Northeastern University

Final Project Report

Yay! — Live Streaming Content Website

Course: Scalable Distributed Systems 6650
Professor: Prasad Saripalli
Author: Jie Zhang

Yay! — Live Streaming Content Website

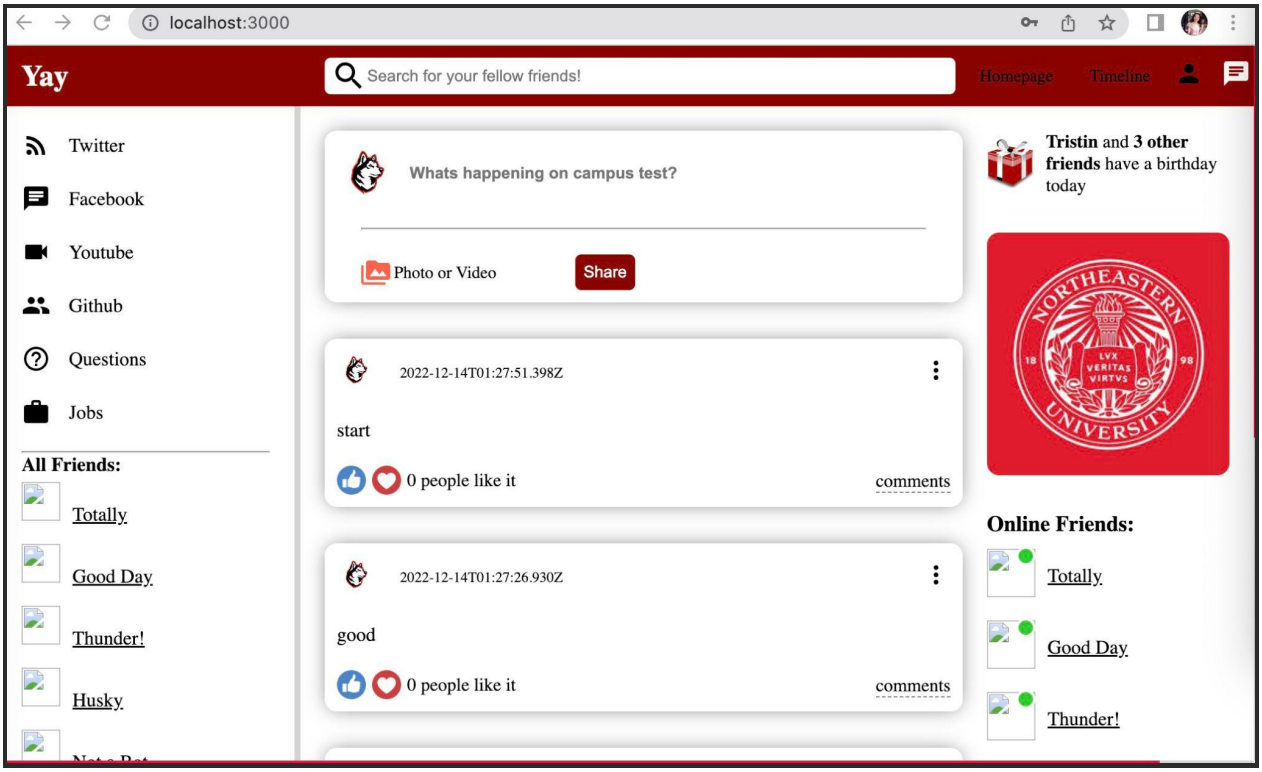
Overview

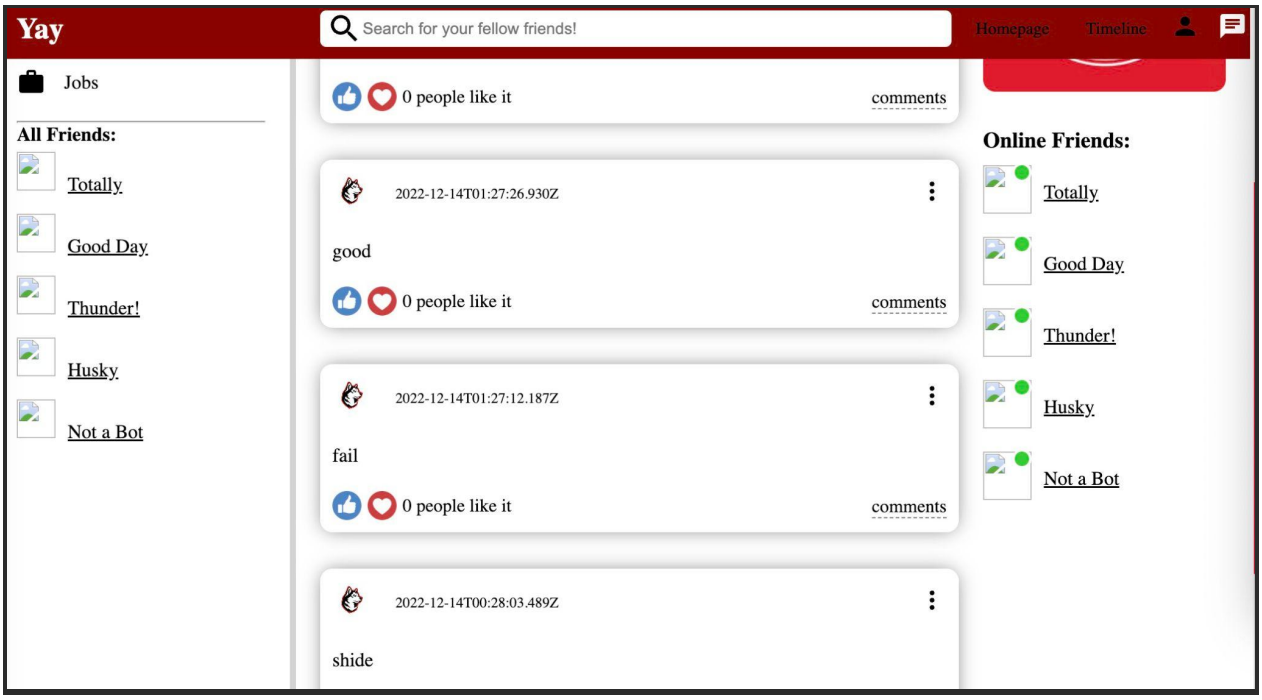
This project focuses on the live streaming website based on the content as the popular websites: Youtube and Instagram. The website platform allows users to post the blog content in real time as opposed to recording and uploading it after the event.

The project will focus on the whole development cycle in the software part including backend, data migration, storage, user login, security, web framework and so on. The project will not dive deep for the specific area but will try best to gain domain knowledge from the development life cycle even though it is a course project.

In order to easily let users remember the website, we called this website ‘Yay’.

How to start demo this project?





Steps:

1. Go to the browser and input the host web: http://localhost:3000/
2. Create a username and login in; or just use demo test account:
   Username: test@gmail.com, password: 123456
3. Post any test on the domain website, and you can see the previou posts and the time.

