Project Documentation – Secured Chat App

Guy Shukrun Cohen 208517110

You can visit the Secured Chat App:

https://secure-global-chat.herokuapp.com/

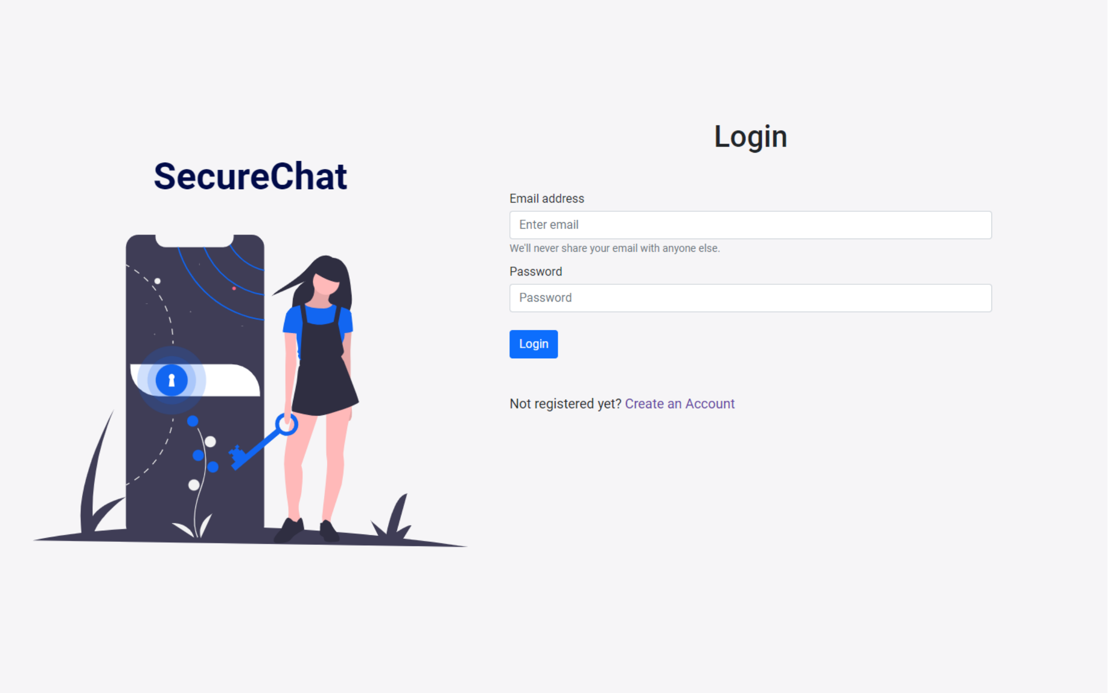
Note: On first startup the site can take up to 30 seconds to load due to Heroku free dyno policy.
The site is **not** responsive yet, so please access it using computer.

System Description

Secured Chat is an online global **secured** chat where people can communicate through the web with their friends, family, and other people.
The chat is global, meaning anyone can send message to anyone else, and make new friends!

System Flow

When the user enters Secure Chat, he is welcomed by the login page:

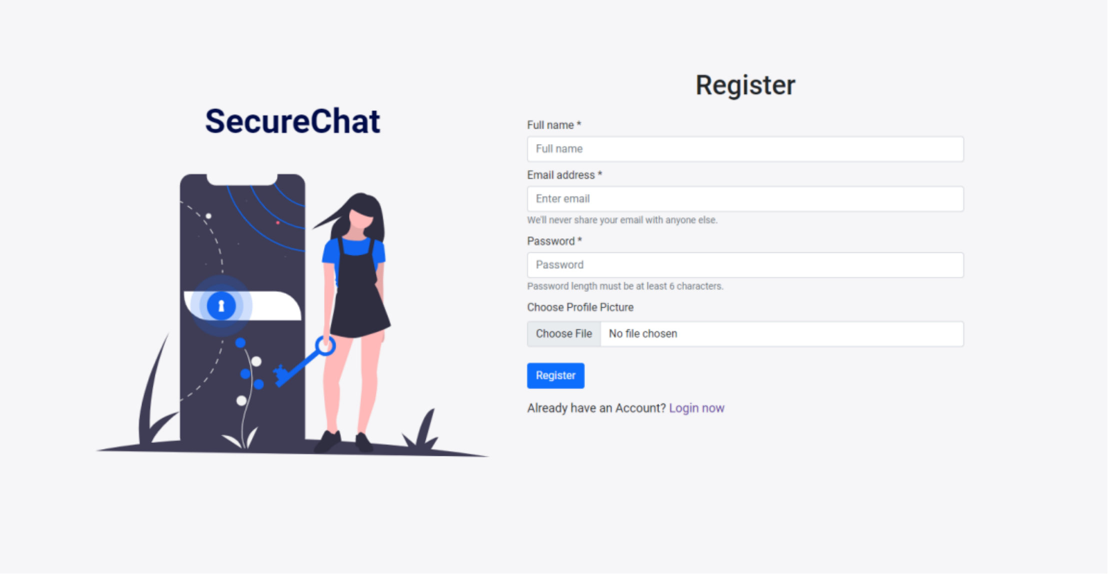


We'll never share your email with anyone else.

