Community Board

Brooke Leinberger (bl2060@rit.edu)
Tristen Kurutz (cdk6546@rit.edu)

SWEN-342

5/5/2025

Overview:

A Java application that serves as an intranet with the ability to be easily locally hosted for a community without outside connectivity. Includes a C project demo that demonstrates how data can be collected and sent to the board for communication to users.

Example: Community Garden

Micheal is running the community garden and wants to monitor soil health. He wants regular updates from deployed sensors to track weather and soil data (wind speed, sunlight, temperature, soil moisture). He registers for continuous monitoring, with updates being saved to his local machine every hour.

Sandra is growing tomatoes. She's worried about the intense wind in the area recently and wants to know if she needs to protect her crop. She registered to get email notifications when the detected wind speed exceeds 50 mph. She's not very technical

Areas of Focus:

Tristen: Community Board Java application functionality. Specifically User to Board and Administrator to Board.
Brooke: Sensor to Board C application functionality. Email notification stub.

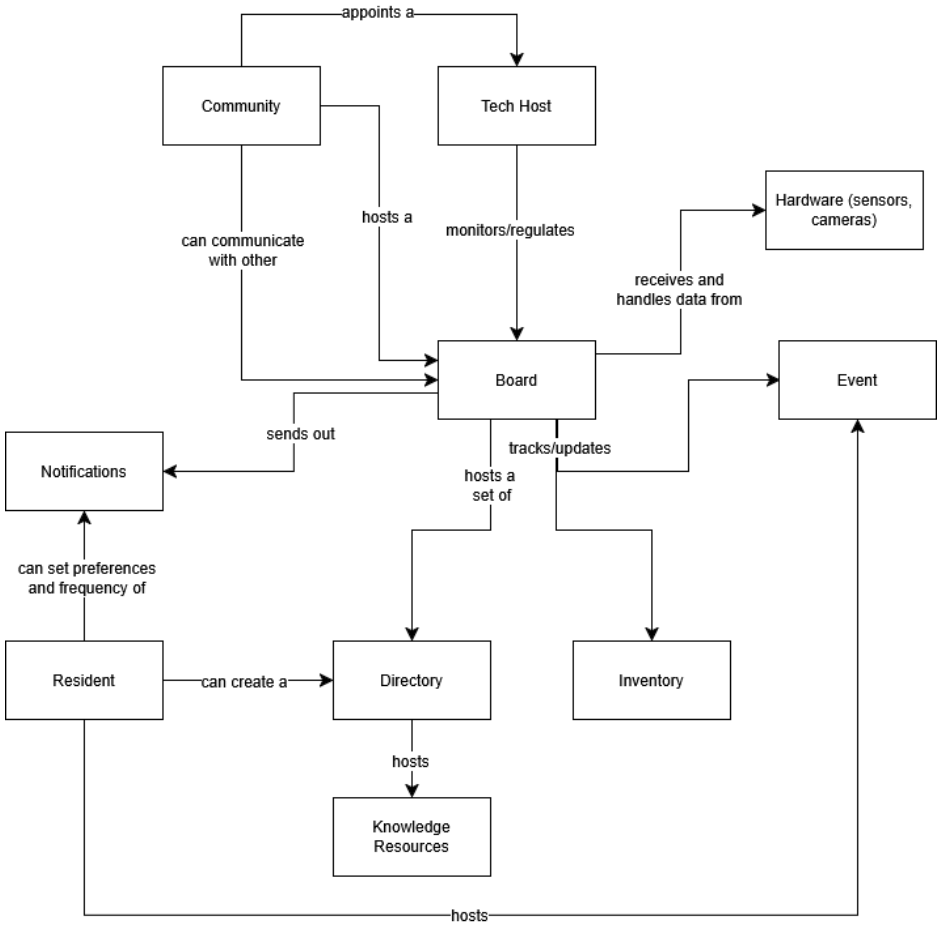
Analysis / Design:

DEMO
2 applications
1 in C (remote sensor)
- Set configurations to something different (MVP)
    - E.G. change schedule of a IoT sprinkler
    - Set update frequency of a sensor
- Collecting data

1 in Java (community board)
- FTP communication (MVP)
    - Syncing between communities
    - Retrieving requested information
- Sensor communication (MVP)
    - Sending out notifications to subscribers at user-set frequency
- Event notification (MVP)
- DNS server (Not MVP)
- Community web pages (Not MVP; doable through FTP)