   Note: http://localhost:8080/ is the local host page.

   Note:
   When the user login in the website, he or she can post a context and click to connect with other users' posts.Currently, the page adds some links and lets the user conveniently explore some popular websites. Also can see the friends lists.

How to debug for this project?

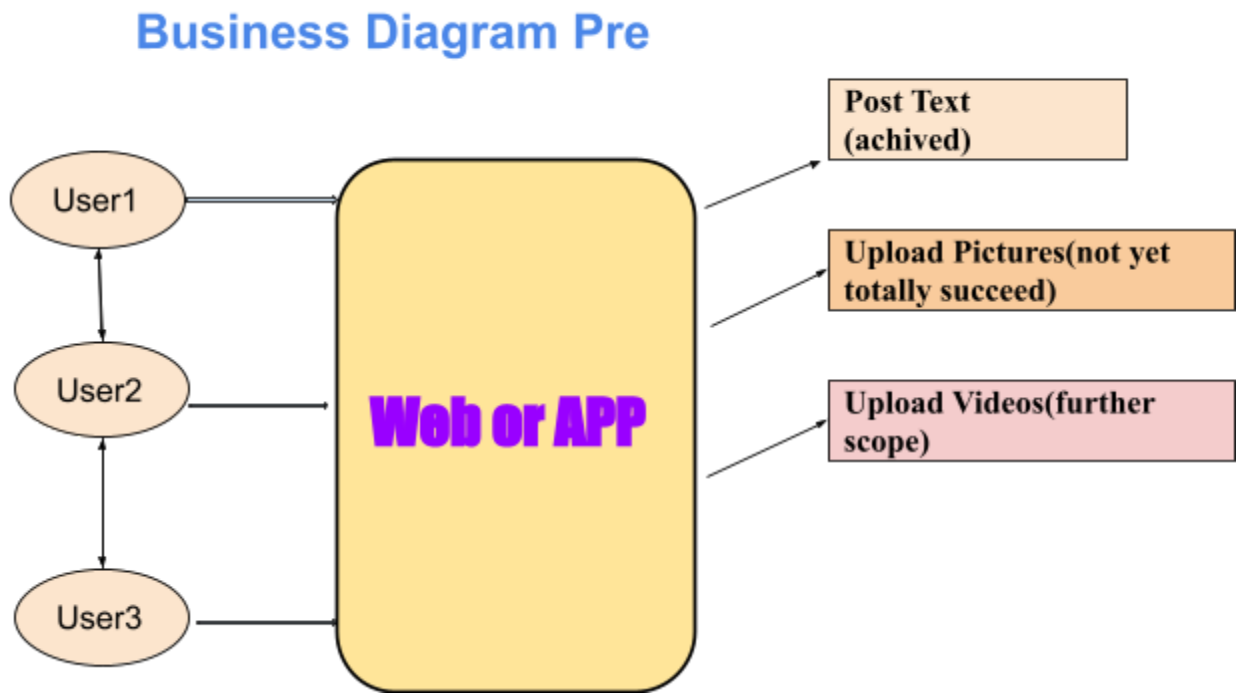
1. See the MongoDB connection: go to dest Docker application and see the connection.
2. Update the source code in Visual Studio Code
3. Use terminals to easily update the resource files and do some operation.

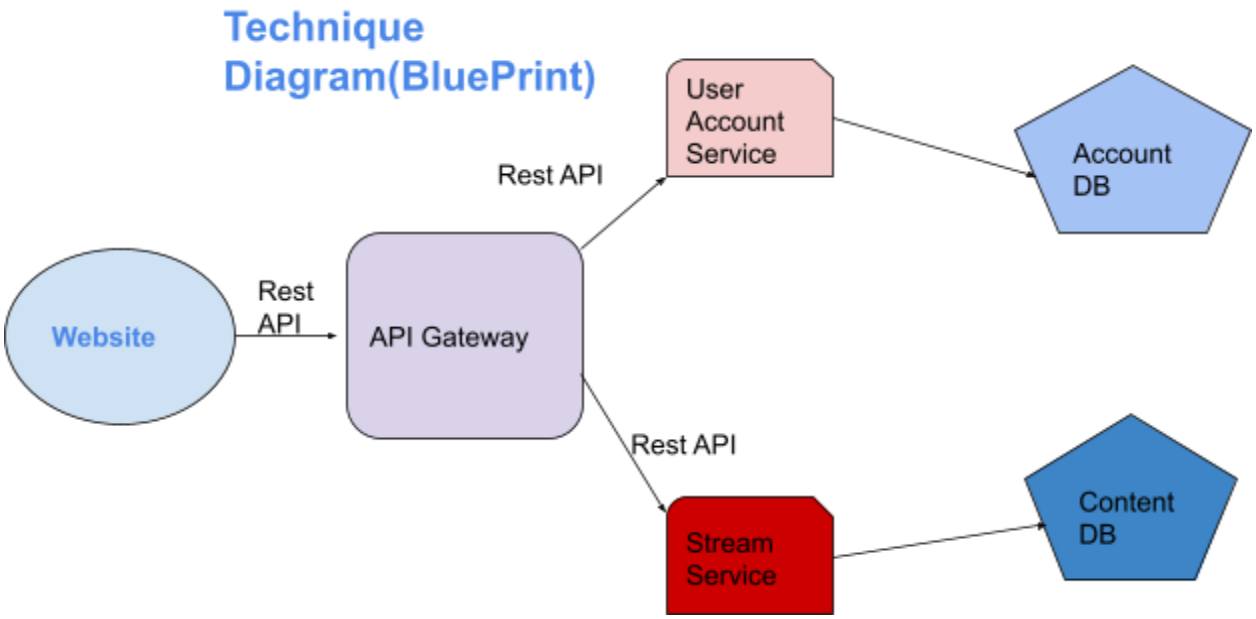
Project Idea

In recent years, there are a bunch of live stream websites like youtube, pinterest, instagram, facebook, twitter, tik tok, snap and so on. There are three basic types of stream website or application: on-demand streaming services, live tv streaming services, live streaming services. For a regular user, we can feel some required features and needs are not actually met when we use these applications. For example, as a facebook user, if someone wants to upload a video, it might take a long time. Also if we want to use snap to take a video, and post it, sometimes it will be delayed or not uploaded successfully.