Not registered yet? Create an Account

The user can enter his credentials to login or create an account if he is not registered yet.

If the user clicked the "Create an Account" button he will be transferred to the register page:



SecureChat

Register

Full name *

Full name

Email address *

Enter email

We'll never share your email with anyone else.

Password *

Password

Password length must be at least 6 characters.

Choose Profile Picture

Choose File No file chosen

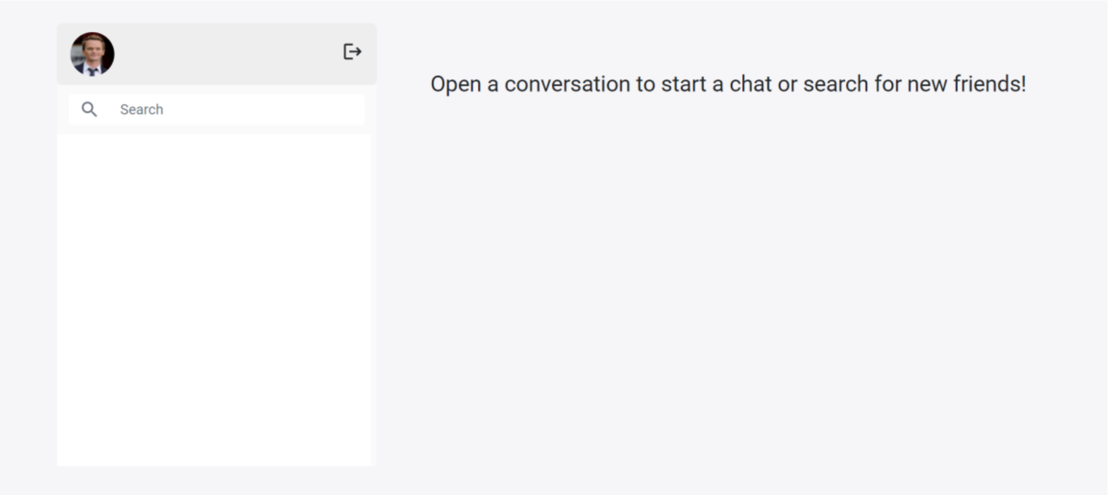
Register

Already have an Account? Login now

In the register page, the user can register and upload a profile picture from his computer.

After the user register he will be transferred to the login page to login the system.

After the user fill up his details and the details are correct in the database, he will be transferred to the Secure Chat page.

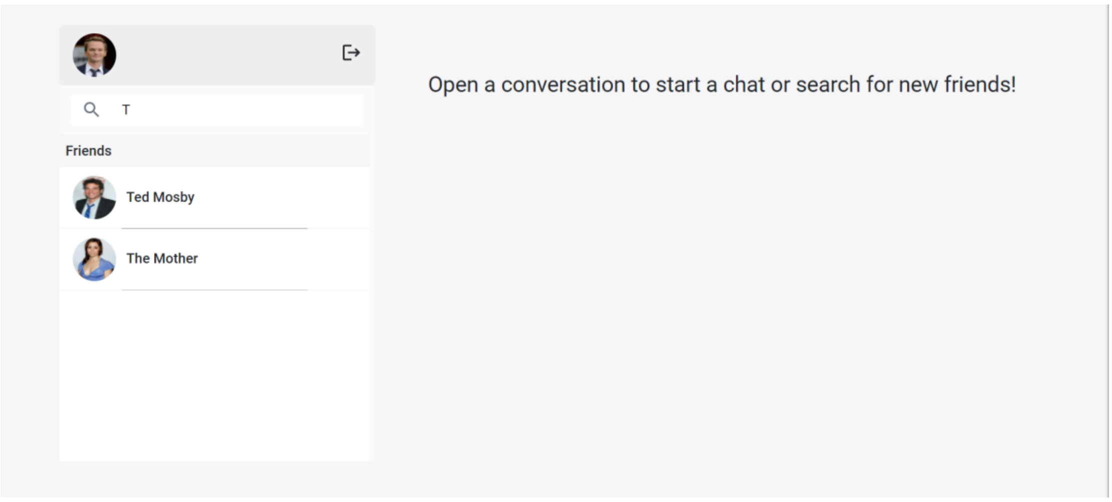


Search

Open a conversation to start a chat or search for new friends!

