Serpent

CS480

Northern Michigan University

Abraham Brege

advised by Michael Kowalczyk

Committee: Randy Appleton, John Sarkela

Abstract

Serpent is a tracking app that intends to hold the same use that Strava does for tracking your runs, bikes, hikes, etc, but for programmers to track their progress across various mediums of work. This is a new concept to the field, as such an app ceases to exist. In building Serpent, I planned to take an approach of how the development process works from start to finish—encouraging other programmers to get out of their comfort zone and build more with the skills they have gathered throughout their education. While this point stands true, there were road-blocks that arose which caused me to have to fall back on some of my promises. The product may not be everything I dreamt of and more, but it was an excellent learning experience. In all, excluding comments, the project amounted to 4,584 lines of code, 1,529 of them being frontend code and the remaining 3,744 being backend code. Next.js was used for the frontend and backend, a standard relational SQL database for the database, Artillery for backend unit testing, and Cypress for frontend unit testing.

1 Introduction

I have a hard time going where others have gone. It's not a positive trait, but one I'm stuck with. So even though the first is never the best and reinventing the wheel, especially in software engineering, brings amazing things, it's not for me. This made it hard to choose a senior project, because *mostly* everything has been done. But *mostly* is the keyword there, because you can imagine my surprise when I found tumbleweeds while searching for a tracking app for software engineers. Think of that, there is no tracking app for the people who make the tracking apps! So while I don't love web development, and I've never touched frontend code, I knew doing something new would give me the drive that I needed to get the project done.

So, why do software engineers need a tracking app? Well, right now there are two ways to track your progress:

GitHub - Tracking commits, repos, etc.

and

Leetcode - Tracking submissions to Leetcode problems, solutions, etc.

Let’s take a look at what they both do right, and where they come up short. GitHub has great quality of data, meaning that you can take any project, and see where you made incremental progress in that project via commits. You can use GitHub for work, school, personal projects, or anything your heart desires. This is a great strength. However, GitHub falls in the aspect of displaying this data, boring graphs and a clunky interface for navigating such graphs really kills any reason to have this data to begin with. Oh, and the fact that I don’t think GitHub, or any GitHub user, has ever really cared for a loud, colorful dashboard to see their stats, but that's besides the point… Leetcode is the antithesis of this assessment, their dashboard is beautiful, all of the colors, the meaningful graphs, it's wonderful. But they only track Leetcode data, subsequently making their quality of data poor. Don’t get me wrong, that’s no fault of theirs, why on earth would Leetcode track anything more than Leetcode data? But for the sake of the argument, it’s a narrow scope they got going on there.

So what would Serpent do differently? Well, to start it gathers data from both Leetcode and GitHub, groundbreaking. But more importantly, it also introduces the highest quality data possible—time. You may have a commit, or a green check by the Leetcode problem you completed, but who knows if that took you a couple minutes, an hour, or even a week to do? I know you don’t, because there’s no way to tell! That’s the kicker, tracking how much time you spent on these tasks. Likewise, combining the good parts of Leetcode and GitHub’s graphs to see all of your data and more in a singular beautiful dashboard, oh it’s a Type-A heaven.

2 Scope

We had our first lesson here. When I started the project, I wanted Serpent to have:

- A fully social aspect allowing users to see the progress of their friends and make posts to bloat their accomplishments, just like Strava
- A fully modular dashboard allowing users to pick their style of graphs and move around their dashboard freely
- The whole 9 yards with settings, user account customization, 2FA, Google Auth, seamless syncing of data, you get the gist
- Separate dashboards for GitHub, Leetcode, and other projects all in their own respective dashboards
- A VSCode extension to make tracking seamless (that I’ll talk about later)