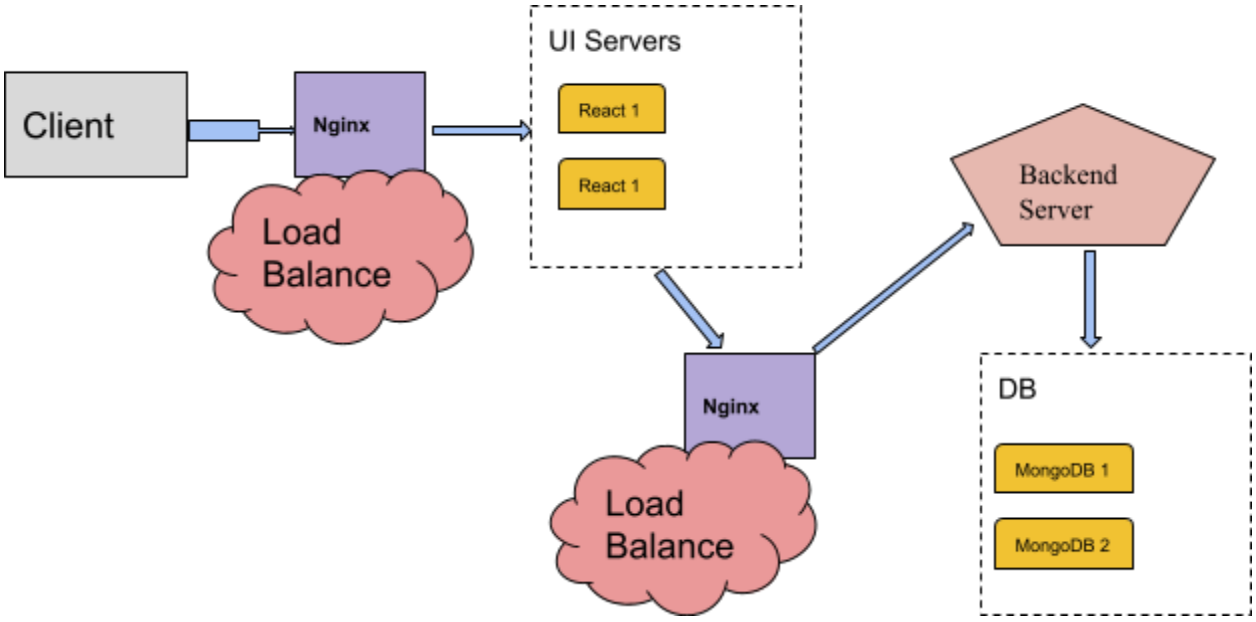
As a computer science student, I really hope I can understand why my request does not get the response when I use the application or service, also I hope

I can understand all the parts of live stream development. After researching and comparing the course content, I decided to do this project individually and try my best to use my distributed system knowledge and concepts to achieve this project. I really hope I can use some advanced algorithms and magic design principles to implement this website.





Project Structure Overview



Design tech (including some tools)

Backend:

1. User login in — basic requirement(hashmap).
2. Users' data storage. Mongodb for database.
3. Communicate with each node: React(online documentation).

Deploy:

Using docker,install the docker desktop app.

Web:

JavaScript programming,

Project Progress Chart

File	Description	Status
Project Proposal	Plan and Tools	Approved

File	Description	Status
Tasks List	Divide each steps to small tasks	Approved
Coding packages	Divide tech parts to small packages in IDE, skeletons for some packages.	Approved
Coding	Coding parts	Approved

Related Course Concepts and work for project

1)How to let the servers and clients communicate with each other?

We have two types of communication between servers and clients. First we can let clients and servers communicate directly in our architecture design.

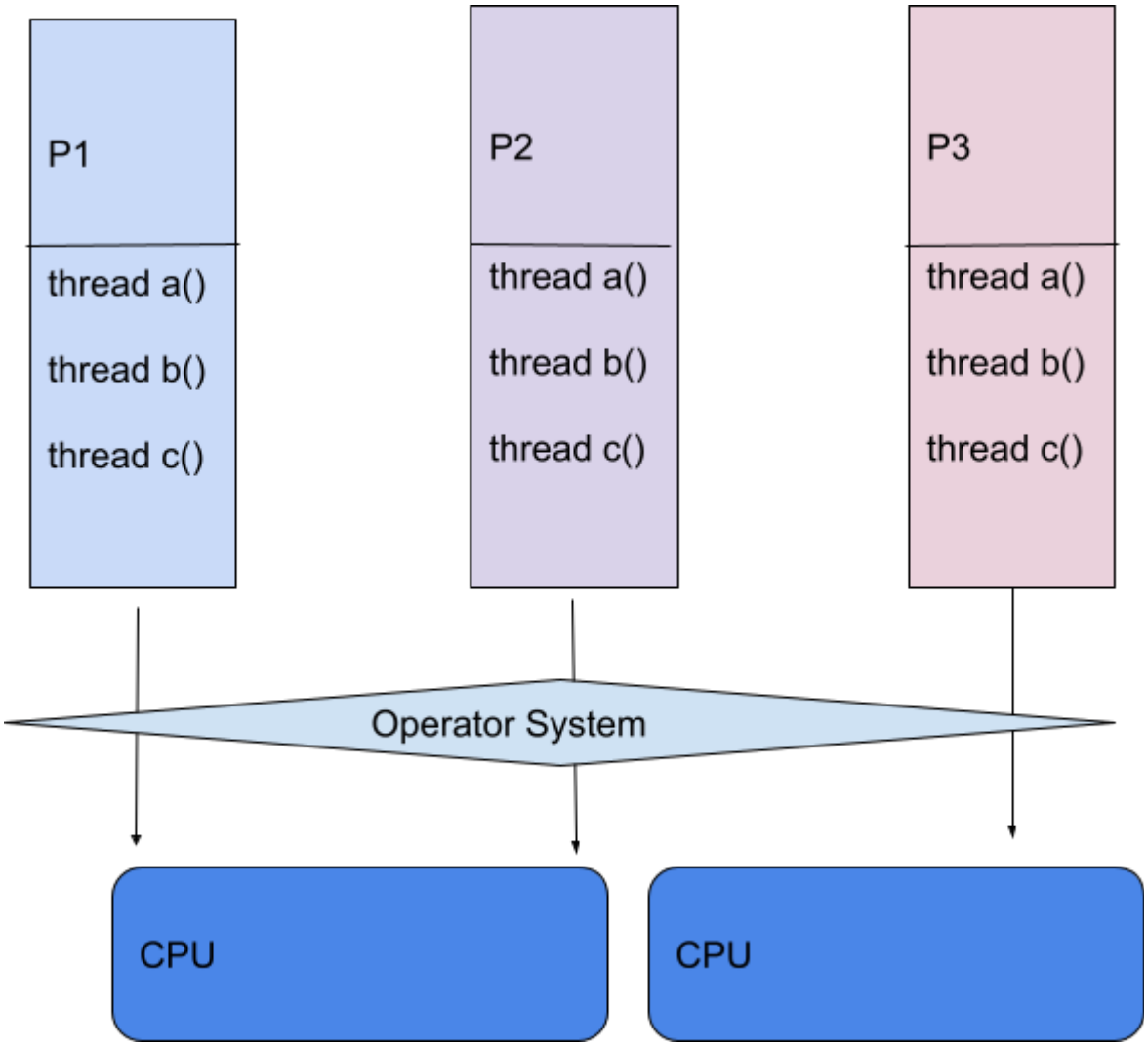
However, there are four weaknesses about this way. First, the request will be multiplied and influenced efficiently in the system. Secondly, our server's port will be shown to the outside and influence the security. Thirdly, the IPC and TCP may be not generated and limit our further project maintenance. Also, if we are required

to upgrade the server, it is not easy to update because the programming source code is huge.

