Simple Chat App Using WebSocket, MongoDB, and Next.js

1st 13519164 – Josep Marcello
Informatics Engineering Program
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung
13519164@std.stei.itb.ac.id

***Abstract*—As technology become more widespread, information moves faster and faster. There are many softwares that can and have used technologies mentioned in this paper. This paper will attempt to unravel the methods and explain the process of creating similar software. This paper will also show the challenges that could happen when creating such application using said technologies. The created software's performance can be affected by many factors. The most crucial component that affects the software's performance is the hardware that is being used to run the software.**

***Index Terms*—Websocket, NoSQL, web technology, HTTP**

I. Introduction

With the emergence of many new web-based applications, a fast method to communicate between the server of the application and the application itself. The tried and true method of using HTTP request paired with Restful APIs or GraphQL is usually enough. But, some type of applications need faster response. Fortunately, with the evolution of technologies and tools there are many ways to achieve low latency and fast response appliaction. This paper will show a such software that can be used to communicate between groups of people that is easy to deploy. This app utilise MongoDB and websocket to give the users of its application fast response. The framework that will be used is Next.js.

Another motivation of writing this paper is to report my exploration of learning the third technologies I have stated above. I will be graded for the course IF3280 Socio Informatics and Profesionalism of the Informatics Engineering Program at Bandung Institute of Technology (ITB) based off of this paper.

II. Theoretical Basis

A. Chat Application

In short terms, a chat application is a software that is created with the sole purpose of letting a person chat with another person or some people at once. The basic features that can be found in a chat application are receiving and sending messages. Often, a chat application can also send multimedia data, such as images, videos, and audio files. Some uncommon features that can be found in a chat appplications are, but not limited to, "last seen", read receipts, and group chats.

B. WebSocket

The contributors of MDN [1] defined websocket as "advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply."

From the citation, we can learn that websocket is a low latency, driven-based method for communicating between machines or processes. While both websocket and HTTP uses TCP/IP as the underlying communication protocol, websocket doesn't need to wait for the server to reply to send another request. In shorter, but more uncommon terms, websocket provides full-duplex communication while HTTP only give half-duplex communication. This advantage of websocket gives it the benefit of low-latency communication.

C. NoSQL and MongoDB

NoSQL is a mechanism to store information and data in a database. NoSQL differs from SQL in the way it accesses data. SQL uses SQL query to create, read, update, or delete data. While NoSQL depends on it's storage type. For example, Redis is a NoSQL database that uses key-value storage. To access data, only the key is needed. Another type of NoSQL database is MongoDB, which is a NoSQL database that uses document store.

MongoDB stores data in a document store that is made up of documents in collections. A collection in this case is analogous to a folder in a filesytem while documents are the files. Or, in SQL terms, documents are rows and collections are tables.

The way NoSQL stores its data is an advantage to SQL. NoSQL databases doesn't need complex migrations to setup its database. Changing the data definition is also fast because it doesn't need to rebuild and migrate the database. Another advantage is its speed. Wang, et al. [2] have shown that NoSQL databases are generally faster than SQL databases.

The disadvantage of using NoSQL is that it is not guaranteed to have ACID[1] properties. NoSQL databases also do not guaranteed to have join operations. One such example is

[1]atomicity, consistency, isolation, and durability

Amazon DynamoDB, while it has ACID properties, it doesn't have join operations.

D. Next.js

Next.js is a full-stack framework created by Vercel, Inc. It is a framework that is based on the popular front end library/framework, React.js. Next.js will also provide the back end framework. Its back end framework is similar to another back end library called express.js. With Next.js, developers can create and deploy their application with ease.

Next.js differs from React.js as it gives the ability to create SSR (server-side rendering) pages to React.js applications.

III. Building the Application

A. Preface

The code for this application is available on https://github.com/jspmarc/websocket-chat-app. The code is licensed using MIT License. While the demo of this application is accessible at http://socif.jspmarc.my.id (do note the protocol is HTTP and not HTTPS, so please be careful when sending sensitive data).

The next subsections (or sub-chapters) of this section (or chapter) will tell the story of writing this application.

B. Setting Up

Building this application needs Node.js because Next.js uses Node.js as its backbone. For the package manager, pnpm (https://pnpm.io/) will be used. This package manager provides faster speed when installing dependencies compared to yarn and npm.

