Wesley’s Ultimate Guide to Coding w4113

# w4113 Software Stack

## Languages

* TypeScript — Frontend language. TypeScript is for the most part equivalent to JavaScript and just has a few major differences. Anything that works in JS will work in TS, but not necessarily vice versa.
* React — Not a language per se, but you can definitely think of it as one. React completely changes how JavaScript interacts with the webpage. Instead of having a separate HTML and JS script, they are combined into a single .tsx file that combines the elements from both. Everything that touches the frontend is built in React.
* CSS — Regular CSS, used to stylize webpages.
* Rust — Crazy fast, crazy safe functional systems language that is highly versatile. This is what we use for the backend and for all audio-related functions.

## Frameworks/Tools

* Yarn/Npm — Package management software for the frontend portion of the site. Yarn is supposedly better, don’t ask me why because I don’t know the difference and they install to the same place anyway.
* Cargo — Package management for the Rust backend.
* Vite — Build system. This is what automatically updates the frontend when TS/CSS files are saved and recompiles Rust automatically when a Rust source file is changed.
* Tauri — Tauri is what packages everything together and connects the frontend and the backend, as well as performing all the window and system functions in the background required to make an application work.

## Libraries

* Tauri — Tauri, obviously.
* Serde — **Ser**de is a Serializer and **de**serializer that is used in a ton of Rust applications. Basically what Serde does is transform any Rust struct/enum into a JSON object, which can natively be processed by TypeScript. This is what lets us sent objects directly to and from the backend.
* Serde\_json — Adds JSON functions for Rust onto Serde.
* Cpal — Primary audio library that we will be using.
* Fundsp — I’m not really sure yet but it has some sort of audio processing capabilities that we may use.
* Log — Used for Rust logging to the terminal.
* Tauri-plugin-log — Allows the frontend to access the logging functions.
* TS-RS — Used to generate TypeScript bindings for Rust objects; basically clones the struct/enum definitions in TypeScript so that everything can be made more easily compatible.

# How to Code ItScreenshot 2023-10-25 at 7.33.25 PM.png

## Filetree Explanation

Some of the file tree is fairly obvious what it means but it is not super clear so I want to explain.

* **/** (basically the entire w4113 folder) is a Yarn/TypeScript project. All of the configuration files in this part are either for the frontend, Yarn, Vite, or GitHub (since this is also the root of the Git repo).
* Since **/** is a Yarn/TS project, **/src** is the source code for the frontend, thus all being TypeScript/CSS files.
* **/src-tauri** is basically a separate Cargo project contained within the project. This is where all of the Rust config files go as well as where the Tauri configuration lives.
* **/src-tauri/src** is consequently where Rust source code files live.
* **/src-tauri/target** is where the compiled binaries go. Nothing in this path should be modified manually
* **/src-tauri/icons** is generated by Tauri automatically and should not be manually modified. Use yarn tauri icon if you need to generate icons.
* **/public** is where assets like images are. These are accessible from the frontend. For example, **/public/image.png** would be accessible just as “image.png”
* **/src/bindings** is where the TypeScript bindings for Rust objects will go.
* **/node\_modules** is managed by Yarn/Npm and shouldn’t be manually modified.
* **/.vscode** is VSCode configuration.