So in recent years, most server and client communicate by the API to handle each service, and provide a one way interface. The API gateway can induce the communication frequency and encapsulates all the details of the interface. Also gateway will generate the TCP and IP to improve the uncoupling code. Using gateway we can improve the authentication and make sure our system runs in a safe and stable enviroment.

For the textbook concept, we can use RPC to let each server communicate with others. RPC(Remote Protocol Call) is a remote invoke agreement based on the C/S model. It can let the client send the invoke request and wait for the server response. It is a very basic request-response model in distributed systems. We can general the step like: local object sent the request to local pc, and we don't need to write the low level language and local pc will sent the request to the serve by the network, when the server received the parameter from the local pc, they will send back the result to the local client.

We can see the diagram like below picture:

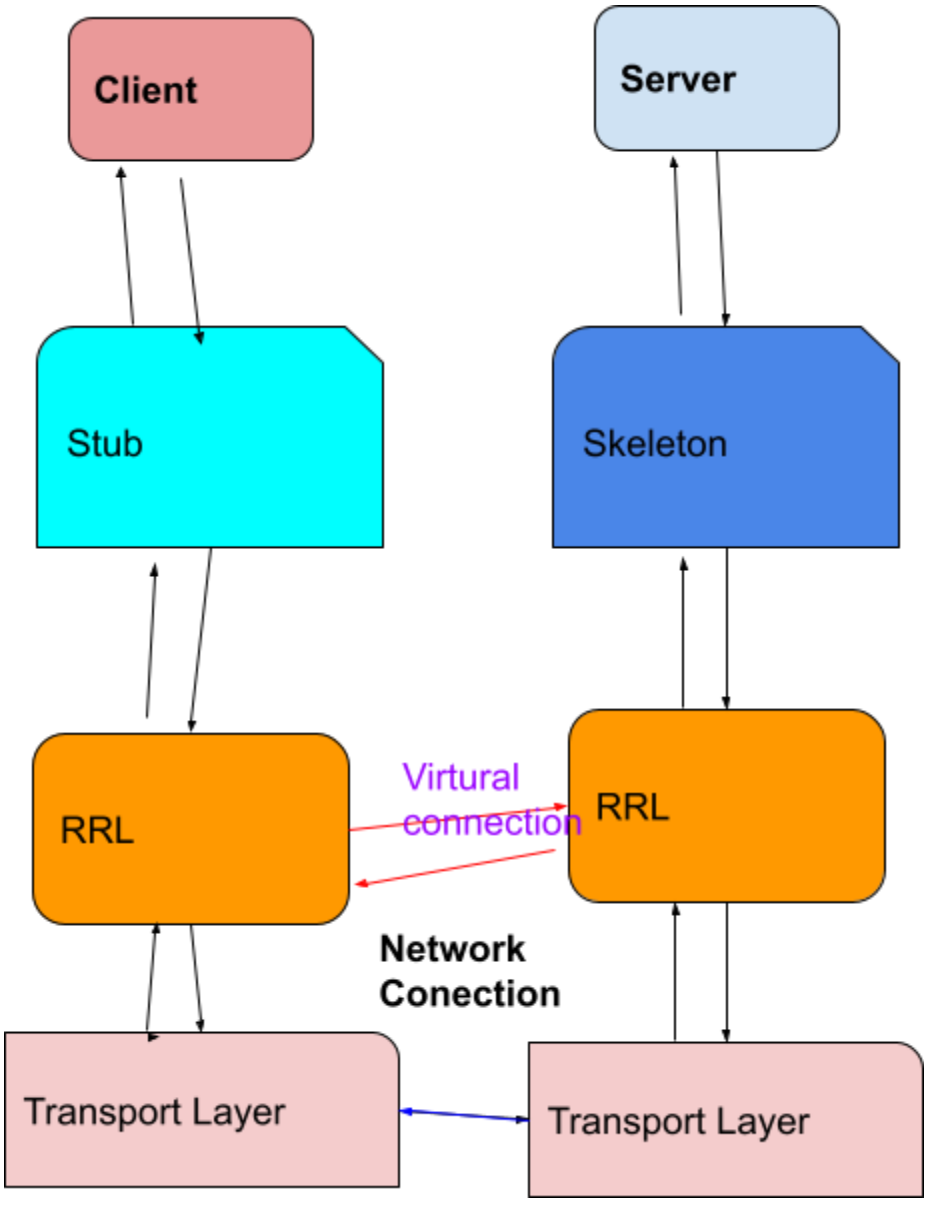


However, instead of RPC, another way is more convenient to support the asynchronous threads in computer science.

In our textbook ,we learned another way to let the communication happen between server and client. RMI(Remote Method Invocation) is a remote way to

invoke the method. We can consider the RMI as a Java programming language to let the RPC become more general. Because RPC can support the object commute . RMI is a more object oriented way to let objects connect with each other. It does not use SOAP to achieve it. RMI uses a Java remote message protocol to let the method be running in any operator system (but needs Java). Compared with RPC, RMI can use java garbage collection, object oriented. Also RMI uses stubs and skeletons to let remote objects communicate. Stub is an agency for remote objects. It has the same interfaces as the remote objects. Invoking a remote object to invoke the stub to achieve it if using RMI. The date transmission when using RMI is java objects, not XML data.

After reading the textbook and comparing RMI and RPC, I will try to use RPC to implement the communication part in each node. Might be to use REACT, Soap, Swift, or snifty.


Client
Server
Stub
Skeleton
RRL
Virtural
connection
RRL
Network
Conection
Transport Layer
Transport Layer

Compare two basic methods we learned in the class, we need to find a good way to let the communication happen and support multiple - threads and asynchronous communication.

2) Distributed ID Generate Way:

In distributed systems we need a unique global id to let the system computing with growing internet usage and high dependencies on different applications. We have many ways to approach this goal. If we are not working in a distributed system we can use a very basic way to approach element school mind: just increasing the number depends on the inserting object's order. Some applications are still using this approach. Another way to generate an ID as a timestamp function, but multiple independent servers can generate the same ID. Also, the same ID is generated for two consecutive requests. After searching online resources and reading textbooks, we found three good ways might be used in our project.

We might use Sonyflake, Snowflake or Baidu ways to generate our global ID in our system.