Because this app is a Node.js app, socket.io will be used as a wrapper for websocket. Socket.io is a library that provides a way to create a websocket server and client easily. For MongoDB connector, the official connector (https://www.npmjs.com/package/mongodb) will be used. Because the data stored is not complex, complex ODMs such as Mongoose is not needed.

This application will also have login and registration for its users. For that feature, JWT-based authentication will be used. The library to support this feature will be jose (https://www.npmjs.com/package/jose).

Other libraries that are used for this application:

- Tailwind CSS: writes CSS directly in HTML classes, and
- dotenv: reads environment variables from `.env` files.

C. Baby Steps

To start the creation of this application, Bilgili [3] provided a simple and easy-to-follow tutorial. At the start, this application is a simple HTML with only an input field and a button. The button does nothing, but when the user typed in on the input field, an event will be emitted to the websocket server. With the event, a message, containing the current input field content, will be attached. The server then will broadcast the content of that message to the other websocket clients. Each websocket clients that receives and event will copy the content of that message to their input field. That was all done almost instantaneously.

The servers can broadcast the received messages because a listener that will listen to a certain event (in this case it's `input-changed`) is created. So, when a client emits that event, the listener will run a function that is assigned to the listener. The same story goes to the websocket client. It creates a listener on each client that also listens to an event (in this case and for simplicity's sake, it's also `input-changed`) that are being emitted by the server.

D. Forbidding

Next, to create the authentication feature, a login and register page is created. An unauthenticated user (i.e. not logged in) will be kicked out of the chat room to a login and registration page. This is achievable using Next.js' middleware. Next.js' middleware is executed on the application's back end. Because how the Next.js routing works, a middleware is only needed to be placed on the same folder or the parent's folder of said page to be executed everytime the user visits the page.

Some code to search the JWT token in the user's cookie is added as a middleware. The middleware will check the route the user is visiting and will only be executed if the route is to the chat room. This code will check whether the token cookie exist or not. When this cookie does not exist, Next.js will redirect the user to the registration/login page. When it does, the user will be send to the chat room normally. When opening the chat room, the browser will do a HTTP request to the application's back end to decode the user's JWT. The decoded token will then be saved to the browser's local storage[2]. The decoded user information will be used when the user sends a message to the chat room.

E. Storing Data

This section is started by designing the database schema. The design is based off of the feature and limitation of both, the application and the database management system. The application will only support a single chat room where users can talk directly to each other in the chat room. To add another chat room, another instance of MongoDB and the application needs to be deployed. With this information, the database schema that are going to be used become rather simple.

The database will only consists of two collections, `users` and `chats`. For simplicity's sake, `users` collections will only store the user's name, username, and password. The other collection, `chats` will also be simple because the applications will only consist of a single chat room. The `chats` collection's documents will only store the sender's name, sender's ID, the content and the time the message was created. Adding the sender's name and ID to the same collection breaks normalization, but it becomes easier to create and faster to query, especially in a NoSQL database. So, it's a worthy trade-off.

With this design in mind, the API for logging in and registering is modified to insert and search the database. After this is added, the users can successfully login and register.

[2]https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

What's left to do is adding generated token to the user's browser's cookie. This is done in the front end side of the application, with the back end only sending the token.

F. Chit-Chat

With authentication done, the feature left to build is sending and receiving messages.

The first page created is modified. The button, when pressed, now sends the new message that are broadcasted to other clients. So, it is needed to modify the previous websocket client and server code. The server, now rather than only rebroadcasting the message to other users, it will also insert it to the database with the other needed data. Also, the input field now doesn't get reflected on the other clients.

With the back end of the application done, what's left is to handle the websocket broadcast events in the front end side. Using React.js' powerful state management, the new messages are added to an array. When an element is added to the array, the page is going to get re-rendered so the new messages will appear on the screen.

How the server and client communicate with each other is the same as III-C. Only the name of the events and its messages content are changed. Namely, the event from client to server is called `send-chat` while the event from server to client is called `recv-chat`. Both events use the same message, which is a JavaScript object with the content the same as document for `chats` collection.

G. Styling

In this part, the power of Tailwind CSS is exploited. Styles are added to the pages to make the application look better, while also providing better experience for its users.

