Functional and nonfunctional requirements for collaboration tool

ELEC-E7320 - Internet Protocols D - 16. 02. 2024

Aarni Haapaniemi, Zong-Rong Yang and Prerna Gupta

Outline

Requirements

Functional requirements

- Create the document
- Modify the document
- Delete the document
- Validate Collaborators
- Saving the document (or anything else)

Use case name:Create a document

Actors involved: Client, server

Preconditions:

Active connection between server and client

Steps:

The client generates a new public-key encryption keypair (sk, pk)

Client sends a “Create doc” request to the server + their public key pk

Server responds by generating a random challenge using pk

Client decrypts the challenge with their secret key sk

Server checks if the decryption looks ok

Use case name:Create a document (ii)

Error:

Decrypted challenge does not match

Server ignores “Create” request

Post-conditions:

A document is created by the server

The client now has access to the document via a token

The client is now called the “Admin” of the document because they own sk

Use case name: Modify a document

Actors involved: Client, server

Preconditions:

Active connection between server and client

Client has the token to access the document

Steps:

The client A modify the content in the document, and for that client it triggers local updated response on the document.

Server receive the modifying request from client and validate the identify of the client.

Server transmit the modifying content to all other clients online.

Ohter clients(client B,C,D…) receive the modifying content from server.

The changes will show on other client's document page.

Use case name: Modify a document (ii)

Error:

Server ignores the modifying action.

Client's token cannot be varified.

Online client losts the connection with Server.

Post-conditions:

All clients now have the same contents on the document page.

Use case name: Deleting a document

Actors involved:

administrator and server

Preconditions:

Active connection between server and administrator

Client has the token to access the document

Steps:

Inform collaborators,It is crucial to communicate the removal of a document to ensure that others are aware of the changes.

Only Admin will be able to delete the document.

Admin will instruct the server to delete and then server will validate and confirm that the request is from authorized admin or not.

Use case name: Deleting a document (ii)

Server will take action accordingly once it has verified the information provided above.

The changes will show on main tool page.

Error:

Server ignores the deleting action.

Administration losts the connection with Server while deleting the document.

Post-conditions:

All clients will be able to notice that the document is no longer there after logging into the tool.

The document will be permanently deleted, recovery might be impossible.

Non-functional requirements

- The system should be secure
- It should feel realtime
- UI should be simple and not bloated
- Way to test the app programatically
- App should be able to support multiple collaborators, e.g. up to 10
- The final implementation will likely be limited by server power because app will run on a laptop
- Compare our app with google docs

A draft of the architecture design

Protocol Information:

Our system employs Websocket Secure (WSS) as a protocol to provide a secure communication channel, which works on top of TLS.

Service Description:

The tool delivers real-time editing capabilities, allowing multiple users to collaboratively edit documents. Anyone can create a new document, and anyone can join a document as long as they know the token assigned to it. Documents are stored in a centralised database, meaning that they can be accesses after a session has ended.

Security Measures:

Our tool prioritizes security with Websocket Secure, PKE and token-based authentication, ensuring confidential admin access and secure collaboration.

Environment Assumptions:

Although we use WSS, our authentication method for Admin do not rely on there being an encrypted connection between the Admin and the Server, as it uses PKE for authentication. However, we do not assume that each client is in a trusted network, and as such we want to prevent un-authorised parties from reading sensitive data such as the tokens or the data sent to the document, which we achieve by using WSS.

Network stack

Application Layer:

WebSocket Secure (WSS): At the top of the stack is the WebSocket Secure protocol, which we use for communication between the client and server.

Transport Layer:

WSS is sent over a TLS session, and thus we will use this in the transport layer.

Network Layer:

Internet Protocol (IP): The network layer involves IP, responsible for routing data packets between the client and server. IP ensures the delivery of encrypted WebSocket data packets over the internet.

Schedule moving forwards

Current status

- We have a proof-of-concept app that utilises websockets for realtime group messaging

Future goals

- Public-key encryption for authentication
- Some sort of database for saving each document and metadata about it
- Ability to edit messages sent to the server
- Some sort of consideration for race conditions?