First Steps

---

1. Gained inspiration from playing Factorio to create a production chain calculator/visualizer.
2. Drafted initial design, terminology, and calculations.
3. Decided to create a Google Sheets app.
4. Created Instagram post outlining my thought processes up to this point.
5. Was in the middle of researching my Google Sheets tech stack when I realized how terrible of an idea using sheets was.
6. Decided to pivot to a web app.
7. Prototyped my calculations and initial design, learning the basics of JavaScript and the functional programming paradigm in the process.
8. Decided on fully developing the user system for the website first, in line with the tracer bullet approach to design.
9. Researched the tech stack for the user system, and settled on using MongoDB, Fastify, Vue, and Node.js.
    - Yes, I realize that designing a system around the tech used to support it and not the other way around is a bad idea… fight me.
10. Created Instagram post outlining my thought processes up to this point.
11. Learned Fastify basics in preparation for creating the user system.
12. Created Fastify Prototype.
13. Learned MongoDB, and general database basics in preparation for creating the user system.
14. Designed some modals to facilitate the log-in and account creation processes.
15. Learned Vue.js basics in preparation for creating the user system.
16. Created Vue prototype for user system.
17. Combined calculation modules, Vue prototype, and Fastify prototype into a single repo labeled **FactorioProductionCalculator**.

Deployment

---

18. Set up dev environments and package configs for client and server directories.
19. Began researching deployment options.
20. Settled on hosting my website from an AWS EC2 instance, which is essentially a publicly-accessible, scalable Linux virtual machine.
21. Created the EC2 instance.

22. Set up an SSH connection between the EC2 instance and my local WSL Linux VM.
23. Moved back to Kansas, and the SSH key stopped working. Spent too much time trying to add extra SSH keys tied to different locations and users / configure connection options to a different region.
24. Gave up and settled on using the EC2 instance connect option, which I probably should've just done from the start because all you have to do is press a single button.
25. Got sidetracked and designed train throughput calculators.
26. Decided to pursue deploying my static web files next.
27. Purchased a notebook solely for taking notes on this project.
28. Settled on self-hosting, because I felt it would be dumb not to use the resources I'd already allocated in the EC2 instance. The obvious choice for that became NGINX.
29. Installed NGINX, and followed a simple guide to set up a server block for my website.
30. Purchased a regrettable domain from a regrettable website.
31. Learned DNS basics, and configured my domain's DNS A records to point to my server's IP.
32. Returned to server-block configuration, and debugged for like 6 hours straight.
33. Finally got my website live at **factorio-production-calculator.com**.

User System

---

34. Repeated the same process from server-block configuration and onwards for my personal website, hosted at **ceofyeast.com**, a much less verbose domain purchased from a much more reputable registrar, Cloudflare.
35. Created a repo containing my server block config files for future reference.
36. Decided to make an unnecessary and self-fellating "retrospective" document on my progress.
37. Began researching Node server deployment options.
38. Implemented the Node server behind NGINX server approach, which uses the NGINX server to host static files and proxy all other requests back to the Node server.
39. Started documentation on Trello board.
40. Daemonized Node server.
41. Got Node servers for FPC and my personal website running concurrently.
42. Created a new development branch for the user system.
43. Used Fastify static plugin to serve static files for my dev environment.
44. Rewrote the modal system to be more robust.
45. Initiated first connection to the database using Fastify plugin.
46. Implemented account creation and access routes.
47. Tested response body + code correctness.
48. Implemented request + response schemas.
49. Tested schema validation.
50. Reworked forms to use Axios, which made paring responses easier

51. Added a store to contain and expose knowledge about the currently logged-in user
52. Converted the website to use HTTPS
53. Released the user system

Planning

---

54. Began prepping for the development of the website as a whole; everything up to this point had mostly just been learning about web dev and prototyping different functionality; some production code was produced though
55. Decided on three development phases; planning phase, backend phase, front end phase
56. During planning phase, I started with pinning down the general requirements of the website
57. Made heavy use of use-case tracing and realization, as well as diagramming; this helped me get more specific with each general requirement; it also helped me to answer my own questions about the website, and come up with new ones
58. Planned out the general architecture and dependencies of the system, and gave names and descriptions to certain concepts such as IRPTU
59. Drafted a thorough requirements sheet to pin down all the specifics; this got as granular as individual functions in each module (with separate modules being views, components, stores, and modules); this step marked the end of the planning phase

Production Calculator

---

60. Started the backend development phase by creating a new development branch to work on the logged-out functionality of the website
61. Completely refactored the calculators “module”, splitting it into multiple different ecmascript modules; I used test-driven development for most of this, and by the end all the code had test coverage
62. Published the refactored calculators code as an NPM package titled **@ceofyeast/prodchaincalculators**; the package exposes its functionality through the “irptu” and “utility” modules
63. Created a testing suite separate from the package, and moved all the unit tests into it
64. Hooked up a client and server system to the testing suite to carry out manual tests
65. Brought the package into the main FPC app, and hooked it up to the loaded factory store
66. Coded a visual interface for the end user to interact with the package

67. Hooked the visual interface up to the store, adding more functionality to the package in the process
68. Tested code thoroughly, and pushed the “Production Calculator” release