H. Logging Out

With all the features and styles done, what's left is adding a logout button. The logout page is pretty simple. It just run some JavaScript code that will delete the token cookie and the browser's local storage. After that's done, the user is then redirected to the login and register page. To complete this feature, a button is added to chat room that will redirect user to the logout page.

IV. Challenges

A. Next.js' Middleware Can't Use dotenv Library

This problem happened when the middleware tried to access an environment variable that was read using dotenv library. This problem is caused by dotenv library needing access to the filesystem, while the middleware is forbidden from accessing the filesystem.

The solution to this problem is really simple. Next.js automatically reads the environment variables, so this line of code needs to be added `const ENV_VAR = process.env.ENV_VAR;`. Do note, this line of code will result in an error `const { ENV_VAR } = process.env;`. While both codes are similar in "normal" JavaScript, because of some Next.js magic, the latter will cause an error[3].

B. Incorrect Amount of Received and Sent Messages

This problem happens when two or more users send and receive messages. For example, let's say user A sends a message to the chat room. Both the server and another user, let's say user B, will receive two, three, or more of the same message instead of just one. The same would happen when user B sends a message.

This problem is caused by multiple listeners to the same events are added. The cause to multiple listeners being added are various. Some common mistakes are:

- React.js' strict mode is enabled
  React.js' strict mode is enabled by default. This is a feature that will highlight potential problem in a React.js application. This feature makes the code, in development mode, runs twice. It is obvious what problem this will cause in the context of initializing and adding listeners to a websocket client.
- Initialization of the **client** socket instance is done multiple times
  Similar to the problem before, but this problem is more apparent in React.js. Internally, React.js will re-render the page everytime a change is applied. initializing websocket client and adding the listeners in the "wrong place" will lead it being re-initialized and the listeners being re-added when the page is re-rendered. To solve this problem in React.js, the initialization and adding listeners should be done inside a `useEffect` hook. It is also recommended to remove the listener of the socket client by adding a remover function to the return statement of the `useEffect` hook.
- Initialization of the **server** socket instance is done multiple times
  This problem is especially apparent in this application. In this application, adding listener to a client is done by doing a HTTP request to an API route which will register a listener to the server's websocket instance. Hitting this API multiple times results in the websocket instance having the multiple listeners to the same event.
  In the writer's case, this was the problem the writer had faced. The solutions was simple, it was by making the websocket instance initially null or any other value. Then, add a conditional such that when the websocket instance is null (or any other initial value), instantiate it and the listeners. If the websocket is not the same as the initial value (i.e. it is instantiated) and the conditional is hit, do nothing.

V. Conclusion

The application works as intended. With limited testing, the message sent and received are almost instantaneous.

[3]the writer of this report and application doesn't understand the underlying code of the framework. Because of that this report will not mention the real cause of the error.

Unfortunately, some lags still happen. Lags could happen when the messages saved in database is too much or when write operations to the database takes longer than usual. Because the message need to be written to the database before being broadcasted to all other users, other users may feel some delay to when receiving a message caused by the slow writes. It can be concluded that although the software is fast enough, hardware bottleneck can still be the pain point of creating fast and responsive softwares.

Acknowledgements

The writer would like to thank his parents, siblings, partner, and friends for all the support and help they've given. The writer would also like to thank his God for the continous blessings He has given to the writer. Lastly, the writer would like to extend his thank you to IF3280's teachers, professors, and assistants. Because, without their teachings and supports, this paper would not have been done and the writer would not have learnt this new and interesting technology.

References

[1] MDN Contributors, "The WebSocket API (WebSockets)," *developer.mozilla.org*, Apr. 28, 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Accessed: May 1, 2022].

[2] R. Wang and Z. Yang, "SQL vs NoSQL: A Performance Comparison," 2017. Available: https://www.cs.rochester.edu/courses/261/fall2017/termpaper/submissions/06/Paper.pdf. [Accessed: May 1, 2022].

[3] D. Bilgili, "Implementing WebSocket communication in Next.js," *LogRocket, Inc.*, Feb. 3, 2022. [Online]. Available: https://blog.logrocket.com/implementing-websocket-communication-next-js/. [Accessed: May 1, 2022].