Abraham Brege - Senior Project (Winter 2025)

Serpent - *Like Strava for programmers.*
https://github.com/abrege11/serpent

Table of Contents

1. The Vision

Computer Science for most is reinventing the wheel, I cannot stand this. The beauty of this subject that I hold so close to my heart also happens to highlight one of my greatest character flaws—I hate solving an issue that has already been solved. In fact, I am transitioning away from classical computing (to quantum computing) for the same reason that I pivoted away from information technology (IT): you are fundamentally limited in creativity. In IT, you're plagued with solving other people's issues using software made by other people—using network sniffing tools wondering 'hm I wonder why it can't do this specific thing', or looking at documentation for software that you're company is licensing and realizing that it is poorly structured and you're the bum who has to pick up the slack. You're a slave to others' intellect, imprisoning your mind. Computer science is a bit better, and don't get me wrong there are countless topics and opportunities for unique growth in this field, but unless you're going into research, they hum a similar tune. If you're not recreating a 'better' app, you're using the same libraries, often wondering why they can't do something a certain way, why the juggernauts of programming languages lack certain features. You're using a programming language made by someone else, to recreate an app made by someone else. For any sane human this is a normality, something most have never even considered, but for me it's a bus hitting my frontal lobe at 60mph and crippling my need for creativity.

Enter Serpent. Now, obviously I was dramatic at first, I'm not writing my own web sockets, creating my own programming language to write the app, or writing my own libraries, but the saving grace is that this hasn't been done yet! As shocking as it is, the people who make the tracking apps never thought to make themselves a tracking app… I mean, maybe it has, but a search of the app stores and the first few pages of google didn't have anything about an app to track your time programming. I know what you're thinking

"But what about Leetcode and GitHub, they track your progress"

This is true! But they both suck in their own unique ways. To understand why each is bad, you first must realize the different aspects of tracking:

1. Quality of data
    - The type of data being tracked, how broad the data tracked is in comparison to the work that was done.
2. The presentation of the data
    - Self explanatory, spoiler, Leetcode is good at this with a beautiful dashboard containing cool graphs. On the other hand, GitHub is embarrassingly bad at this, displaying boring graphs that no one cares about.

1.1 Why GitHub is bad

Quality of data: 7/10
Presentation of data: 4/10

GitHub thrives in their quality of data—commits. Commits cover such a broad set of data. It doesn't matter what project you're working on, it could be an assignment for school, project for work, or a personal endeavor, it can all be tracked. So why is it a 7/10? Simple, how long did you spend working on that commit? What if you don't have your project setup as a Git repository? What if you were working on a paper-based coding assignment? What if you were even doing a leetcode problem? The list goes on.

NOTE: I'm being overly critical of course but let's not forget that this isn't a flaw of GitHub by any stretch of the imagination. Why in the world would GitHub track any of this? GitHub tracking commits on a project that isn't a Git repository, i.e. there are no commits. I mean c'mon, obviously not. But I bring up these points to highlight the holes needed to be filled in this area.

GitHub lacks in presentation, 'Insights' is boring. Oddly enough that's all it takes to be a 4/10. They don't have a 'dashboard' like structure, rather a pagination model. This I would argue is a flaw, even professional software for tracking implements dashboard-style interfaces to display data. Likewise, a lot of the data displayed is very verbose, low level metrics, good for its niche consumer, but I am marketing towards a broader, more casual audience.

1.2 Why Leetcode is bad

Quality of data: 4/10
Presentation of data: 10/10

Again, to no fault of Leetcode, their data is bad. I mean this holds similar defenses as GitHub, why in the world would Leetcode track anything other than Leetcode problems. However, I do love the data they track—submissions, solutions, contest data, profile views and many more are the reasons Leetcode got a 4/10. Their data is the antithesis of broad, but it is so rich in information.

Leetcode knocks it out of the park for presentation. I can only dream of creating a dashboard as beautiful as theirs. I, along with many others, have experienced times where the only reason I even log onto the site is to uphold my daily streak, along with knowing my progress will be portrayed so elegantly on my dashboard. They hit this one out of the park.

