

EECS4222

Distributed Computing Systems

Project 2: Distributed Chat Application

Description of System Architecture

I have deployed a distributed chat-application using Go. I have extended the functionality of the example chatapp provided on Github.

Technologies Used: Websocket for real-time communication, Redis for storing messages, JavaScript, HTML, CSS for frontend, nginx for frontend configurations, Go-language for backend implementation, docker for deploying it to Azure.

Frontend

Deployed on Port: 30222

I have deployed 4 html pages:

- User login Page - login.html
- Registration Page - signup.html
- Messaging Page - index.html
- Guest view Page - guest.html

I have used nginx as a reverse proxy server to handle the websocket requests between the frontend and backend.

Backend

Deployed on Port: 14222

I have deployed my backend server of the chatapp using Go language. It is used to handle the incoming websocket requests, user authentication and registration. Redis is used to store the messages exchanged on the chat application. It is deployed on port 6379.

Design of Components

- I have used 2 handlers sendMessageHandler and websocketConnectionHandler. They are used to handle the messaging feature of the chat application.
- Nginx is used to handle proxy requests to the backend.
- I have used gorilla websocket and redis version 9 for implementing the application. I have also used the cors package for handling the CORS policy.
- The frontend has been deployed using the example chat app.
- The login page and signup page has been designed using CSS, JavaScript and HTML.

How the components interact with each other

- The Message page (index.html) lets you send messages in the real time using Websocket connections.

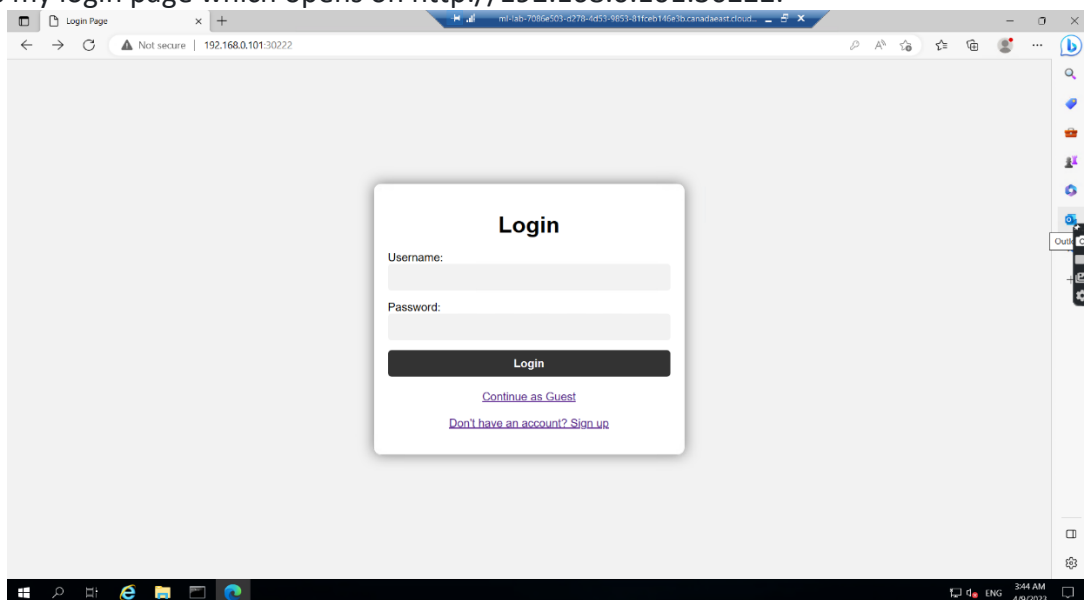
- The Nginx proxies requests to the backend server and the backend upgrades the HTTP requests to Websocket requests.
- The backend stores the messages on the Redis Channel. When a user sends a message, a HTTP POST message is sent to the backend, where it is first encoded to JSON format and stored on the Redis Channel. The Redis INCR function generates a message ID for each message sent and is sent to the Redis Channel "messages".
- Websocket connection reads the messages stored on the Redis Channel and sends it to the frontend. The LRange operation of the Redis Channel is used to extract the last 50 messages from the Redis database.