Here is the overall comparison table between different approaches:

	UUID / GUID	MongoDB	Flickr: Centralize MySQL	MySQL: Cluster	Twitter Snowflake	Sonyflake	Baidu UID Generator
Ordered	No	Yes, Possible	Yes	Yes	Yes, Roughly	Yes, Roughly	Yes, Roughly
Scale	Yes	Yes	No	Yes, Partially	Yes	Yes	Yes
Length (bits)	128	96	64	64	64	63	64
Uniqueness	Rare	No, Possible	No	No	No	No	No
Simplicity	Yes	Yes	Yes	No	No	No	No
Geo-Latency	Fast	Fast	Slow	Fast, Possible	Fast	Fast	Fast
Indexable	Hard	Yes	Yes	Yes	Yes	Yes	Yes

Comparison table between different approaches

https://blog.devgenius.io/7-famous-approaches-to-generate-distributed-id-with-comparison-table-af89afe4601f

3) Tips to improve Internet-based audio and video streaming applications in order to achieve acceptable quality:

Some improvement tips:

1) Improve the streaming protocols such as HLS or DASH for transferring video over the Internet. They are usually the first choices when it comes to delivering video over HTTP.
2) Trying to produce multiple streams (perhaps, three) of different bitrates and resolutions, then the player can inspect its available bandwidth and intelligently choose one of the bitrates to adapt to the changes in the network conditions (aka, ABR streaming).
3) Choosing the right segment duration during the packaging step (i.e., creating the HLS/DASH compliant stream). The packager chops up a video into small, equal-sized chunks and normally, then has to configure the size of a chunk in the packager or the live streaming provider's platform.
4) Increase in Internet transaction speed and improve the physical equipment.
5) Use a High-Speed Internet Connection to Stream.

List of all technologies in Project

Visual Studio Code—---- Programming IDE

MongoDB —---Database

Backend— two node1 for the backend server and do the load balance

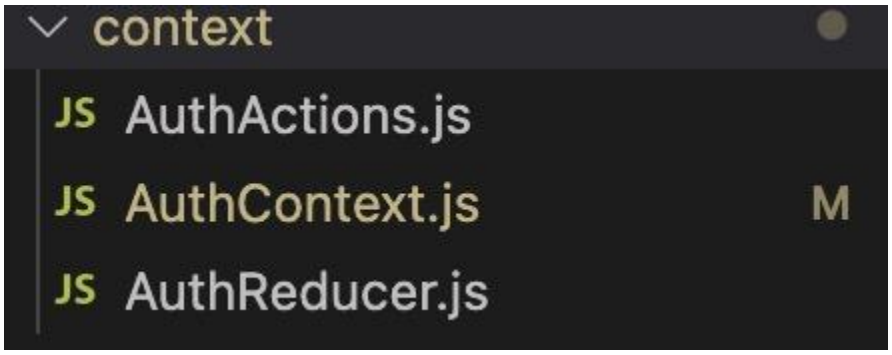
Backend Proxy – Ngnix for load balance and fault tolerance