1.3 Why Serpent trumps all

Classical computing uses 1's and 0's (bits) for computations. Quantum computing engulfs this concept, being able to represent the traditional bit, while furthering the concept by allowing superposition—the combination of these states. Serpent is the quantum computing of GitHub and Leetcode.

Linus, being the angel he is, loves the idea of open source software. This means that GitHub provides an API that allows developers to use their data. It technically isn't limitlessly free, but it's damn near, allowing up to 10,000 authenticated requests per hour per user… Sounds pretty free to me. Likewise, Leetcode uses a public GraphQL database which you can query using the respective queries. Therefore, Serpent will do the following:

- Tracks user's GitHub activity
- Tracks user's Leetcode activity
- Tracks additional metrics like time spent on GitHub and Leetcode tasks (commits, submissions, etc.)
- Allows users to create their own activities to track like Calc2 homework, a personal project setup as a GitHub repository, and really any other activity imaginable.
- Display the combination of these pieces of data in a great dashboard, picking the *good* data from each medium and *ignoring* the boring data.

So there it is, my sales pitch for this software project, and why I think there is a market for it. I can't promise this is something that other people will want to use, but the beauty is that I don't really care! I'll use it if no one else, and after all, it's a senior project, so I won't lose sleep over 'wasted time' since it was something I had to do anyways.

2. The Stack

The stack for this application is quite simple. Next.js for front and backend development, a standard SQL relational database, JavaScript for database operations and other miscellaneous tasks, Google's OAuth API for authentication, along with GitHub's REST API for GitHub authentication, and finally my home server to host it all.

2.1 Why I chose what

2.1.1 - Front and Backend

Next.js is the framework of choice for Syncurrent, the startup that I currently work at. Don't get me wrong, I primarily web scrape, so I use Python in my day to day there, but I did some research and see that Next.js is the progressive choice for new web applications at the moment. That said, I have never touched a frontend code base past Cypress tests. So this decision came from nothing but speculation of a good fit.

2.1.2 - Database Operations

On the other hand, I used JavaScript exclusively for database operations at Northcross Group, so I am pretty comfortable with the workflow. This was a strictly preference-based choice, JavaScript doesn't do database operations any better than Python, and although I'm more comfortable with Python at this exact moment, keeping the code base as JavaScript based as I could sounded more appealing than a random flurry of Python files. That's really all.

2.1.3 - The database itself

A relational database wasn't my first choice, as most recommend non-relational competitors like GraphQL for 'social media' type data. While I was leaning towards this as I am technically creating a social media, it's important to remember that the point of Serpent is not to be social, rather that is an additive you can choose to partake in. Therefore, the strengths of such databases seem to be the power of recommendation, like an 'Explore' or 'For You' page. I don't want this feature, so a relational database database will fit my needs just fine.

2.1.4 - Authentication

Developers preach that if you don't have OAuth, you might as well not have an app. No one wants to enter in their account information anymore, they want to click 'continue with this account' and get the fun started. I've never cared about this personally, but I see the point, and it's free, so it's really just lazy to not implement it.

2.1.5 - Hosting

Lastly, I don't want to pay a dime for this stuff, and Besser Company was kind enough to give me some end of life laptops before I departed to college, so the only thing in between me and the freedom of not paying AWS bills is again laziness. After all, I don't expect this app to blow up so for right now I can get by with a home server (I hope). If I do somehow outgrow my home server due to traffic needs, that's a good problem to have and AWS is the next best option. But I'll cross that bridge when I get there. For now, two Dell laptops running Ubuntu are my choice.

3. Features

I touched a bit on features of the app prior, but I'll provide a more detailed list of what I would like Serpent to accomplish as an app.

The following are 3.x

1. Tracking user's GitHub activity

- I want users to be able to link more personable data to their GitHub activity like adding how much time was spent on a certain commit, issue, branch, even an entire project.