How you implemented the additional feature

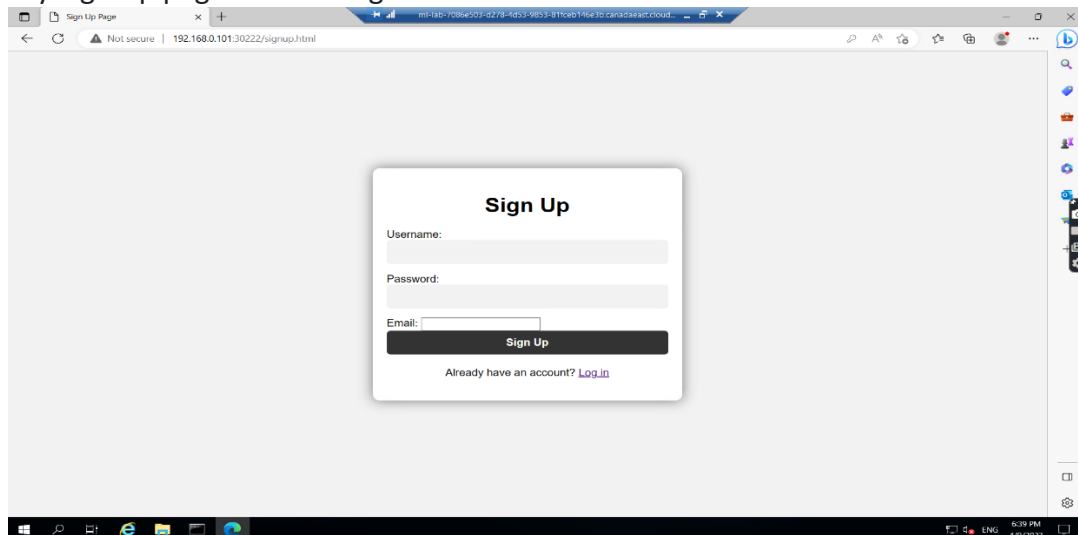
Feature Implemented: Feature 1 (User Authentication and Registration)

- The user logs in through the Login Page which sends a POST request to the backend to authenticate its credentials. For the purpose of testing, I added 3 users to a list of users in the backend server. The backend server authenticates the credentials on the login page. If the username and password is correct, it takes you to the index page. If the username or password is incorrect, it throws an error "Invalid username or password". The users are:
 - Username: "user1", Password: "password"
 - Username: "user2", Password: "password123"
 - Username: "Sejal", Password: "testpassword"
- There is an option to Continue as Guest which takes you to the guest page but you cannot send messages over there.
- A user registration feature has also been implemented. After registering, the sign-up page takes you to the index page directly and adds the newly registered user to the list of users implemented in the backend.

This is my login page which opens on <http://192.168.0.101:30222>.



This is my Sign-up page for user registration.



Sign Up

Username:

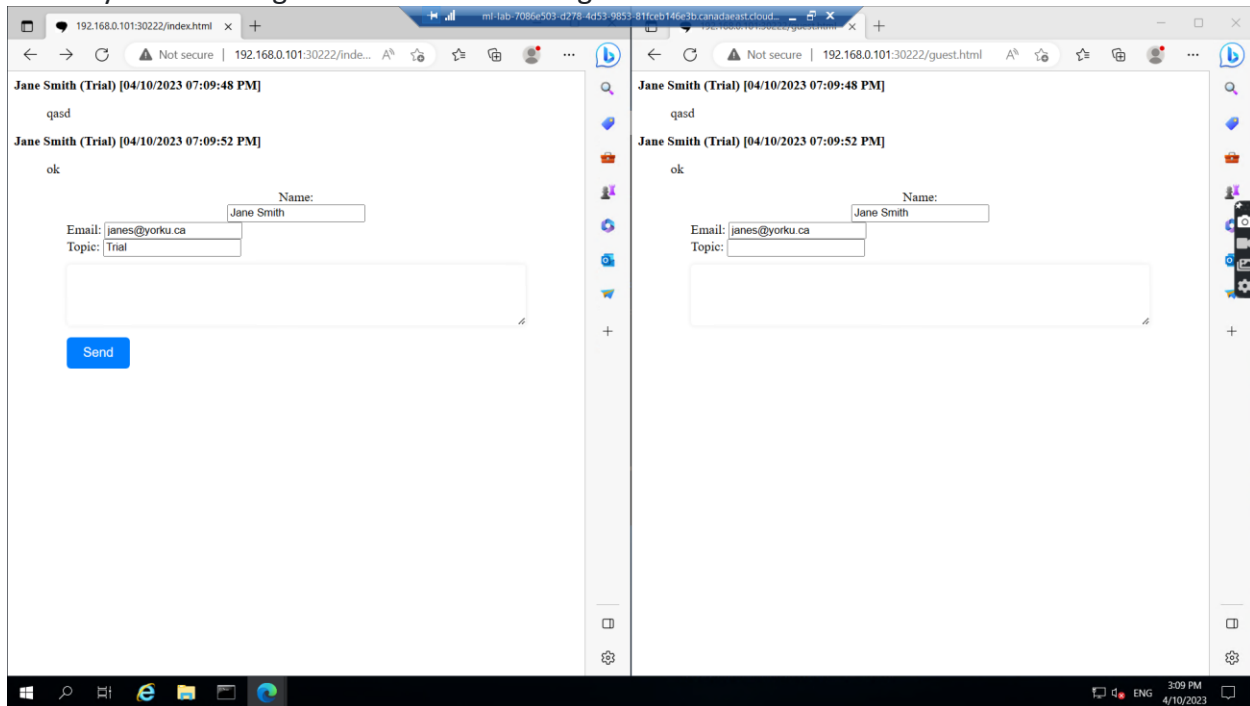
Password:

Email:

Sign Up

Already have an account? [Log in](#)

This is my registered user login and guest login page. It does not have the send button as guests can only view messages but not send messages.



Jane Smith (Trial) [04/10/2023 07:09:48 PM]

qasd

Jane Smith (Trial) [04/10/2023 07:09:52 PM]

ok

Name:

Email:

Topic:

Send

References

- Matthew Ife. (2018, Aug 29). Proxy websockets and HTTP through the same location in nginx. Server Fault. <https://serverfault.com/questions/910805/proxy-websockets-and-http-through-the-same-location-in-nginx>

- Joshua Zhu. (2013, Nov 6). WebSocket support in NGINX. nginx.com.
<https://www.nginx.com/blog/websocket-nginx/>
- James Shewey. (2017, Jan 4). What's the difference between ClusterIP, NodePort and LoadBalancer service types in Kubernetes? Stack Overflow.
<https://stackoverflow.com/questions/41509439/whats-the-difference-between-clusterip-nodeport-and-loadbalancer-service-types>

Feature Testing

User Authentication

Backend stores a list of users for testing:

Username	Password
user1	password
user2	password123
Sejal	testpassword

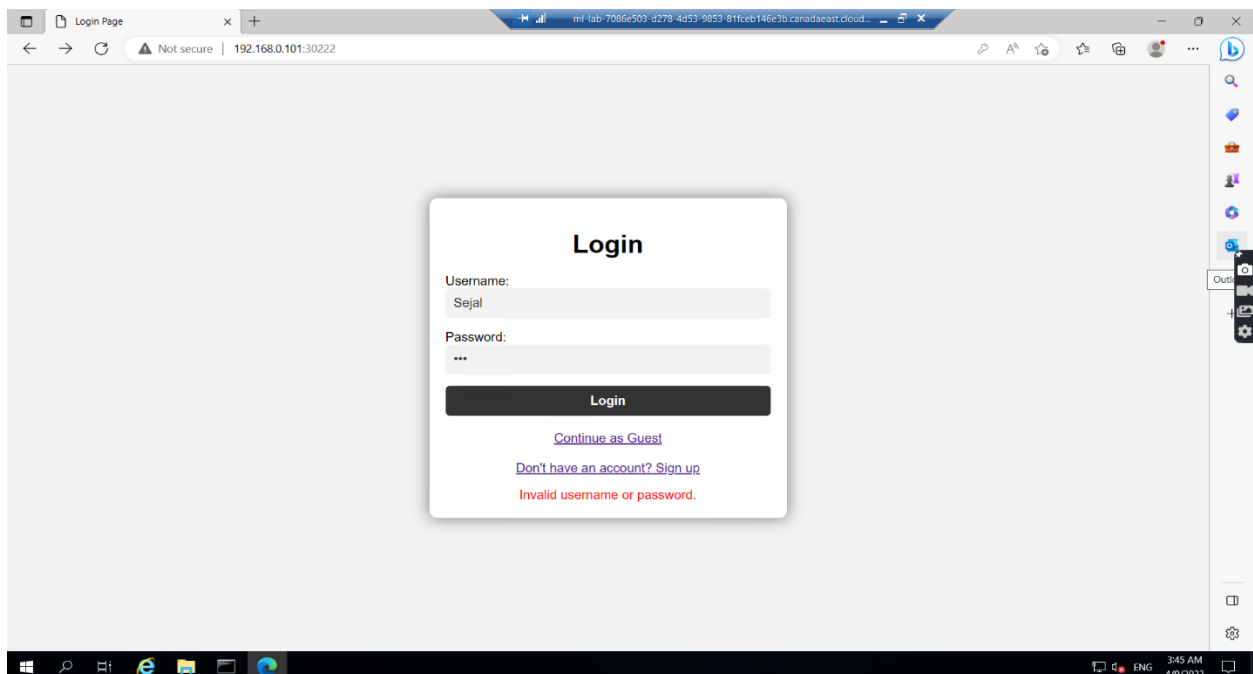
To test the functionality of the feature implemented, just try logging in with any user information, it will throw an error if the username or password is incorrect. If the username is correct, it will proceed to index.html page.