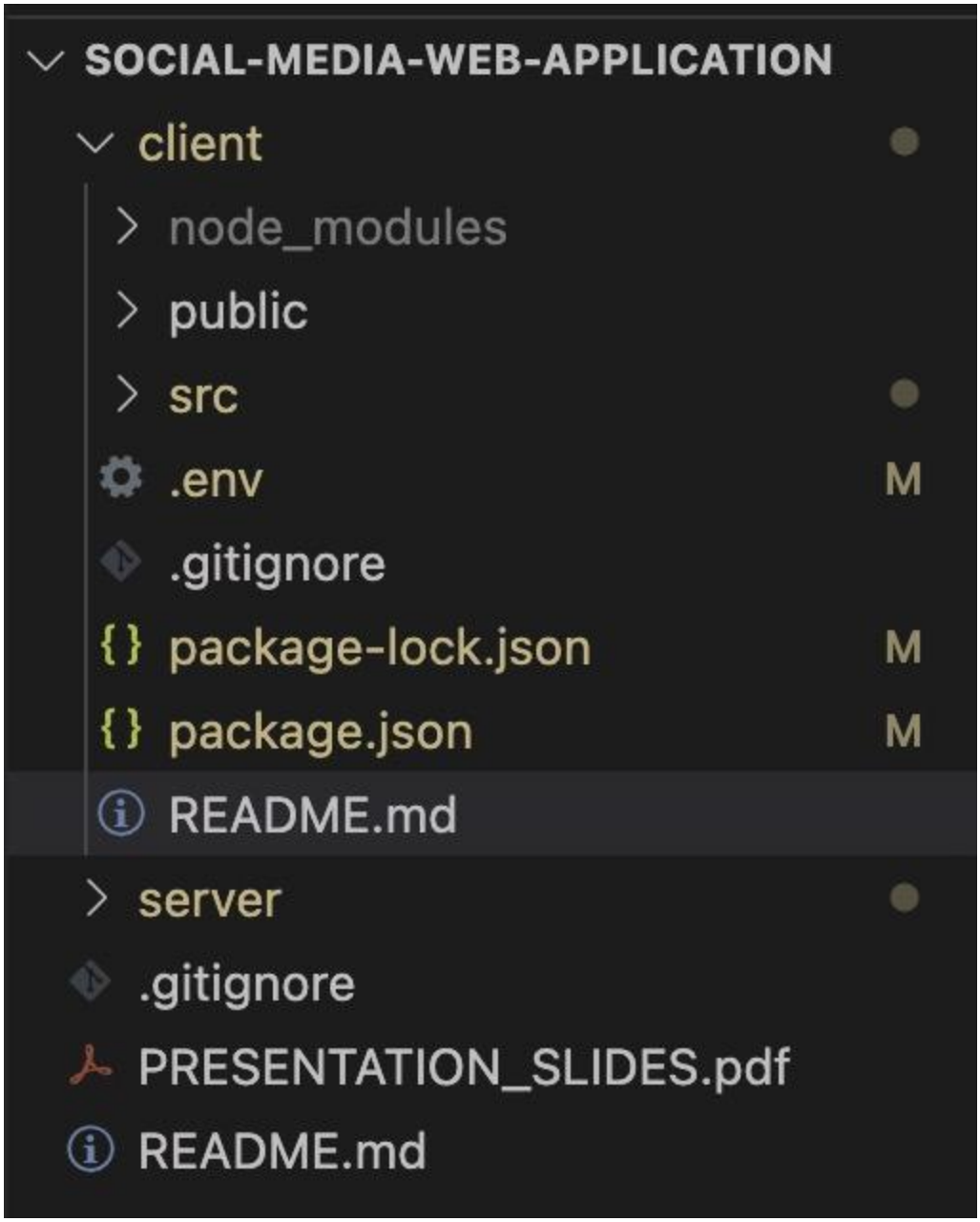
UI server —-----React 1 and React 2

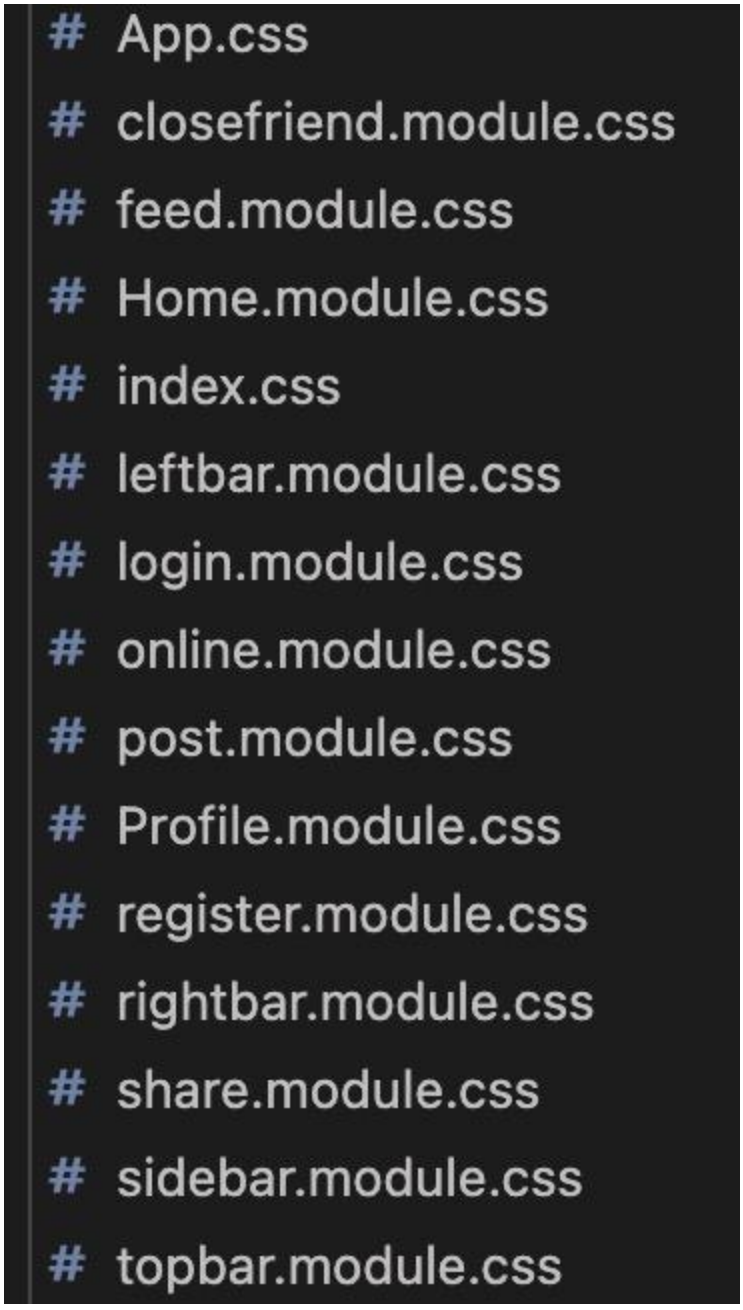
UI Proxy — Nginx for load balance

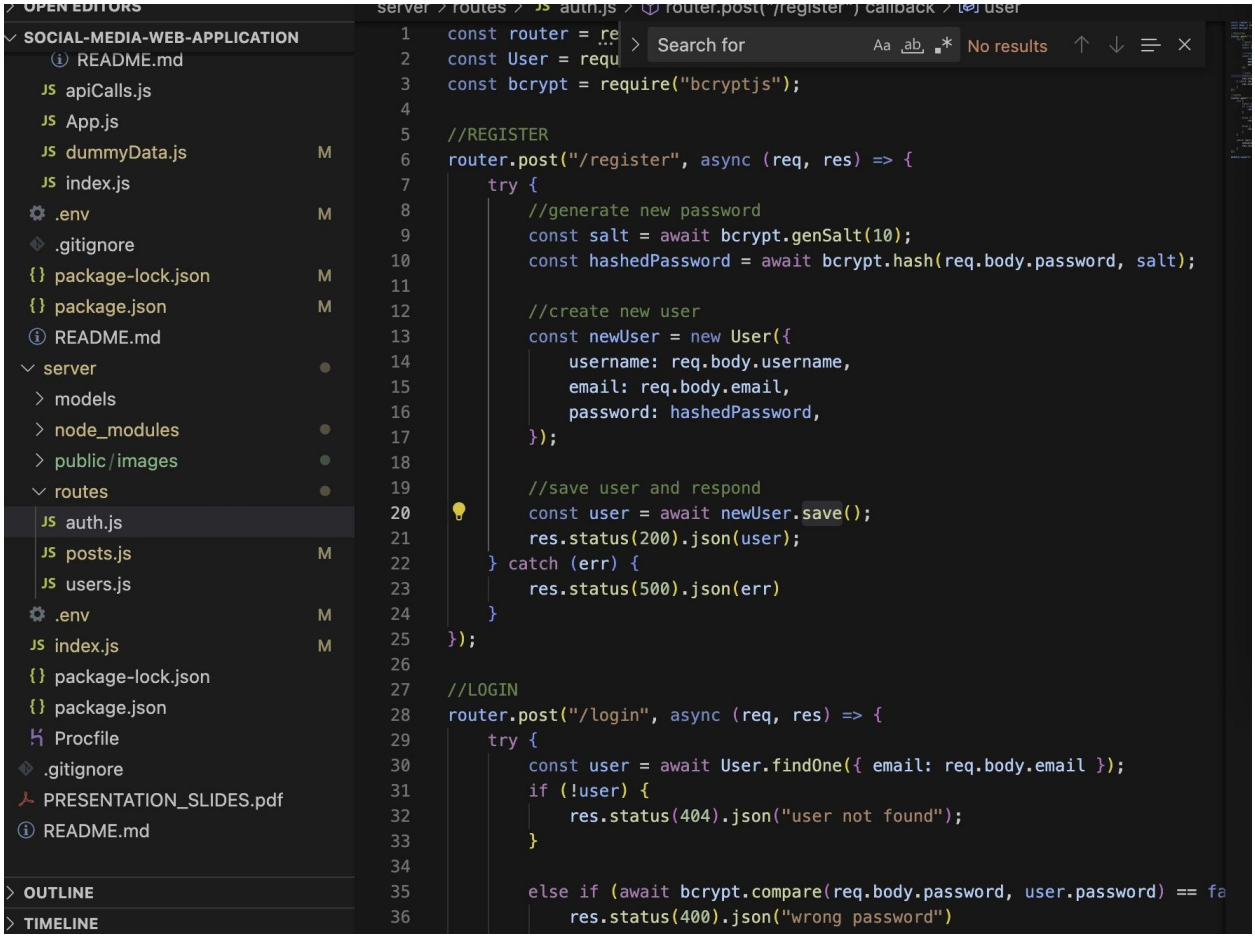
Client side —-- use Nginx to load balance and send the individual requests to the server and achieve the fault tolerance.