2. Tracking user's Leetcode activity
    - I want users to be able to link more personable data to their Leetcode activity, like adding time spent on a certain problem/submission, solution, contest, etc.

I hone in on these two points a lot, but for more context, think of a GitHub commit or a Leetcode submission like a run, bike, swim, etc. Imagine if you were just like 'I went on a run!' and nothing else. It would be pretty boring. Serpent is what provides that duration of the activity, how far you traveled, where you traveled, and even the map of travel, which I'll get to later.

3. Tracking user's activity elsewhere
    - Allowing users to create their own activities, maybe it's a class, maybe it's a homework assignment, maybe it's an independent project not tracked using Git.
4. A great (maybe modular) dashboard
    - Later in this document, you'll see that I designed the original dashboard to be modular, allowing users to add and delete data as they please. As I've thought more, I've considered removing a layer of modularity, allowing people to add and remove different graphs from a predefined template. This is up in the air, but regardless I know I need a dashboard.
5. A VSCode extension
    - Part of the project is making the app easy to use. People aren't going to use a clunky tracking app, it's already an annoyance to track, I have to make it easy. A VSCode extension is definitely a move in the right direction. I picture a way to have a module pop up allowing users to start, stop, and pause without even leaving their environment.
6. Tracking itself
    - Tracking is the pinnacle of this app, so it's important to get it right. Here's what I envision. You can start tracking a 'session' and split that session into 'intervals'. Maybe out of an 8 hour session you had 3 hours of Leetcode, 1 hour of downtime, 2 hours of one commit, 1 hour of another, and 1 hour of homework. You see the point. I also want to make it possible to go back and create intervals. So if you don't want to tediously start and stop, or are working on things simultaneously, you can edit a session after and create intervals manually.
7. The social aspect
    - I want users to have the option of adding friends to see their work, the keyword here is option. There will be a social tab, but you don't have to partake, and you won't miss anything if you choose to not participate.
8. Nice profiles

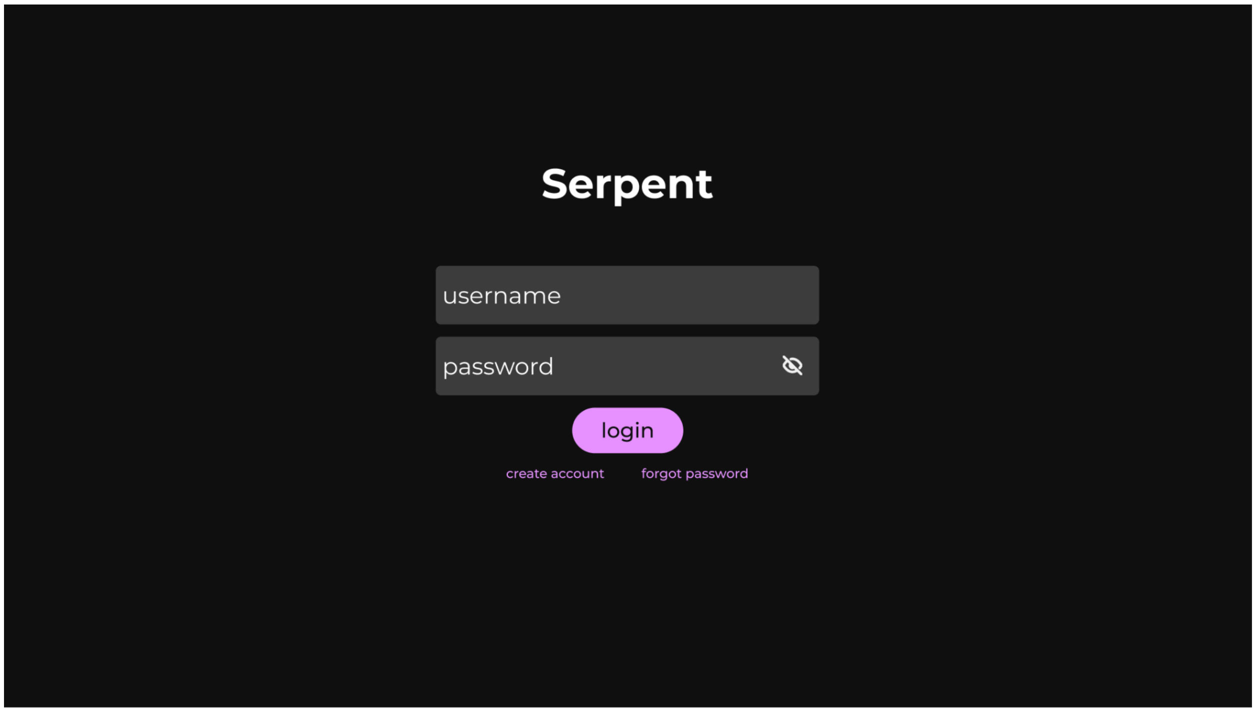
- To me, a profile is the heart of any app. You want to feel proud of your profile, you want it to display the right information, show off your strengths, and be pleasing to the eye. There is no set way to accomplish this, and I haven't drawn up exactly how I will. But it's at the forefront of my mind.