3 API Sections

- Client <-> Board
    - Someone else in the community, using the network to connect to the Board
- Board <-> Board
- Sensor <-> Board
- [Link to API documentation]



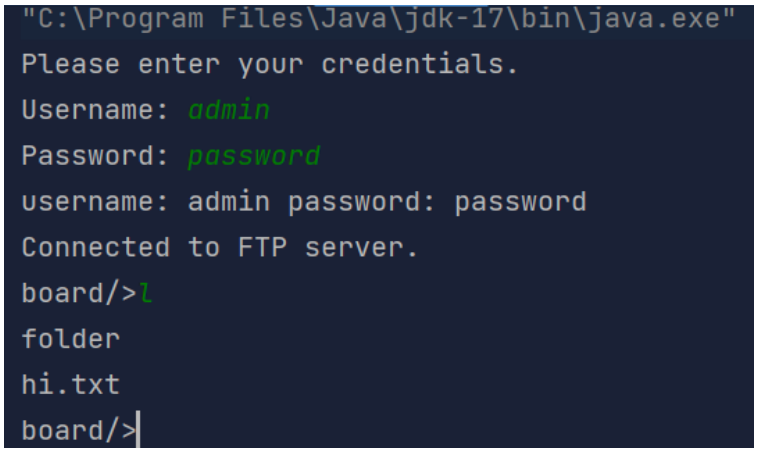
Domain Model

Project Results:

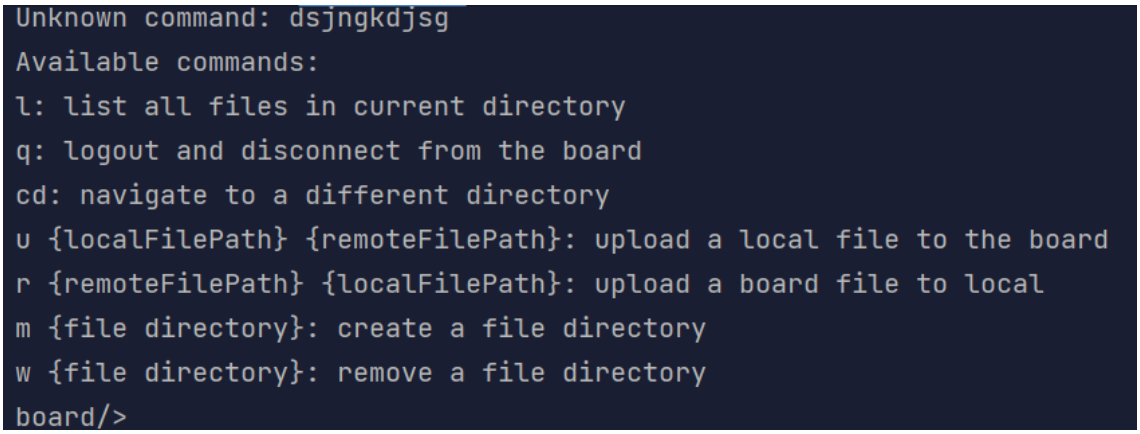
CommunityBoard Java Application

Below are a few screenshots of the Java application that represents the CLI interaction with the Board server. A user that is using the CLI application to communicate with the Community Board can list files in the remote directory, navigate through directories, delete files, delete directories, make directories, and quit. For the upload commands, the BoardCLI sends an HTTP POST request, with the action and username in the body, to the server inquiring whether a user has permission to execute those actions depending on their group's permissions in that specific directory. Additionally, groups are locally cached in the program through a hashmap, and that hashmap is saved to a config file so groups are consistent between restarts.

We were expecting to also implement a Board to Board communication API, but unfortunately with time constraints we never got to it. If this project were continued, Boards would be able to send messages to each other and also sync files through FTP.



Login, list directories and files.