Project Code Overview:

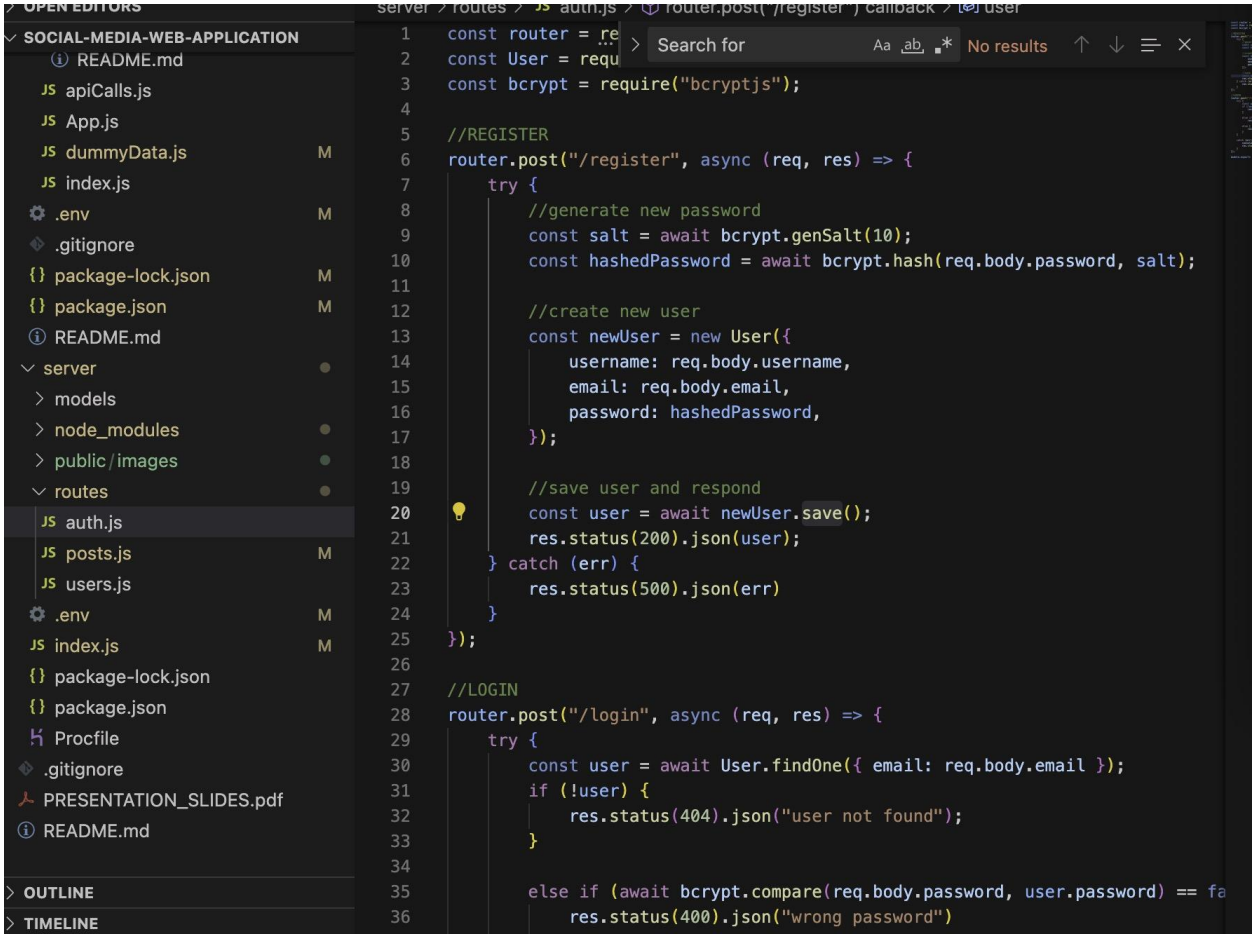



SOCIAL-MEDIA-WEB-APPLICATION
client
node_modules
public
src
.env
M
.gitignore
package-lock.json
M
package.json
M
README.md
server
.gitignore
PRESENTATION_SLIDES.pdf
README.md


App.css
closefriend.module.css
feed.module.css
Home.module.css
index.css
leftbar.module.css
login.module.css
online.module.css
post.module.css
Profile.module.css
register.module.css
rightbar.module.css
share.module.css
sidebar.module.css
topbar.module.css


OPEN EDITORS
SOCIAL-MEDIA-WEB-APPLICATION
README.md
apiCalls.js
App.js
dummyData.js M
index.js
.env M
.gitignore
package-lock.json M
package.json M
README.md
server
models
node_modules
public / images
routes
auth.js
posts.js M
users.js
.env M
index.js M
package-lock.json
package.json
Procfile
.gitignore
PRESENTATION_SLIDES.pdf
README.md
OUTLINE
TIMELINE
server > routes > auth.js > router.post("/register") callback > user
Search for
No results
const router = re
const User = requ
const bcrypt = require("bcryptjs");
//REGISTER
router.post("/register", async (req, res) => {
try {
//generate new password
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(req.body.password, salt);
//create new user
const newUser = new User({
username: req.body.username,
email: req.body.email,
password: hashedPassword,
});
//save user and respond
const user = await newUser.save();
res.status(200).json(user);
} catch (err) {
res.status(500).json(err)
}
});
//LOGIN
router.post("/login", async (req, res) => {
try {
const user = await User.findOne({ email: req.body.email });
if (!user) {
res.status(404).json("user not found");
}
else if (await bcrypt.compare(req.body.password, user.password) == fa
res.status(400).json("wrong password")