9. Graphs, and damn good ones
    - Graphs are a make or break for this type of app as we've covered. My goal is to let users pick their favorite graphs for their dashboard. This will be a challenge to get the data in a format that will allow a user to choose different graphs, but not rocket science. The only unique aspect is held in my concept of the 'map' of your activity. A cool graph I thought of, which I'm not claiming is original by any means, is a simple line graph, but coloring the lines in ways to show what you spend your time on. This can be seen later in the mockups.

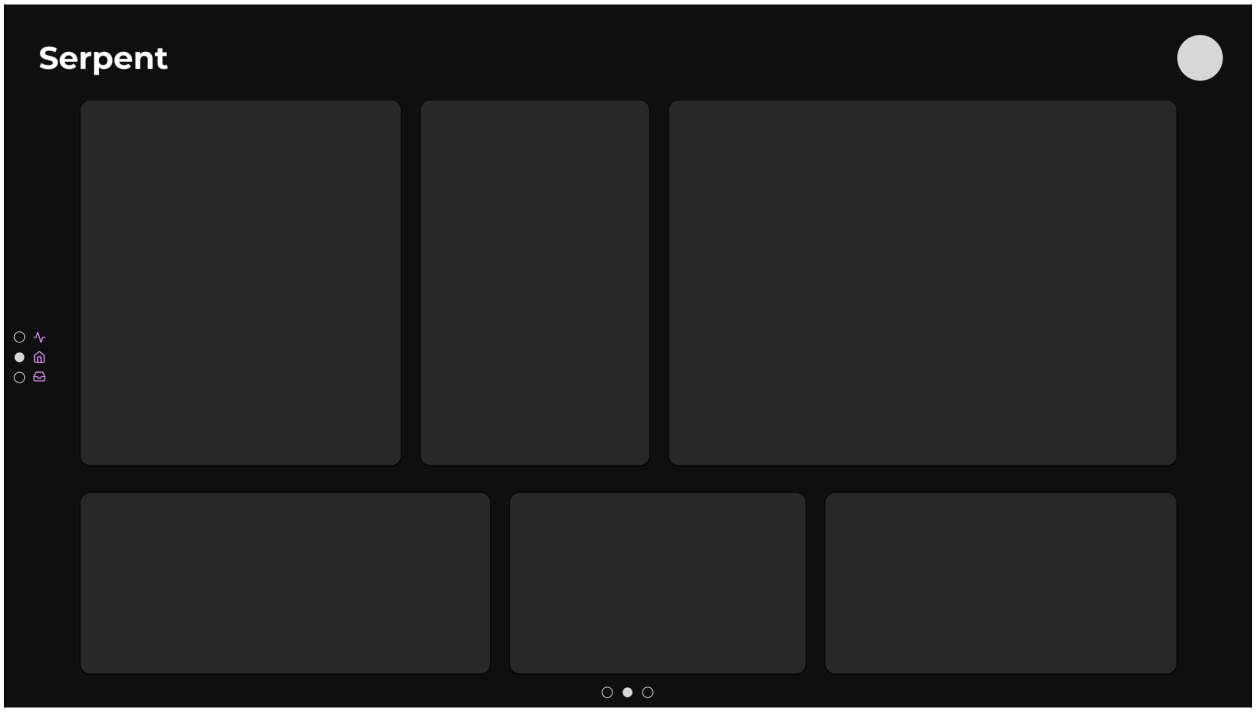
4. Mockups

I'll drop all of my Figma mockups for the MVP below and then cover them.

