Project 1 distributed computing systems

Made by Cakston Calvin & Lars Vonk

Introduction

In this project we have chosen the assignment to create a “Message exchange system with encryption”. We have chosen this assignment since it aligns well with current interests.

We will try to create a basic application where multiple clients will be able to exchange messages in an encrypted way between client and server and where using web sockets the updated messages will be shown live to all the users.

Functionality

This system aims to create a fault-tolerant, reliable and secure system to exchange messages between users. The fault-tolerance and reliability will be provided by running multiple instances of the server, persisting the data on the host system which runs the orchestration where the database node can fetch the initial data on restart to obtain the current state that it had before the crash.

The security of the system will be done using the classic end-to-end encryption using the assymetric encryption algorithm RSA (Ron Rivest, Adi Shamir and Len Aldeman) to generate a public key for encryption and a private key for decryption. Each of the systems will do this (the client and the server). Then exchange their public key for encryption before sending data, and using their private key to decrypt the data after arrival of encrypted data.

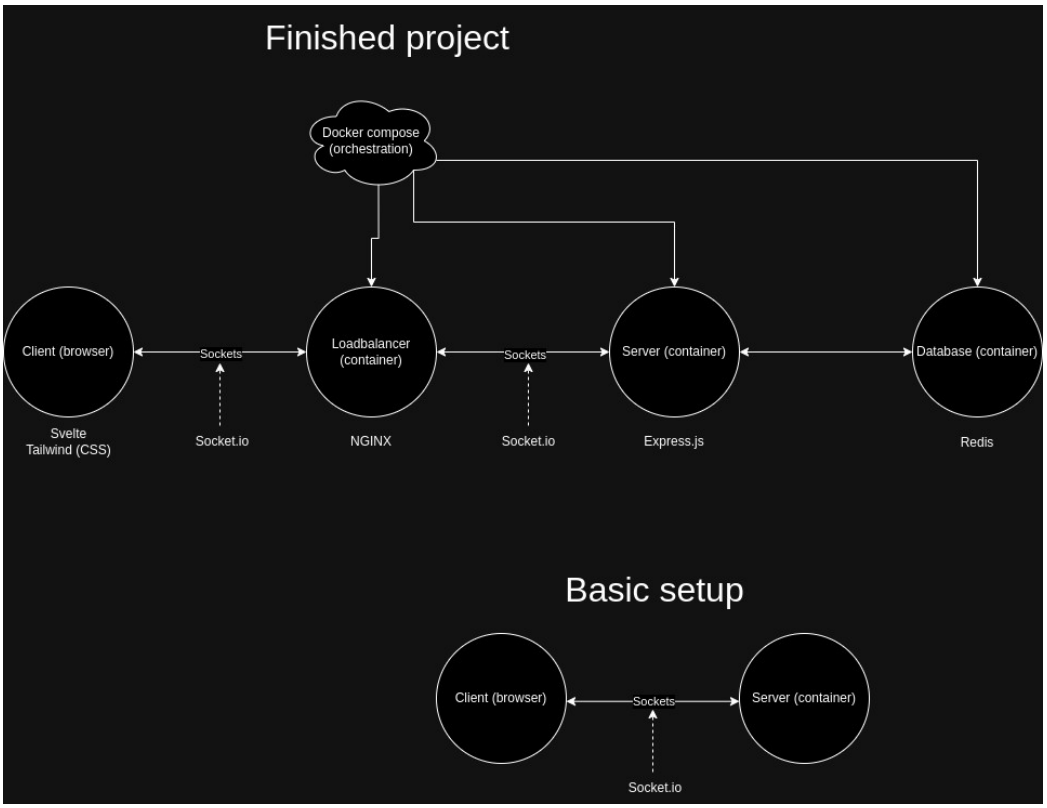
Protocols

Our system will communicate using HTTP to establish the handshake between the client and the server and TCP to have a long-lived connection between the client and server.

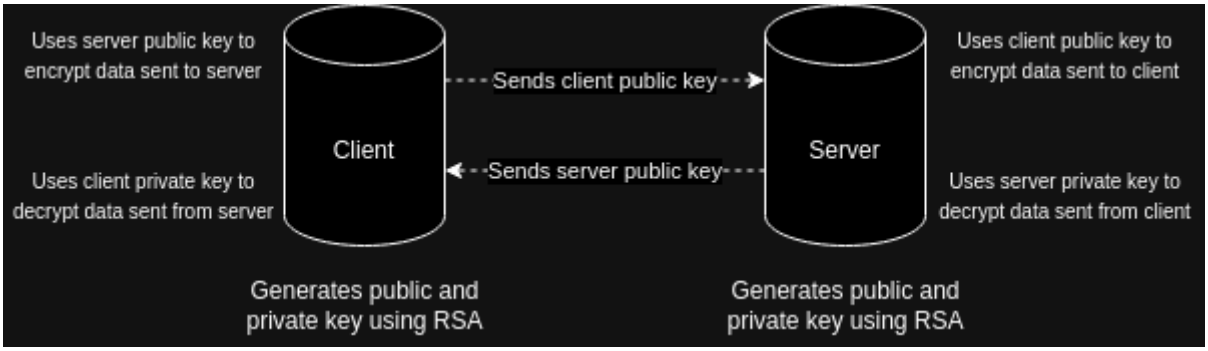
The server and database will also connect using TCP and exchange data over this connection.