- And all of the other features that actually survived

This was such a gross oversight of my technical skill that it's laughable looking back. Mind you, I had never even touched a frontend project before! Now, I'll cut myself slack in that my courseload being as hard as it was did play a massive role in having to shorten this scope, but I don't think I could've pulled it off regardless. So where did I go wrong, and what did I actually end up getting done? The first one to leave the list was the modular dashboard. Upon designing, I realized that I actually didn't like the look or feel, and it would be a great deal of work to implement, so I canned it and now just have static graphs. This led to the next drop off: the separate dashboards. This was a great idea, and with more time I could have, but nearing the middle of the semester I decided it wasn't in scope. I had just completed the first dashboard and was pressed with time to get the rest of the pages done so told myself that I would get to it if I had extra time at the end (I knew I wouldn't), so, canned. Finally, I'll group the 'whole 9 yards' and social aspect into one as they both pertain to accepting other users. Originally, my plan was to have a normal app that allowed users to create an account, socialize, etc. However, when the question arose of how this app would be hosted, paying for an AWS server to host my senior project's 9 users wasn't what I jumped for. In fact, I had two leftover laptops that I could easily get Ubuntu running on to host the small amount of traffic. It seemed perfect, and it almost was, but then came VLAN. After a couple articles about network security and a couple Youtube tutorials on getting Ubuntu installed, I had my two servers up and running and could SSH into them. The next step was to make sure the switch was secure so the devices on my parents network wouldn't be in danger with the foreign traffic. This, of course, is where a VLAN comes in to isolate the traffic of my server from the traffic of the home network. This was all and well, until our switch was too old to support VLANs. Okay, well that's not ideal. Likewise, by the time it gets delivered, winter break will be over and I'll be back at college. Hopefully it won't be too much work to get a new switch setup and configure the VLAN over facetime with my parents… Yeah, you can see why this got canned. After the new switch came in, I looked up what we would have to do to get everything setup, and I stopped counting after step 20 and told my parents to return the switch (mind you the configuration itself wasn't that bad, making sure it was secure was where it went astray). The cherry on top—my development up to that point was strictly based on getting authentication working along with setting up the logic to support multiple users, so I also got to throw away almost all of my code and start from square one.

They're all good lessons to have learned, some more of my fault than others, but all helpful moving forward. One more canning took place: the hopes to document all of my progress rigorously to show the development process in full. I am quite disappointed that I let this slip through, but as work and class got increasingly stressful, I was working on fumes for most of the project, not being able to allocate enough time to a detailed log of progress.

3 Design and Approach

With a huge goal in mind and no work done, obviously I had no clue where to start. What I did have though, was an understanding of my strengths and weaknesses. I did some backend work at my internship, so that wasn't as daunting, but I knew the frontend would be the bulk of my problems and that I should probably face it after I got the easy stuff out of the way. Here was my thought process:

1. You can't know what you're going to make unless you know your model. So the database design has to come first.
2. It's easier to recreate than to create. Designing the frontend first makes it easier to code later. This allows the learning process to go from 'how do I make a frontend web page', or 'what is a component' to 'how do I make a box', 'how do I make a graph', 'how do I make a timer'
3. When you have a frontend design and a database, if you think for a second, you actually have every single endpoint you could possibly need as well. You know what data you need to fetch for each page, you know where to send a user when they click a button, you really know it all.
4. Once you have your endpoints, you have the data to provide the frontend to help develop the frontend and make sure everything fits.

This was a perfect use of my strengths, starting with database design, something I am very familiar with, getting to have fun designing my project, moving to another familiar topic of creating endpoints, and ending with the bulk of the work and learning which would be the frontend.

4 Implementation

Following my plan, I started by sitting down and thinking about every possible table I could need and their respective columns. Then, I created a diagram to show the relations between tables in Draw.io to be able to reference when I was developing. In all, this took a couple hours to finish.