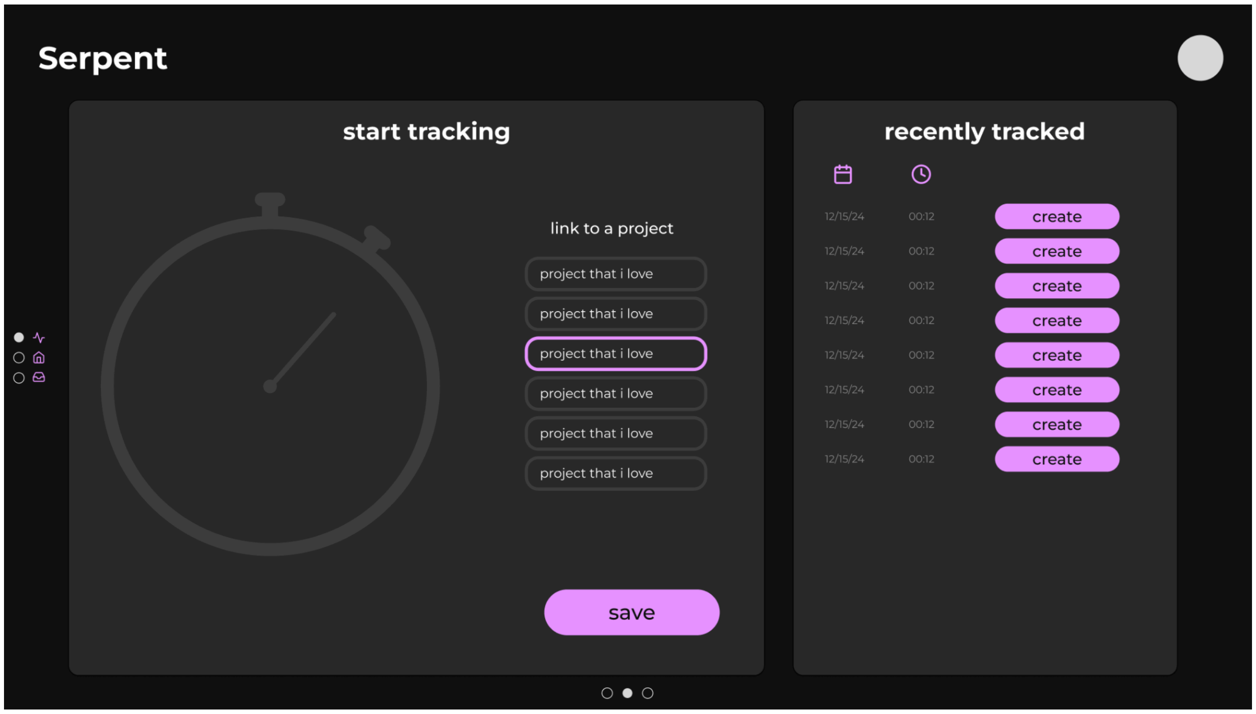
4.1 - Login



4.2 - Dashboard

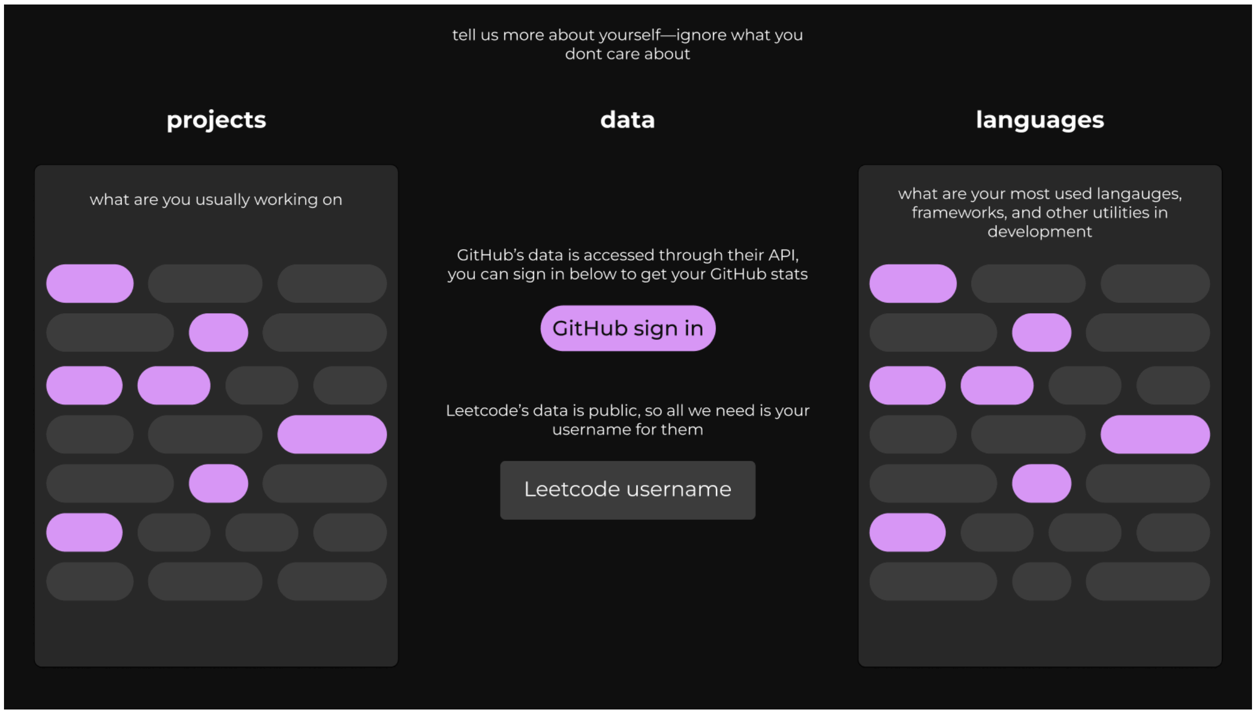


4.3 - Tracking



12/15/24	00:12	create
12/15/24	00:12	create
12/15/24	00:12	create
12/15/24	00:12	create
12/15/24	00:12	create
12/15/24	00:12	create
12/15/24	00:12	create
12/15/24	00:12	create