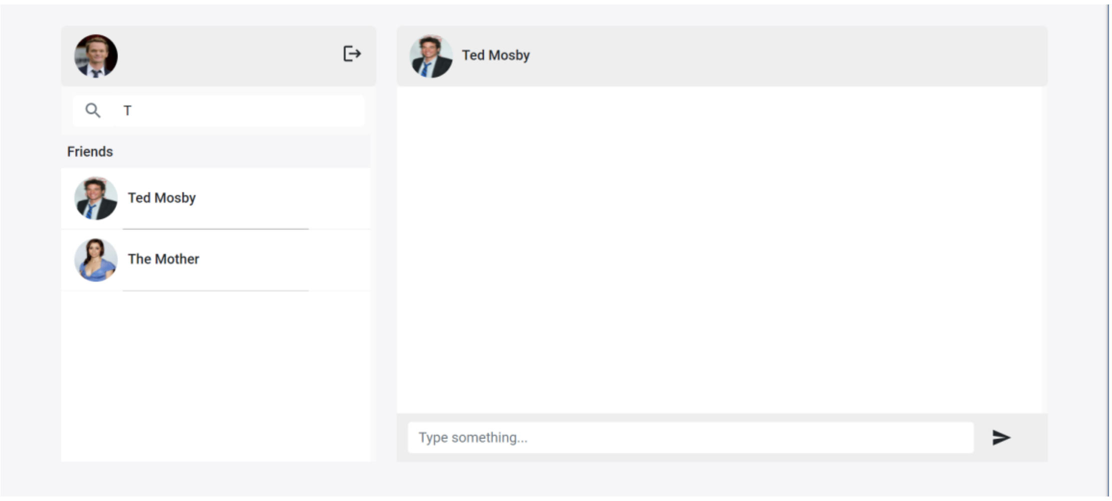
The user can search for new friends to talk to!

Let's say that the user, Barney, wants to chat with some new friend, Ted.
He searches him up and get all the users that registered that their name starts with the letter 'T'

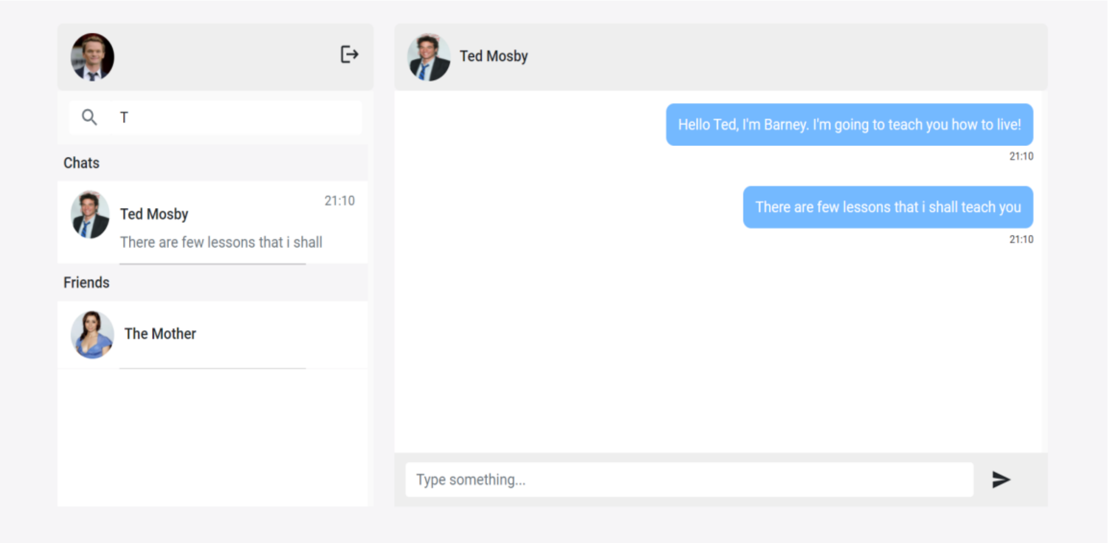


Open a conversation to start a chat or search for new friends!

