Collin Hansen

Web Developer

Contact

Email: **chansen3910@gmail.com**

Phone: **(432) 271-6960**

Profile

Passionate software developer with experience in front-end and back-end technologies and architecture.

Dedicated to creating efficient, extremely smooth, user-friendly, and visually appealing websites which exceed expectations.

Skills

- Amazon Web Services, Google Cloud Platform.
- Podman, Docker containerization.
- React, Angular, Vue, Svelte.
- Web Components API, History API, WebGL, various browser APIs.
- HTML5, CSS3, JavaScript, WebAssembly.
- MariaDB / MySQL (SQL), MongoDB (NOSQL), OracleDB (SQL + PLSQL).
- TomCat, Payara, Spring, Express, Actix, Quiche, Drogon.
- C / C++, C#, Rust, Java, Node.js, Python.
- Make, Maven, Gradle, Jenkins.
- Debian Linux, Ubuntu Linux, Windows.
- Knowledge of Agile methodology, and the Software Development Life Cycle (SDLC).
- UX design with consideration to Swiss design simplicity / hierarchies of color and detail.
- Responsive, cross-browser front-end implementation.
- Image creation, alteration, and optimization using Krita and GIMP.
- Video transcoding / chunking using FFMPEG for HTTP Live Streaming (HLS), supporting Adaptive Bitrate.
- Competent and intentional use of data structures and algorithms.

- Experience with encryption algorithms, JWT, TLS, HTTP/2 (SPDY), HTTP/3 (QUIC), and information security best practices across the full stack.
- 3D modeling and animation in Blender, specifically targeting GLB for graphics on the web.
- Knowledge of legal requirements and the importance of user privacy and the handling of sensitive data.
- Knowledge and experience using Git, Subversion, Version Control.

Projects

Eclectic Hummingbird

A containerized prethreaded Linux socket application server written in C++ which currently supports HTTP/1.1. Designed to cache and serve static resources from a directory efficiently and asynchronously.

On startup, the server socket is configured to listen for edge-triggered events (client connections) in non-blocking (asynchronous) mode.

The cache is then constructed by traversing all subdirectories of a provided relative path (an std::string parameter) to locate and parse resources into readily-available objects (an unordered map of URI / path keys mapped to instances of the custom Resource class).

After the initial construction and configuration stages, the main thread waits for new connections, accepts them to obtain the client socket, sets the socket to asynchronous non-blocking mode, and epoll_waits for the edge-triggered I/O ready state change.

Once the socket is ready for IO processing, it is pushed to the back of a mutex-protected deque, managed by a separate threadpool dedicated to handling the client socket IO processing.

A single available thread is notified when the mutex condition variable flag "queueNotEmpty" is set to true, and the socket is popped from the front for dedicated read / write.

Several classes aid in the tasks of socket configuration, epoll context management, parsing of HTTP requests, creating and sending HTTP responses, and preparing static resources conveniently.

I have several plans to improve the performance and reliability of the server before eventually adding automated encryption key management, connection / performance analytics, cybersecurity attack mitigation, JSON Web Tokens (JWT), and support for more protocols (HTTP/2 w/ TLS, perhaps HTTP/3??!!), per the associated W3C RFC definitions.

NewTube

A containerized video-on-demand web application built in Java on the Payara server platform.

It began as a way for me to share recorded lectures with my classmates in a particularly difficult Networking class with a particularly difficult professor who could not speak English very well.

The school would not allow us to take or share videos of lectures, but this professor was seriously difficult.. So instead of allowing us all to fail, I took matters into my own hands and screen-recorded from zoom in 1080p.

As you might imagine, it was readily and effortlessly logistically feasible to manually share +/- 1.5 hours of 1080p video content after each lecture with everyone in the class, so I researched streaming protocols and ended up on this insane time-and-money-sink rabbit hole of attempting to make my own clandestine version of YouTube...

Due to the cost of hosting, my fear of being caught and thrown in the clinker by the university CS department, and the lack of security (only a TLS wrapper around the domain, no implementation of JWT or other encryption layers), I took the live version of the website down almost immediately after finals and destroyed the database.

The database used efficient stored procedures in MariaDB to access and mutate video metadata and analytics, user data, and comments / replies.

None of the data was ever cached so the users would always receive the most up to date information, which probably also contributed to this project's demise.

The thing is that this was being created outside of class, and my neck was being stuck way out, and very few of my fellow students appreciated the amount of brain effort, time, and money I put into this to make it so easy and available for them.

The video transcoding scripts are built upon FFMPEG to take in videos of either 1080p or 720p resolution of the .MP4 container format, transcode them into H264 encoding, down grade the resolutions to support adaptive bitrate (1080, 720, 480, and 240), and finally chunk each resolution into 3 second .TS containers.

The .M3U8 and video thumbnails are also automatically generated by this makefile script, although sometimes they had to be changed manually.

I initially committed to strenuously uploading the packaged / chunked HLS resources to S3 storage manually until I finally got smart, and started using service agents to do the uploading for me.

F3!.js

A web componentized scene manager front-end framework extended by a simple web component which stands on the shoulders of Three.js to make it easier to manage immersive 3D scenes on the browser in an efficient and asynchronous manner.

Also includes a dev tool web service called Map Maker, which processes uploaded static map models in the GLB file format into JavaScript files containing an octet map instantiation, where values are objects with members that hash into categorized triangle collections ("walls", "floors", and "ceilings" - using the y-component normals).

You'd be correct if you say this algorithm is ravenous for memory, but the resulting map massively improves the performance of subsequent collision detection algorithms.

The produced octet map instatiation allows for O(1) lookup access to ONLY the triangles of the static mesh which are in the immediate vicinity of the calling subject in real time.

The parameters provided when the map is created allow developers to control the granularity (number of indices within the 3D space boundaries, the accuracy / closeness of the parametric equations used to determine if a given triangle is actually in or just grazing another unique indice).

Other features of this componentized framework are automatic memory cleanup, simplified asynchronous asset loading, scenic transitions in HTML / CSS / JS, simplified I/O event listening, configuration, and cleanup, as well as a simple ui component / context manager.

This one is still being cleaned up and streamlined, but I wish to someday demonstrate the O(1) collision detection and eventually make a map loader / coordination module for dynamic maps (which mutate as a response to events or morph mesh targets on embedded animations).

Study Buddy

This application demonstrates the MERN stack (MongoDB, Express.js, React.js, Node.js).

Study Buddy is a study application which incentivizes learning by taking quizzes with virtual assets, badges, and a condescending, materialistic owl mascot.

The game also runs on server time. When a user creates their account, the region determines the server time.

The owl modulates his sarcastic tone accordingly and the player may find certain in-game facilities only available during certain hours.

NutriRDS

This application demonstrates the MEAN stack (MongoDB, Express.js, Angular.js, Node.js).

This is a web application where you can bring your own dietary requirements and build your diet plans with assisted coordination.

Spotlight Operator

A community-wide improvised role-playing game developed using Svelte.js, served by middleware written in Rustlang using quiche, actix, and MySQL.

Users are fed prompts which they are instructed to reply to with a comment or video response.

At the end of the cycle, the chains of replies are rated by another set of users. The winners are rewarded with credits and the game begins again. Player hiscores are tallied on the site.

Thank you for your time.