4.4 - Preferences


tell us more about yourself—ignore what you dont care about

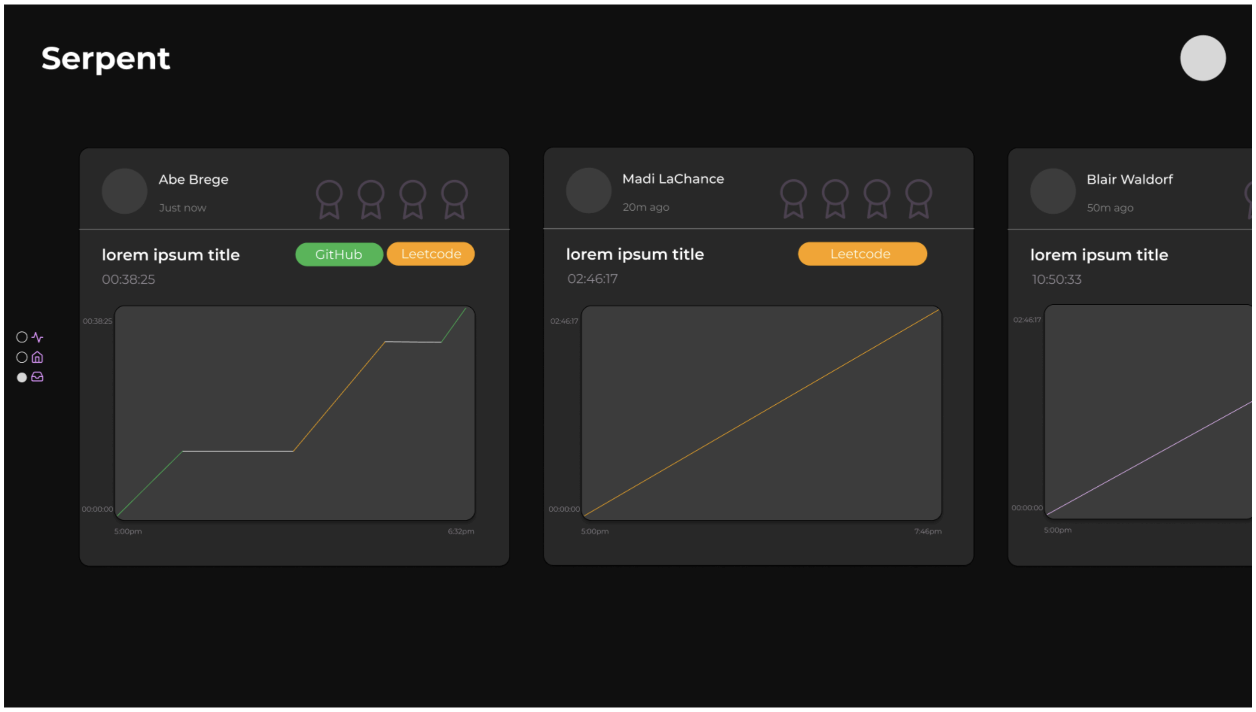
what are you usually working on

GitHub's data is accessed through their API, you can sign in below to get your GitHub stats

Leetcode's data is public, so all we need is your username for them

what are your most used langauges, frameworks, and other utilities in development

4.5 - Social Media



4.1 - Login

Self explanatory, just a login screen.

4.2 - Dashboard

This is the original idea of the main dashboard, each blank square is the placeholder for a module that could hold some data to display. I do like this idea and might go with it, but I don't know if my heart is set on it. If I can find a way to represent the data in a discrete way allowing users to customize certain sections, I'll probably go with that.

4.3 - Tracking

I will most definitely change this ugly mockup of tracking. I mostly just wanted the idea to be out there. I treat that as more of a framework. The idea is the let users have a list of recent commits or submissions, along with other custom activities to tie to intervals of work.

4.4 - Preferences

Pretty simple as well, just a cool way to allow users to select things that they use to be able to curate their experience. Things like projects they work on a lot, languages they use, you get it.

4.5 - Social Media

This is probably what I'll stay with for the social media aspect. The only thing I would change is scrolling vertically rather horizontally. The horizontal scrolling was building off of another idea I had but it's subject to change.

5. What I have and what's to come

5.1

So currently I have:

- Login screen
- Create account screen
- Home screen
- About page
- Database created and structured
- GitHub data fetching endpoints
- Leetcode data fetching endpoints

5.2

So not much. Here is the long, and probably incomplete TODO list.

- Authentication
    - GitHub Auth
    - OAuth

- Dashboard
    - Frontend implementation
    - Endpoints
    - Database operations
- Tracking
    - Frontend implementation
    - Endpoints
    - Database operations
    - VSCode extension
    - Editing past sessions
    - Saving session to create posts later
- Settings
    - Just all of the settings. Ugh, that one is going to be annoying.
- Social Media
    - Frontend implementation
    - Endpoints
    - Database operations
    - Creating posts
- Profiles
    - Frontend implementation
- Unit tests
    - Cypress for frontend
    - Artillery for backend

And probably more that I'm not thinking of. All of this is a lot, especially for a solo developer, but this concept leads me into the final part of this proposal, the process.