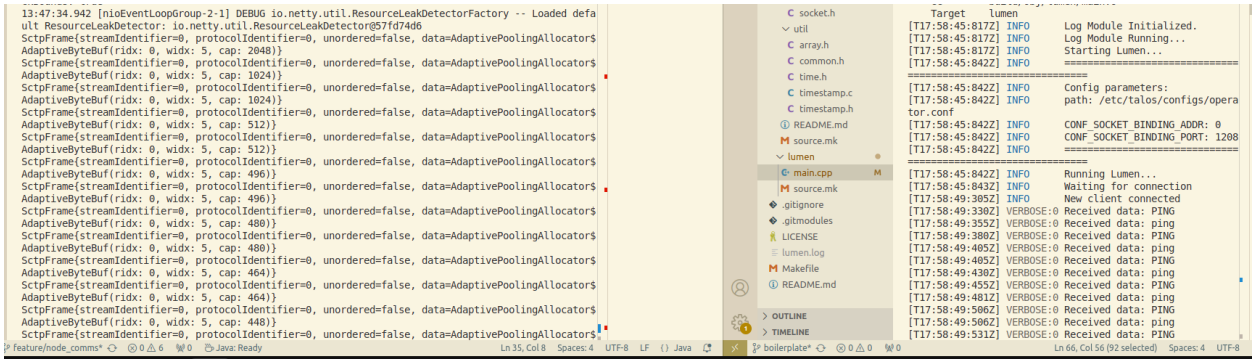
Available Commands

```
Please enter your credentials.
Username: admin
Password: password
username: admin password: password
Connected to FTP server.
board/>u "C:\Users\[redacted]\Downloads\pthreads" "pthreads"
OK
REPLY: 226 Transfer complete.

File uploaded successfully.
```

FTP Transfer (Local to Remote)

Node C Application



Ping Example

Above is a screen capture of a “ping” example between the two processes. The board initiates communication by sending an arbitrary ascii message (in this case, “PING”). The node responds by inverting the case of every letter in the message, before sending it back. The board, upon receiving a message, will immediately echo it back to the node, and the cycle repeats.

This admittedly crude example shows the expected dataflow from within the Node. A message comes through the socket, where the socket thread copies it into a *Message* buffer, and queues it for the node’s other threads to process and return to the Board. Due to expected time constraints, and unexpected implementation hurdles, this example proved the most concise to demonstrate the communication functionality while meeting our deadlines.

Extending this functionality to accomplish the original goals of this project is relatively straightforward. The Socket interface includes a public definition for how messages are defined, as well as methods to check how many messages are waiting, retrieving waiting messages, and sending responses. These methods internally manage locks to avoid race conditions and deadlocks.

Extending on the board side is less straightforward, but still manageable as the Netty SCTP client implementation used for this demo has extensive examples and boilerplate to integrate custom functionality.

Lessons Learned/Reflection:

Tristen:

- This is one of the most in-depth projects that I've done at RIT thus far. It was definitely a bit out of my comfort zone as I've never worked with FTP protocols before. Additionally, Jackson's libraries are poorly documented or not documented at all, so it was a learning curve to implement our application with this library. I also had not done much Java work, specifically with HTTP requests, since the Summer of my freshman year two years ago. I definitely learned a hard lesson of doing more research of libraries before fully integrating them into our system. If I had realized that the Jackson FTPServer library was so poorly documented, I would have tried to find something else to use. Additionally, I found out that I wasn't as familiar with HTTP requests as I thought, I was apparently very rusty on how they worked, specifically with how response bodies are constructed and how to start a HTTP server instance on our Board. Lastly, despite how much this is exaggerated through the Software Engineering degree, we really should have come up with a more rigid system architecture before starting to program. I feel like I was mostly coming up with everything on the fly and led to my code being decently unorganized. I should have created more Service/Handler/Controller style classes to keep the code more organized. Both the Board and BoardCLI classes are very bloated and if this project were to continue, they are in desperate need of being refactored. Lastly, my Java files aren't properly documented, and aren't completely consistent with the docs we made in the repository due to changes; this would need to be completed in future iterations to ensure maintainability and extensibility. This project was really cool, and I wish that I was in the right mental space to completely immerse myself in it - during our final implementations in our project I was going through a lot of personal issues.

Brooke

- I took this project as an opportunity to refine my skills and experience in distributed, multithreaded systems. Having done similar projects on more than one occasion, I didn't fully consider the importance of strictly adhering to process principles anyway, such as through documentation of design choices (more UML/Architecture docs), generous time allocation (there *will* be a bug out to claim your sanity), and risk analysis (failing to foresee how busy the last 3 weeks before graduation would be, not assessing different dependencies early in the design phase).
- This project also gave me an opportunity to try new technologies and techniques that I didn't in other projects.
    - SCTP client/server implementation
        - Easier message handling
        - Absolute pain to compile correctly
    - Lighter-weight socket implementation than used in previous projects
        - More performance

        - Easier to debug
    - More care and thought around threading resources such as locks, and buffers
        - Fewer head-shaped dents in my favorite brick wall