Software Implementation and Testing Document

For

Group 21

Version 1.0

Authors:
Irfan Y
Gabriel B
Leonardo N

1. Programming Languages (5 points)

Golang - we chose this language because of it's inherent concurrency and also it's highly performant compiled nature. We use it on the backend for building out our HTTP server infrastructure.

NextJS - we use this for the front end, and for our all our UI. We personally chose NextJS due to its ease of use, quick development time, and also server side rendering capabilities. Our UI is not going to have many dynamic components, and will leverage the WebRTC protocol, therefore it is important to take as much compute off of the client as possible.

React - we chose React because of how popular it is and previous experience with it. It also is very easy to refactor and works across platforms very well in a web browser.

TypeScript - we went with TypeScript over plain JavaScript for type safety and a more clean scalable codebase. It also allows us to have types and specific data types later on when we build out our file upload and WebRTC file transfer functionality.

HTML + CSS - these are foundational and had to be used. It is an easy markup language for building and styling a UI.

2. Platforms, APIs, Databases, and other technologies used (5 points)

Currently we are using NextJS as the framework/platform for the front-end.

We are going to leverage WebRTC for an encrypted client to client data transfer.

We also are using Digital Ocean for hosting our PostgreSQL instance and HTTP server.

We are likely to move our front-end to Digital Ocean as well to benefit from the reduced latency of moving the frontend closer to the backend.

We are also using AWS S3 Buckets for storing files that users will be uploading.

3. Execution-based Functional Testing (10 points)

We had each member of our team go through the functional requirements on the RD and test that they were working as intended. This is limited to only the functional requirements we have implemented so far however.

4. Execution-based Non-Functional Testing (10 points)

We did test for non-functional requirements by timing how quick pages would load, and how quick responses could be sent back. We also made sure all of our endpoints and hosted databases were quick in their response and not experiencing error.

5. Non-Execution-based Testing (10 points)

We made sure all of our code was syntactically correct and easy to understand/read. After a group review of our code base, we even went ahead and refactored the entire skeleton of the backend to make it easier to develop on and scale overtime.