6. The Rest

One of the largest points I want to hit on for this project is the idea of development itself. I have never done any frontend development, never delivered a web application outside of work in any capacity, and have never taken on a project of this size. The point is, it doesn't matter.

Before working in the industry I shared what I imagine is a pretty common mindset—thinking these types of applications aren't feasible for a solo developer. I remember wondering if there was some secret database that only corporations had that allowed the data to be secure, making a personal project not secure for other users. Similarly, maybe all of these cool features derive from secret in-house libraries that a solo developer couldn't implement. This is a naive mindset in hindsight, but pursuing computer science in any form leaves plenty of developers in a similar position. You know enough to understand the complexity of these applications but lack the skills to implement them. This, for me, created a temporary crisis wondering what I even can accomplish, limiting myself before I even tried. Interning at Northcross Group demystified all of this. Believe it or not, it all really is just software, and you have access to literally everything they do. They use the same languages, frameworks, and services that you can. I know this is obvious to anyone in the industry, but I doubt I was the first and only novice developer to experience this mindset—not thinking I can accomplish what these large teams can.

Now look, I'm not saying that because I interned at a few companies and realized that companies don't use some secret magic tool to make software means you should listen to me, but please read and digest the following paragraph regardless as I don't think I've ever felt stronger about a topic in my life:

Software engineering is beautiful for many reasons, but what I love most about it is that you need nothing but yourself, your mind, and your work ethic. There is NO OTHER INDUSTRY (besides math, art, and literature) that allows this level of independence in success. Think of it, if you're in mechanical engineering, you can't feasibly buy a CNC machine to develop your product. If you're an architect, you can't get cleared to build a bridge independently. If you're a chemist, you can't simply spawn a lab and get expensive chemicals to experiment with. The list goes on. Software engineers need a computer (usually provided by a university at minimum) and sometimes WiFi, that's it. It doesn't matter where or who you are. If you have your computer, not only can you make whatever you want, but what you make can be used in any context, recreational, professional, etc. That is something I think everyone takes for granted. This extends my previous point: there is a clear economic (and sometimes legal) barrier between hobby and industry in almost every other profession. Software engineering? Absolutely not. In fact, open source projects are the antithesis of this idea. I think this is glaringly obvious but commonly forgotten. Say I wanted to make an app, could a team of 10 do it better? Yeah, duh, but I don't need that team of 10 to do it, that's the difference. Software engineering is a challenge of manpower, not resources. Again (and in bold), **software engineering is a challenge of manpower, not resources.** Apart from hosting costs, which are a flaw in my point, there is literally nothing between you and Google. God this is beautiful, inspiring to say the least. Don't believe me? Okay, well I agree that you can't make a better car than Ford, a better index fund than Vanguard, or better tools than Dewalt, but can you make a better operating system than Microsoft? Yeah, in fact someone has, his name is Linus Torvalds, and while 'better' is a holistic measurement, it's a damn good competitor.

All of that is to explain the resources.md file in the repository. Here I'm going to document the resources I use to learn and develop this application. What you'll find is that it's just not that hard. These things are learnable, these resources are available, all you have to do is get the stuff done. God that's beautiful, and that's why I really want to hone in on the process of this application, how I tackle issues, overcome barriers, all of the above. People love to overcomplicate the simplicity of making something. Understanding that what you’re working towards is possible is half the battle sometimes.

7. What I Will Learn

- The Next.js framework
- Implementing new tools such as Google’s OAuth, and GitHub’s REST API
- Setting up and using my own home server to host my application
- Designing and implementing a relational database
- Designing UI/UX with Figma
- AGILE practices to analyze, design, code, test, and release a full-stack web application
- Querying public GraphQL endpoints
- Implementing a VSCode extension
- Unit tests