Design

For this project we have made two initial designs. One for how the network of systems will work. We will be using docker compose for the orchestration of the systems which will exist of 3 parts that are hosted by the orchestration tool and one which is hosted by the client itself. These are the client, the proxy, the server and the database. The server will be replicated and the proxy will be configured to sent requests to the correct server in the cluster.



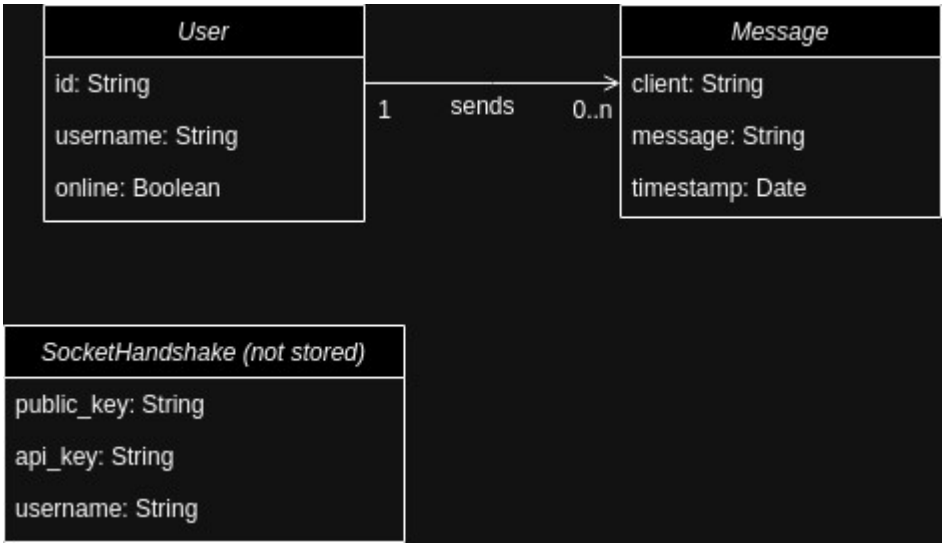
Secondly we have made a simple design how our encryption will work. We have decided to use the most common way of end to end encryption where both parties will create their own public and private key and exchange public keys. These can then be used to encrypt messages to be sent to the other user and decrypted. We will make this work in a group chat where all data will be encrypted in transit and to be decrypted at arrival.



For this project we will be using Svelte for the client, NGINX for the proxy, Express.js for the server and redis for the database. These have been chosen for their simplicity of use but also their flexiblity for our use case. Redis has specifically been chosen to also try it out since SQLite would have also been a good choice and has been considered.

Class diagrams

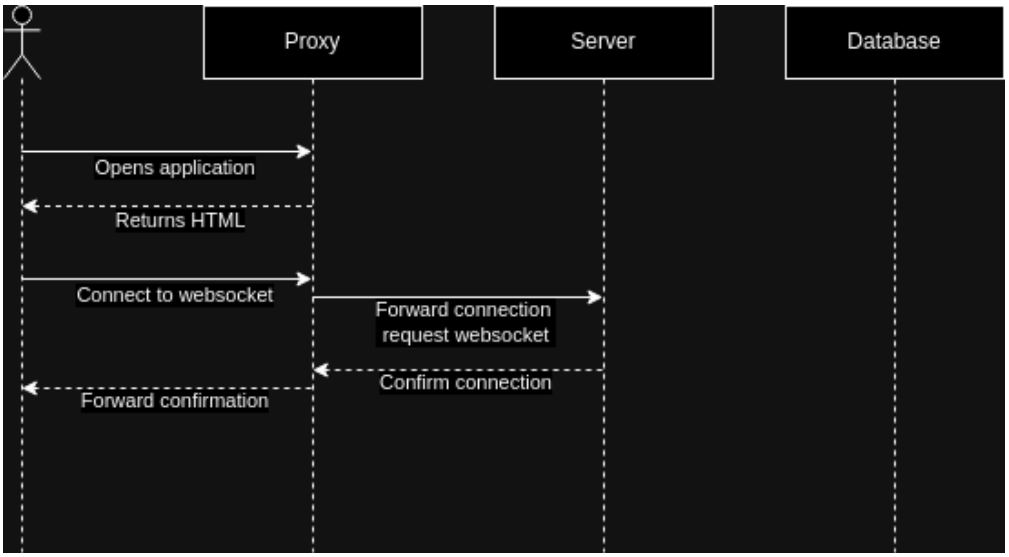
In the class diagram it can be observed that our data structure will be quite easy. We will have a User data object and a Message data object. The other data will be not stored in the database but will live in the websocket handshake (I also modelled this for convience).



