form action as a method post. Because this is where they are typing their information to post to the database. This html displays text boxes the user can post to.

The more complicated html file is the countdown.html file. It takes in 3 things from the controller, day, month, year. I then used a built-in date function to get the time for the date provided. I then worked on all the logic inside of an interval function that updates every 1 second. So, the countdown timer updates every 1 second. I then get the current date by just using a code similar to this:

const now = new ***Date***().getTime();

I then use several Math.floor functions to get the days, hours, minutes, and seconds. I then pull the document information from html file to display the countdown. However, this file is also where the modular arithmetic comes into place. I worked with the zeller congruence modular formula with the database information to produce the actual day that the assignment is due.

Instead of setting a variable to an int or float in javascript, you set it to let or const. I wanted the zeller formula to work as if they were all ints, so I used Math.floor on the part of the modular equation that has any sort of division. If the month entered is January or February (1/2), then you do the same formula but the month is 13 for January, 14 for February, and the year is year--. Here is a snippet of the Math.floors and the modular formula used.

let ***t*** = ***Math***.floor(((13 \* (***m*** + 1)) / 5));  
let ***p*** = ***Math***.floor((***k*** / 4));  
let ***u*** = ***Math***.floor((***j*** / 4));  
 ***z*** = ***q*** + ***t*** + ***k*** + ***p*** + ***u*** + (5 \* ***j***);  
***z*** = ***z*** % 7;

Once all the logic in the html file is done, it outputs it to the screen. It outputs how far away the day is with a countdown, the actual day it is (Monday, Tuesday, etc), and it re-displays the date typed in as (mm/dd/yyyy). If the date has already past, then it simply displays an expired message, and the timer will not work.

**Conclusion**:

The project does exactly what I wanted it to do. I learned a lot about html and API’s during this project as well. It was a great opportunity to dive into something I don’t have much experience in because I’ll most likely have to deal with databases in my career. I believe the GUI of this project is very simple and easy to understand for a user. This was another big goal of mine. I wanted the user to clearly understand what was happening. I believe that this app could be of use to some people that have a lot of due dates for their career. It could be a helpful app for a manager of a branch.

**Summary**:

I was able to do what I wanted for this project; however, it could still be improved a lot. Some future ideas I have would be to implement dependency injection. To put it simple, it would allow for easier test cases to be ran on the API. If I were to redesign the API with this injection, it is easier to create fake instances of the database to run more tests on it.

Scalability is something that should always be thought of when developing an API, and the injection would help with this. Another avenue I could take for this API would be security. I could implement systems to salt/hash the secret information the user typed in. Because at the moment, any user can check any other user’s due dates.

Of course, the design and GUI of the website would be something that I could work on. I also have to run this project on my local server website because of domain limitations. This was a large limitation because I don’t have access to my own website, and my html experience is minimal anyway. My lack of knowledge about html definitely was one of my biggest limitation for this project.

# References

1. Node.js - RESTful API. Accessed 2021. https://www.tutorialspoint.com/nodejs/nodejs\_restful\_api.htm

2. *Zeller's Congruence.* Accessed 2021.

https://datagenetics.com/blog/november12019/index.html

3. npmjs. Accessed 2021.

https://www.npmjs.com/