User Registration

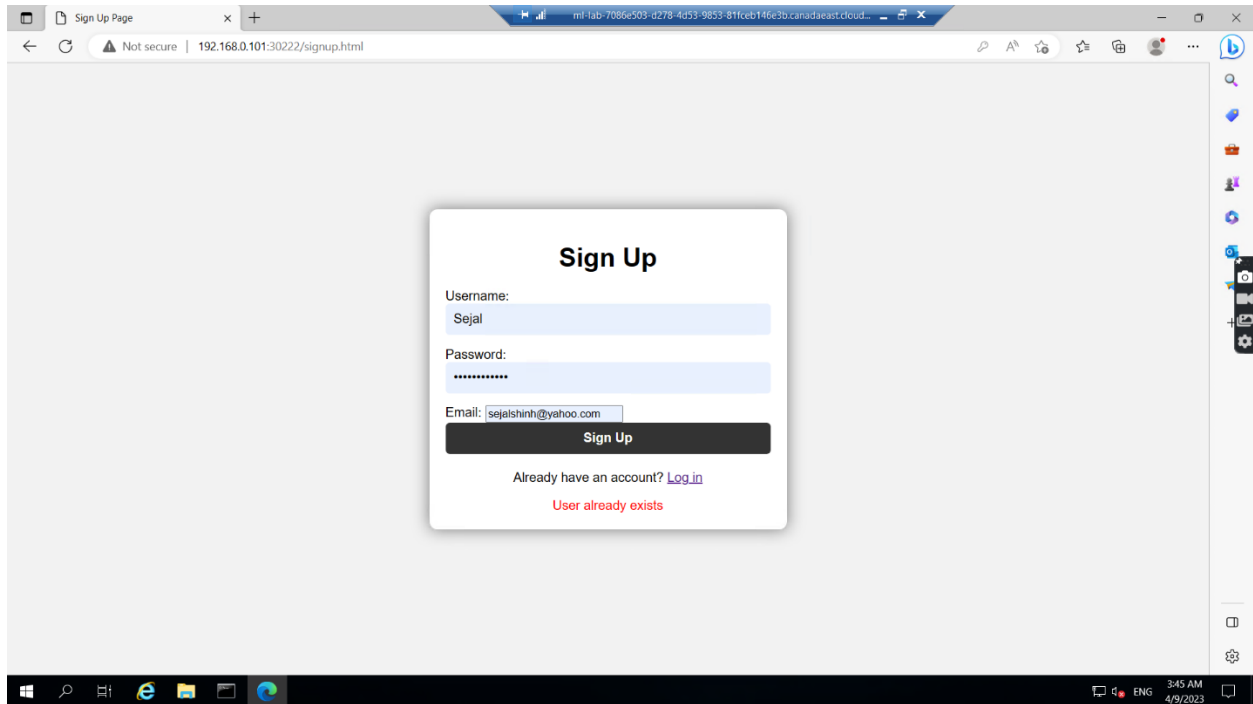
Just enter the user information for a new user and the user will be added to the list of users stored in the backend. After the session restarts, the information is deleted. On pressing Signup, it takes you to the index.html and adds the new user to the backend. If the user already exists, it will throw User already exists error.

In case of successful registration or authentication, it does not show anything but just takes you to the chat message.

Screenshots



Error on entering incorrect password.



Error on trying to register when the user already exists.