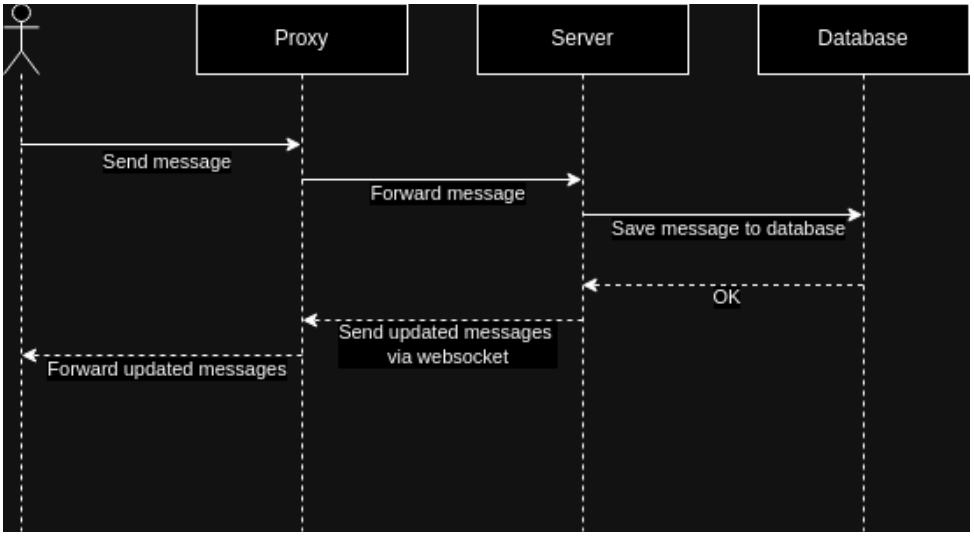
Sequence diagrams

To draw some of the steps that the system performs on some actions.

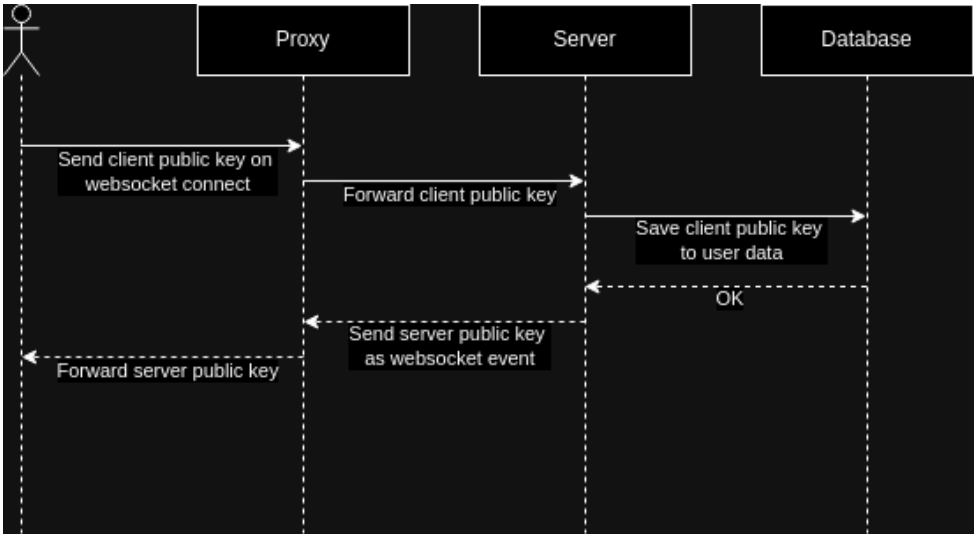
Websocket connect



Send message over websocket



Key exchange for end-to-end encryption



Methodology

This project will be run on systems using Windows 11 and Debian to ensure the quality over different base operating systems (Unix and non-Unix).

Like described in the design we created the components of this project using the following frameworks and libraries.

Programming languages

We used Javascript and node.js for this project. Node.js is javascript but run in a runtime and not in the browser.

Libraries and tools

We used the following libraries and tools:

- Svelte was used to create the client (and tailwind for the styling).
- NGINX was used to create the proxy
- Express.js was used to create the server
- Redis was used to create the database
- Docker was used to run each component in a containerised environment
- Docker compose was used to orchestrate multiple docker containers
- Socket.io was used for communication using web sockets between the server and the client

For the creation of the encryption we used the basic included libraries offered by node.js and the browser (node-crypto and Web Crypto API).

Test plan

To test if everything works well we will use the following tests:

- Check if new node starts and retrieves the correct data
- Check if a new node is started on node crash
- Check if all users get the updates from the websockets when new message is sent (users connected to different servers in the cluster)
- Check if all data is encrypted when sent
- Check if data is successfully retrieved on database crash
- Check if users are successfully routed to another replica on server crash

Conclusion

With some modern tools it should be quite easy and intuitive to run systems using replications. This is important since some of the use cases that occur nowadays require scale and especially in a dynamic way. Tools like Microsoft Azure and Amazon Web Services have made running these kind of solutions in the cloud even easier.

The challenge within this project will be synchronising all the data between the servers so it keeps being up to date whenever an update occurs.