JACOBS
UNIVERSITY

Chat APP using NodeJS, EJS, mongoDB, SocketIO

by

Alex Tretyakov & Adnan Abu Ramadan

CS Project in Human Computer Interaction

Submission: 30.05.2021 Supervisor: Dr. Sergey Kosov

Jacobs University Bremen | Department of Computer Science and Electrical Engineering

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

Date, Signature

Contents

1 Introduction

With the rapid technological development, humans strive to improve the way they communicate. From the creation of personal computers to the birth of the internet, more and more people kept getting drawn to towards better and more efficient means of communication. In the early days of the internet, P2P messaging was gaining traction, but that was quickly over taken by services that are solely cloud based. With this nostalgia for the early days, and the current trend of becoming anonymous, we decided to create a chat application that immitates the p2p functionality. With the help of NodeJs (a javascript runtime environment) as our main framework for server-side scripting, EJS as our view engine, and mongodb as our database, we created a simple yet functional chat application.

2 Working Environment

The working environment consisted of:
OS: Windows10 (2021), MacOSX
IDE: Visual Studio 2019
EJS ver. 3.1.6
Express ver. 4.17.1
NODEJS ver. 16.1.0
mongoDB ver. 3.6.9
mongoDBCompass ver. 1.26.1
Socket.IO ver. 3.0.1
BCRYPT ver. 5.0.1

3 Implementation

The development environment was created in the Visual Studio IDE, and the necessary modules required for all libraries were installed using yarn/npm packet managers.
In the package.json file, you can see all installed and used packages and their versions.

The program has users connect to chat with each other. The user is created by using the insertOne() mongoDB method, such that the user is saved into the DB, and the password which is saved, is encrypted using bcrypt's hashing method. This allows for a hashed password to be saved into the DB, instead of a pure string of the password.

The webservice has multiple pages, including a signup and sign in page, and a chat page.

MongoDB

The mongoDB database is created and accessed using mongoDBCompass.

The database is connected to an amazon web services cluster via mongoDB Atlas, and then integrated with mongoDBCompass.

The database is accessed using POST and GET requests. A post request will update into the DB, adding a user for example. A get request will take info from the DB.
The user registration is done by doing a POST request to the DB, which adds the entered user data to the DB. This post request entails an encryption function, from the BCRYPT package, which encrypts the entered password before saving it.
The user sign in page works using a POST request which requests the user data from the DB and performs a data comparison check. Similarly to the sign up, the sign in utilizes the BCRYPT libraries, which in this case will perform the password comparison. The user enters the plain password string and the comparison function compares the hashed (saved) password, by first getting it for the respective user, with the typed paassword in the textbox. A correct login will generate a success.
Upon successful login, the client is redirected to the chat page, and connected to the socketIO event service using 'io.on'. Refer to the section SocketIO for more.

EJS/Client

EJS will take care of the webpage html schema. Inside this, we have the client services. Here, the client has a text field and a send button, this activates socketIO events, with a specified event listener handling them. The event listener takes the inputted message, send it to the server to process, and then the server sends the message to all clients, with the name of the client that sent the message attached to it.

server

The NodeJS library takes care of the server side of the program.
The server handles the user registration and login, providing the needed POST/GET/use/set functions. Also, the connection is established using the IRL which we have from our on-lone cluster in mongoAtlas. The client connects to it, through mongoose.connect(). The user is added to the DB using a fixed schema, such that all users follow the same schema, which includes: username, and password. The schema is a mongoose schema defined in the server.
The server listens on the specified localhost and port using http.listen.

SocketIO

The chatting service is handled through SocketIO. This makes it simple to connect users to the chat service and connect to other users.
The main function here is the 'io.on('connection')...' function, which is called when the user successfully logs in. When it is called, the user is connected to the chat service, which is using socketIO events, that are handled by a client inside the EJS file, which houses the hmtl for the webpage.
The client is event-driven and when a message is sent by the client, it is 'emitted' to every user connected to the service.

4 Conclusions

All in all, this project show how complicated it is to create something as simple as a chatting service, even though it is used every day by milions of people.
It also shows how even though some APIs are very simplified, it can still get very confusing and cluttered, when coding a webservice.

In addition, it shows how simple it is to connect a database, mongoDB, to a webservice, and using its services for it, to create users and retrieve their data securely using password hashing.
In reality, we need a lot more time to optimize everything and create a clean chatting service, but in the current stage, this is a working prototype. This would make a great summer break project for us.

5 Manual

Building the project

- (Make sure you have npm and nodejs installed on your system)

- Run ‘npm init -y‘ to initialize the nodejs project - Run ‘npm install‘ to build all the packages - **Please note** that nodemon need to be installed separately as a dev-dependency using the ‘npm install nodemon –save-dev‘ command.

Running the project

- Use the following command to run the project: ‘npm start‘ - This will start up the project and specify the url: (ex. ‘http://localhost:5050‘)

How to use the app

- Start up the app (i.e. run the project) and navigate to the specified url (ex. ‘http://localhost:5050‘). - Register a user using the provided registration field - Next, enter your username and password. After submitting, the page home page will redirect you to the chat page.

- To register another user and navigate to the same url (ex. ‘http://localhost:5050‘), open a new tab or window and follow similar proceedures as before.

- After both users logged in successfully, you are free to chat))

6 Workload

Alex: FrontEnd (EJS, html design), NodeJS, README.md.

Adnan: mongoDB service, project paper write-up.

Both: NodeJS, SocketIO, Bcrypt password hash, research,

7 Sources/References

https://socket.io/get-started/chat/

https://www.npmjs.com/package/bcrypt

https://ejs.co/# docs

https://docs.mongodb.com/