Next was the designing. I have absolutely zero ability to make things aesthetically pleasing, and have never used Figma. So, I started with pencil and paper again, sketching up the rough idea of what I envisioned. I made sure to start with the easiest pages, tackling the home, about, login, and signup pages first (although the login and signup pages got scrapped pretty early in development). Then I worked my way up into the intermediate difficulty of pages, the social posts page, the settings page, the activity editing page. And finally, I spent the bulk of the design on the dashboard page, making the decision to go all in on one static design over a modular one. This meant that I needed to fit all of the data I would collect in my project onto this one page, making sure everything had the correct spacing and size, didn't look crammed, and was easy on the eyes overall. This was hard, and it stumped me for days on end sometimes, being fed up with previous ideas, having multiple different dashboard designs side by side comparing what I liked from one and not the other, and about every other setback you could think of. But in all, after about two weeks, some spent also on the setup of the project, I had the design finished.

Now for the backend. I started with getting the data from GitHub and Leetcode. Leetcode uses GraphQL and keeps their url public, so you actually can just directly query their schema which is nice. This was huge, because at the time I was worried about rate limits of other users fetching data, this completely negated that worry. It's a double edged sword though, because they also don't advertise this use. So the structure of their data is a mystery. There are a couple repo's and reddit posts out there about very surface level queries to get a user's account info like their name, profile picture, etc. But when it comes down to the finer details, you're left guessing the name of the fields to see what matches. Not to mention, I don't know if you've ever used GraphQL (I hadn't until then), but it's not exactly a dream to use even if you do know the structure. The saving grace here was ChatGPT. Instead of having to search Leetcodes endpoints myself for the queries to get the exact data I wanted, I just asked ChatGPT what the query would be to get certain data, and it got me information that I couldn't find anywhere else online. How?

Well I'm sure there was more than I cared to look for on say page 87 of Google, but luckily for Chat, it is trained on that anyways, so it didn't seem to think the task was as hard as I did. The querying of this data is trivial so it was easy to implement the calling of a GraphQL endpoint and getting its response in a JSON object. GitHub was easier, they are high on being free and easy to use, so they just have extensive documentation about what to query to get the data you want, boring.

With the Leetcode and GitHub endpoints in place, I took my first wrong turn here: veering away from my plan. I planned to do the entire backend and then the frontend. This would have served me way better, but inevitably, once I had the login and signup routes done and had configured Google's OAuth, I thought it would be cool to get the entire authentication front and backend working just to have something to show. This led me to burn three weeks just to realize halfway into February that I didn't need authentication anymore. Boom, almost all progress is gone. This is where I gave up on the idea of documenting all of my work and learning, it was also where I gave up on making commits one by one because I didn't know if I would be approved to make such a drastic change, so mostly everything is done on one branch and commit from now until early April when the project was done. Granted, I didn't commit anything in the meantime, so if you look at the repo you will actually see everything added in groups on April 9th commit by commit, but it isn't a representation to what actually was done when.

After this, I decided that after taking this hit I would start on the VSCode extension. This was such a short lived dream that I didn't even cover it earlier as something that got canned, here's how the attempt at the extension worked:

1. Set everything up in the repo to make the extension
2. Realized that the extension was useless if I didn't have the endpoints that it would call to work (the time tracking endpoints)
3. Realized that the tracking aspect was the most boring and I would never find the motivation to do it
4. Realized it was only really a feature to begin with to promote ease of use to other users
5. Realized I was the only user now
6. Realized that I didn't care enough
7. Deleted the branch
8. Canned.

After all of that, it was mid February and I had done nothing but design a frontend, design a database, and get Leetcode and GitHub data. So naturally, I completely devolved from my plan. Short increments of fun and healthy progress became cramming my homework and work into the weekdays and devoting days in whole, sunup to sundown on the weekends to force myself to make progress. Don't get me wrong, it worked, but it created a slew of mistakes that would bite me later.

I decided to get the hard part out first—the dashboard. Not knowing how hard it would be meant waiting until the end to potentially get hung up on it a week before the project was due wasn't an option. Luckily it actually wasn't too painful, as I found, searching up how to make a component, how to make it resemble a box, how to make that box accept props to display data, how to section out the box to hold that data—all really weren't that bad. Likewise, I used Chart.js for the charts, and I really just had to figure out which charts to use and display the data accordingly. I will say, I tried some things like a heatmap for the GitHub commit graph, but I couldn't get them to work and unfortunately had to settle with about just as boring of a line chart as I dogged on GitHub for having… Poetry in motion, or graphs in this case.