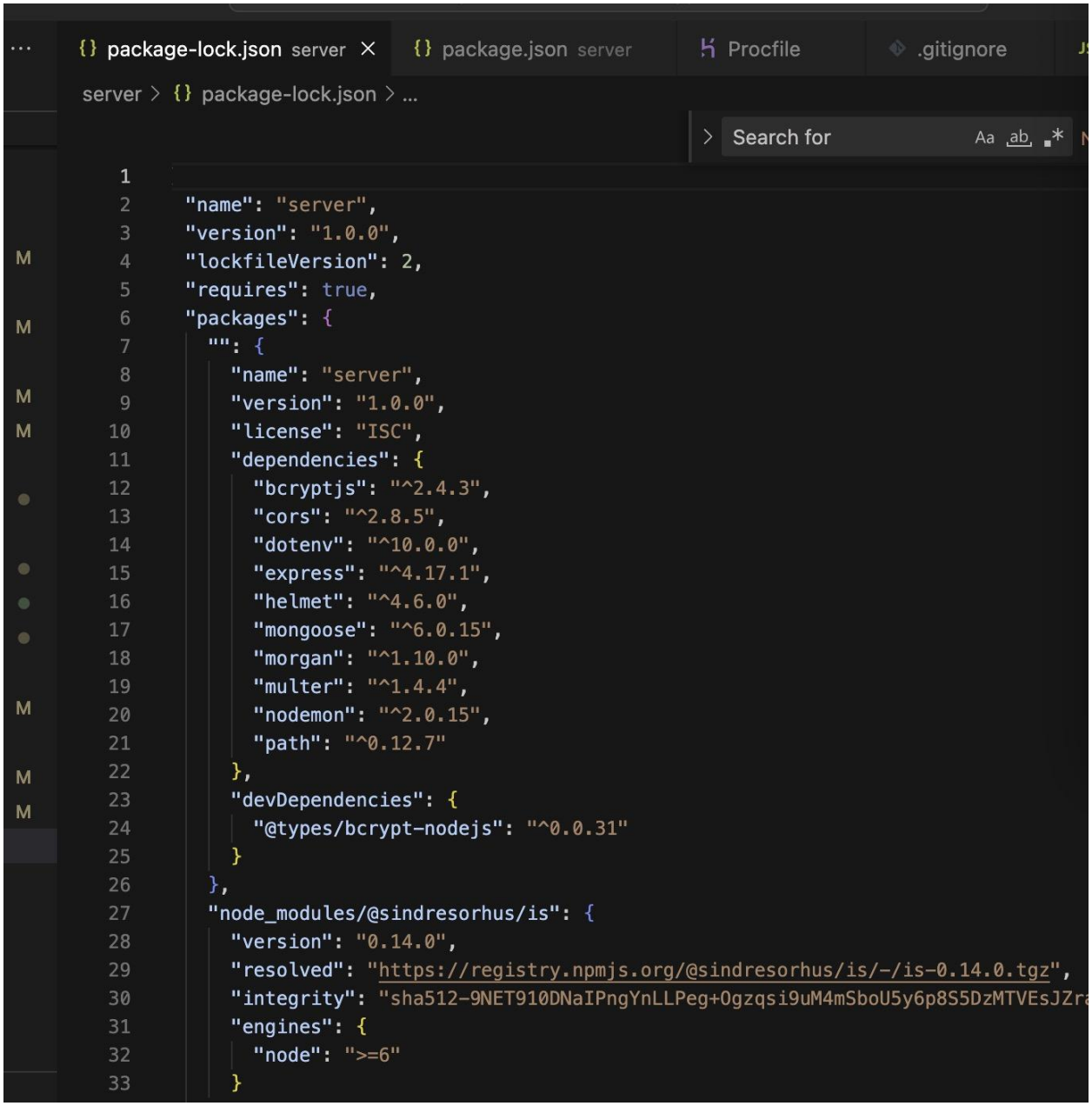

components
pages
CloseFriend.jsx
Feed.jsx
Online.jsx
Post.jsx
Rightbar.jsx
Share.jsx
Sidebar.jsx
Topbar.jsx


OPEN EDITORS
SOCIAL-MEDIA-WEB-APPLICATION
README.md
apiCalls.js
App.js
dummyData.js M
index.js
.env M
.gitignore
package-lock.json M
package.json M
README.md
server
models
node_modules
public / images
routes
auth.js
posts.js M
users.js
.env M
index.js M
package-lock.json
package.json
Procfile
.gitignore
PRESENTATION_SLIDES.pdf
README.md
OUTLINE
TIMELINE
server > routes > auth.js > router.post("/register") callback > user
Search for
No results
const router = re
const User = requ
const bcrypt = require("bcryptjs");
//REGISTER
router.post("/register", async (req, res) => {
try {
//generate new password
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(req.body.password, salt);
//create new user
const newUser = new User({
username: req.body.username,
email: req.body.email,
password: hashedPassword,
});
//save user and respond
const user = await newUser.save();
res.status(200).json(user);
} catch (err) {
res.status(500).json(err)
}
});
//LOGIN
router.post("/login", async (req, res) => {
try {
const user = await User.findOne({ email: req.body.email });
if (!user) {
res.status(404).json("user not found");
}
else if (await bcrypt.compare(req.body.password, user.password) == fa
res.status(400).json("wrong password")

server > models > JS Post.js > <unknown>

```
1   const mongoose = require("mongoose");
2
3   const PostSchema = new mongoose.Schema(
4   {
5           userId: {
6               type: String,
7               require:true,
8           },
9           desc: {
10              type: String,
11              max:500,
12          },
13          img: {
14              type: String,
15          },
16          likes: {
17              type: Array,
18              default:[],
19          }
20  },
21
22      { timestamps: true }
23
24  );
25
26  module.exports = mongoose.model("Post", PostSchema);
```

Search for


package-lock.json server
package.json server
Procfile
.gitignore
server > {} package-lock.json > ...
Search for
{
"name": "server",
"version": "1.0.0",
"lockfileVersion": 2,
"requires": true,
"packages": {
"": {
"name": "server",
"version": "1.0.0",
"license": "ISC",
"dependencies": {
"bcryptjs": "^2.4.3",
"cors": "^2.8.5",
"dotenv": "^10.0.0",
"express": "^4.17.1",
"helmet": "^4.6.0",
"mongoose": "^6.0.15",
"morgan": "^1.10.0",
"multer": "^1.4.4",
"nodemon": "^2.0.15",
"path": "^0.12.7"
},
"devDependencies": {
"@types/bcrypt-nodejs": "^0.0.31"
}
},
"node_modules/@sindresorhus/is": {
"version": "0.14.0",
"resolved": "https://registry.npmjs.org/@sindresorhus/is/-/is-0.14.0.tgz",
"integrity": "sha512-9NET910DNaIPngYnLLPeg+Ogzqsi9uM4mSboU5y6p8S5DzMTVEsJZr
"engines": {
"node": ">=6"
}

server > models > JS User.js > ...

```
const mongoose = require("mongoose");

const UserSchema = new mongoose.Schema({
    username:{
        type:String,
        require: true,
        min: 3,
        max: 20,
        unique: true,

    },

    email:{
        type: String,
        required: true,
        max:50,
        unique: true,

    },

    password:{
        type: String,
        required: true,
        min:6,

    },

    profilePicture:{
        type:String,
        default: "",

    },

    coverPicture:{
        type:String,
        default: "",
```

Search for

Project Further Scope

1. Might be to explore the specific area during the specific course work or popular technique principles.

   such as:

   Develop the mobile application —-- website convert to mobile application;

   Computing cloud —--- AWS and other cloud

   Ks8 to handle the decoupling problems —-- Kubernates and Docker containers..

   Data Migration and management—-- handle big data streams..

   Machine Learning — interval data.
2. Improve the timing and logic to make the system more reliable and scalable.
3. Group communications and individual communication during the users.
4. Handle the delay when uploading videos and storage problems.

Project Reference

MongoDB Documentation

https://www.mongodb.com/docs/

https://www.mongodb.com/compatibility/docker

Getting Started—-- React

https://reactjs.org/docs/getting-started.html

Npm-install

https://docs.npmjs.com/cli/v8/commands/npm-install

How to create a React App with a node backend

https://www.youtube.com/watch?v=w3vs4a03y3I

JavaScript

https://www.javascript.com/

Nginx

https://www.nginx.com/blog/connect-scale-secure-apps-apis-with-f5-nginx-management-suite/

Load balance

https://www.nginx.com/resources/glossary/load-balancing/

Fault-tolerance

https://www.ibm.com/docs/en/powerha-aix/7.2?topic=aix-high-availability-versus-fault-tolerance

Dummy Data

https://analystanswers.com/dummy-data-definition-example-how-to-generate-it/

Json

https://www.w3schools.com/js/js_json_intro.asp

https://www.w3schools.com/jsref/jsref_obj_json.asp