yachapp

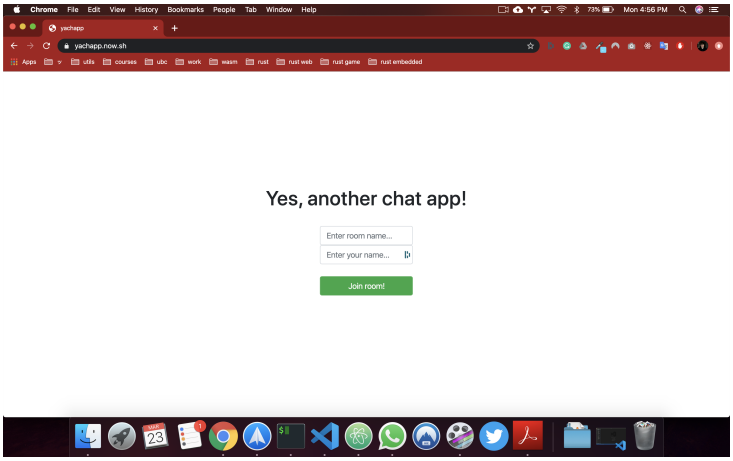
Danilo Chiarlone 34441162

Introduction

yachapp (short for "*Yes, another chat application*") allows up to 5 concurrent users to exchange messages via the utilization of MQTT. Beyond that, every message is also published to AWS's IoT core where it can be monitored or be utilized to perform other tasks.

High-Level Overview

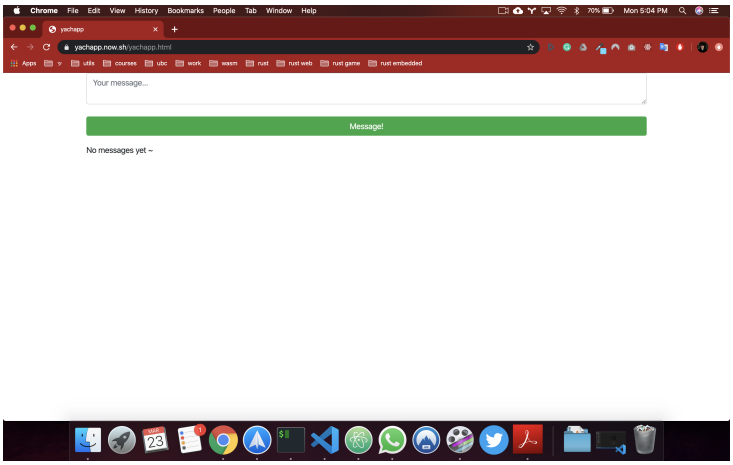
Upon accessing the site at yachapp.now.sh, the user is greated with the following screen:



The user has to enter:

1. their room name (the topic in IoT terms), and
2. their name, which serves no purpose other than visually distinguishing the source of messages that are sent.

After entering their info, the site will it store in a session and the user will be redirected to their chat page:



Here, I entered a room called *readme*. This page makes a `POST` request to the server every two seconds trying to fetch from a relational database[1] every message in that room. As this room hasn't been used before, there are no messages.

When posting a message, the backend will publish it to our AWS device and insert it into the database so it can be fetched later.

Check this link for a video demo of the walkthrough.

Technical Overview

- Database setup:

```
CREATE TABLE messages(
  mid SERIAL NOT NULL,
  room VARCHAR(16) NOT NULL,
  chapper VARCHAR(16) NOT NULL,
  message VARCHAR(128) NOT NULL,
  PRIMARY KEY (mid)
);
```

- Fetching messages[2]:

```
app.post('/fetch_messages', (req, res) => {
  let query = {
    name: 'fetch_messages',
    text: 'SELECT * FROM messages WHERE room = $1;',
    values: [req.body.room]
  }
  client.query(query, function(err, result){
    if(err){
      res.status(500).send(err);
    }
      res.status(200).send(result.rows);
  })
})
```

- Messaging:

```
app.post('/message', (req, res) => {
  device.publish(req.body.room, JSON.stringify(req.body.message));
  let query = {
    name: 'message',
    text: 'INSERT INTO messages (room, chapper, message) VALUES ($1, $2, $3);',
    values: [req.body.room, req.body.chapper,req.body.message]
  }
  client.query(query, function(err, result){
    if(err){
      res.status(500).send(JSON.stringify(err));
    }else{
      res.status(200).send(JSON.stringify(`sent: ${req.body.message}`))
    }
  })
})
```

1 : Here I am using PostgreSQL with the free tier ElephantSQL, which allows for 5 concurrent users.
2 : Here I am using ExpressJS/NodeJS for server-side programming.
∗ : In terms of DevOps, I am utilizing `now` where I am separately deploying the backend and the frontend.