SE575 - Final Project Deliverable

Blockchain Demonstration Simulator

Project Team Members

1. Blake Nguyen
2. Trevor Schirra
3. Chaewon Yun

Demonstration Video Link: https://youtu.be/5o1NCRC--94

Description:

A blockchain demo, completely deployable to k8s using a containerized (pod) architecture to package the front and backend as a single portable unit. This demo focuses on concurrency at many levels, from source code concurrency in the proof of work for speed, to the stability offered by managing pods through Kubernetes.

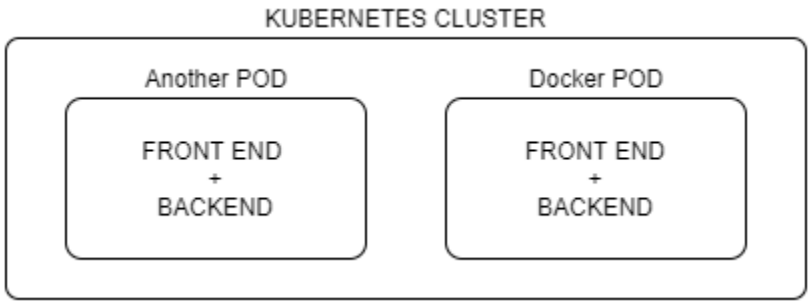
1. Standalone modern front-end application built using HTML, CSS, JavaScript, and jQuery and showcasing design attributes such as web components and reactive behaviors.
2. Implemented HTML REQUEST to pass Block data, Preference changes, and any other commands from a user to Rust.
3. Used a JSON file to transfer data from rust to the front-end application.
4. Highly parallelized mining approach mimicking the workings of Go to minimize mining time.

Project Code Location: https://github.com/TPXS-Drexel/bc-rust/tree/main

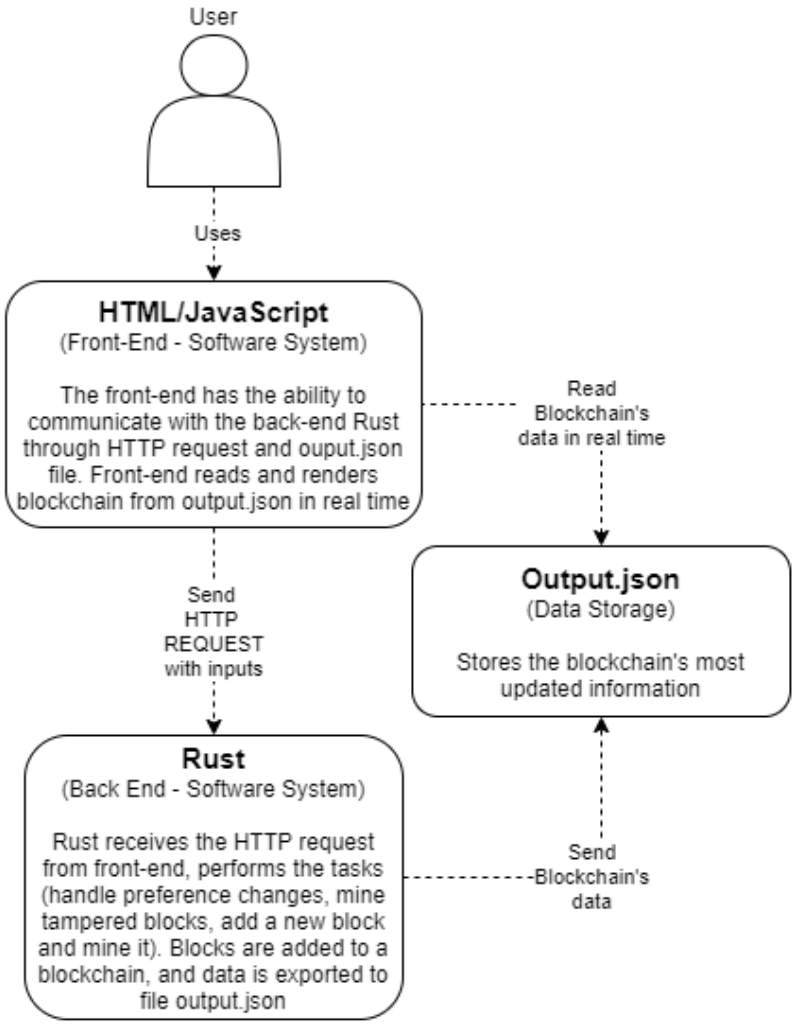
Project Build Information: See readme section from the GitHub repo:

https://github.com/TPXS-Drexel/bc-rust/blob/main/README.md

High Level Design Overview



Addition pods/containers can be deployed as requests increase.



The picture above is the overall architecture of how front-end and back-end connect. There are four main components in the system: User, front-end software system using HTML and JavaScript, back-end software system Rust, and a JSON file.

The front-end will display all the information for the user to view and make changes if needed. The user can add a new block to the blockchain system, change the Proof-of-Work complexity, change the number of attempts, tamper with the block's data, and then mine it again to see new changes occur to other following blocks. All user interactions to the front-end will be sent back to Rust back-end via HTTP requests. Besides that, the front-end will always read the output.json file asynchronously to keep on displaying the most recent updated blockchain.

The back-end Rust will accept HTTP requests from the front-end along with user's inputs (such as block's input data, changes in preferences, mine again request with block's new input data). Rust back-end will then update all of the constant variables if requested, mine a new block using user input with the previous block's information, or re-mine a whole chain if the user tampered with one of the blocks. The back-end will always overwrite the output.json file as soon as any changes

are made to the blockchain system. This will help to ensure that ouput.json will store the most updated blockchain.

Finally, the output.json file will act as a mini database to store the blockchain data. It will be deleted when the application is turned on to avoid showing the old blockchain data, and then it will be created immediately with an initial default block. The output.json does not interact with the system. The back-end stores data in the output.json file so that the front-end can read and render the blockchain.