Next was the tracking page. This was also not rocket science. The good part about never doing any frontend work before is that the frontend work I chose was pretty light. Breaking it down I needed two boxes, one to store the users' recently logged activities to be able to select and edit, and another to actually start and stop the stopwatch while being able to select the activity they were working on. This, from a frontend standpoint had its challenges, centering two icons on a grid with three columns on any size of screen was a headache. Likewise learning how to use state variables to track which activity you have selected while the timer is running to link them that activity to the interval was a learning curve, but using state in React is still considered beginner, and once you do it once, it's not hard to replicate to the other state variables you have to manage. The hard part actually came from the fact that I was tracking what languages were used where and their accumulated time, but wasn't tracking anything about the languages! Sharp. So after wondering why none of my activities were adding to the accumulation of the total time spent on languages, I finally went back and changed the dashboards util functions and the logic on how everything is tracked. Not to mention, making sure that your total duration is the very start of the first interval to the very end of the last interval wasn't terrible, but wasn't fun to try and debug when initially your times are off by an hour or two in duration.

The rest, editing and activity pages were trivial, the editing page is just pulling up a session id to edit the activities or intervals at which they occurred. While the activity page is really just a wrapper for an insert, update, or delete query in the database, all not too bad.

At the end, I implemented lazy loading so that the dashboard didn't take 15 seconds to load while the endpoints were being hit to get the data and it loaded the main page before the data. Oh, and you can never forget the unit testing, which was quite easy to implement. I used Artillery and Cypress and checked all of my GET routes, because only having one user meant the POST routes affected my data in a critical way, and Cypress on the main pages: home, dashboard, and tracking.

5 What I'd Change

There's a lot that I like about the project, don't get me wrong, but there's no shortness of ugly parts either. For example, once the authentication was gone, I just took out the aspect of blocking pages from a user, I didn't change any logic around supporting other users. This leads to a bunch of 'userId = process.env.USER_ID' floating around my code. Why? Because I still base everything off a user_id in the database as if there were other users! I could gut half of my database logic and make my code way cleaner if I just took the time to refactor it all out at the start, but I didn't, and I've been treating my user_id as if it will ever have company (it won't).

I made the first Box.tsx component and instead of following the correct rules of component design, I just made functions for TextBox and GraphBox inside of the component itself instead of creating their own separate components which use Box.tsx. It's small but a sin in the world of frontend. Likewise, other files use Box.tsx, so I should have refactored this component to be outside of the dashboard, but I got lazy and never did. This is probably a 3 minute endeavor but I don't care to do it.

The last thing would be the clunkiness of the tracking, you can't track more than one activity at once if you multitask. And overlapping time frames have to be done in the editing page.

6 What I want to add

Well, to begin with: all of the things I removed. But furthermore, I think it would be cool to add features like taking statistics about the lines of code you actually wrote. I'm sure there is some way to get hands on the code written from the repo if the repo was public. With a VSCode extension, I think you can also track the lines of code written while the extension is enabled. So combining these to give the user fun insights into something like the number of times they write a function, use a for loop, use a variable name, or anything else would be fun to have.

7 Conclusion

In all, I learned a lot, and I'm happy with the output. While it's not a modular application open to the general public that combines all of your data into one spot with a bunch of different graphs and an easy-to-use VSCode extension, it was fun to make, and still about 60% of what I wanted.

In case you were wondering, I'll probably never use Serpent. The name 'Serpent' actually came from about week three of working on the project when I watched a video outlining how bizarre tracking apps are, motivating people to accomplish things for some external source of joy rather than doing something they genuinely want to do. So, I figured this tracking app, like all others, is the devil, and Serpent was born (and will die right after I give this presentation).