Then he clicks on the Ted Mosby contact and a chat appears



In the chat, Barney can write something and send it to Ted



T

Chats

Ted Mosby 21:10
There are few lessons that i shall

Friends

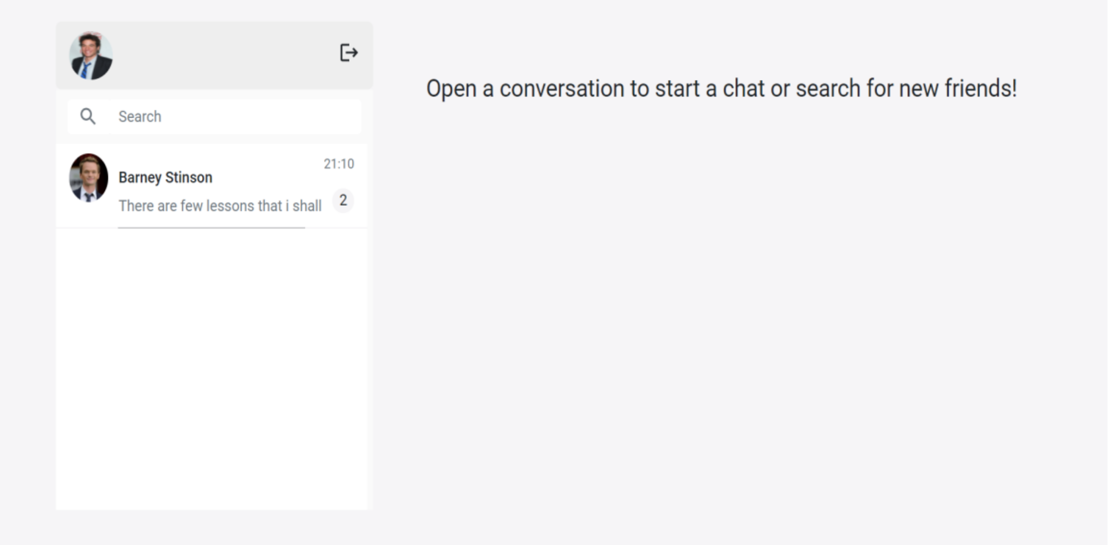
The Mother

Ted Mosby

Hello Ted, I'm Barney. I'm going to teach you how to live!
21:10

There are few lessons that i shall teach you
21:10

On Ted client, if he logs in, he will see the message appear on his conversations tab:

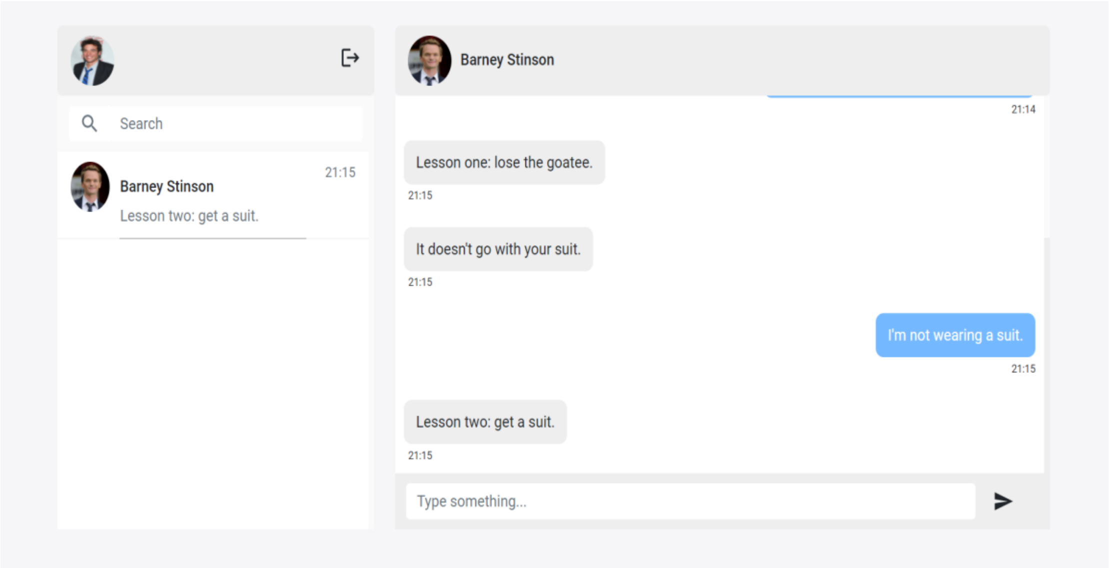


Barney Stinson 21:10
There are few lessons that i shall 2

Open a conversation to start a chat or search for new friends!

New 2 messages from Barney Stinson!
While two users are connected and chatting, their chat will update on real-time thanks to Socket.io, library for sending real-time data between users.

The chat state after ted responded (on Ted side client)



Barney Stinson 21:15
Lesson two: get a suit.

Barney Stinson

21:14

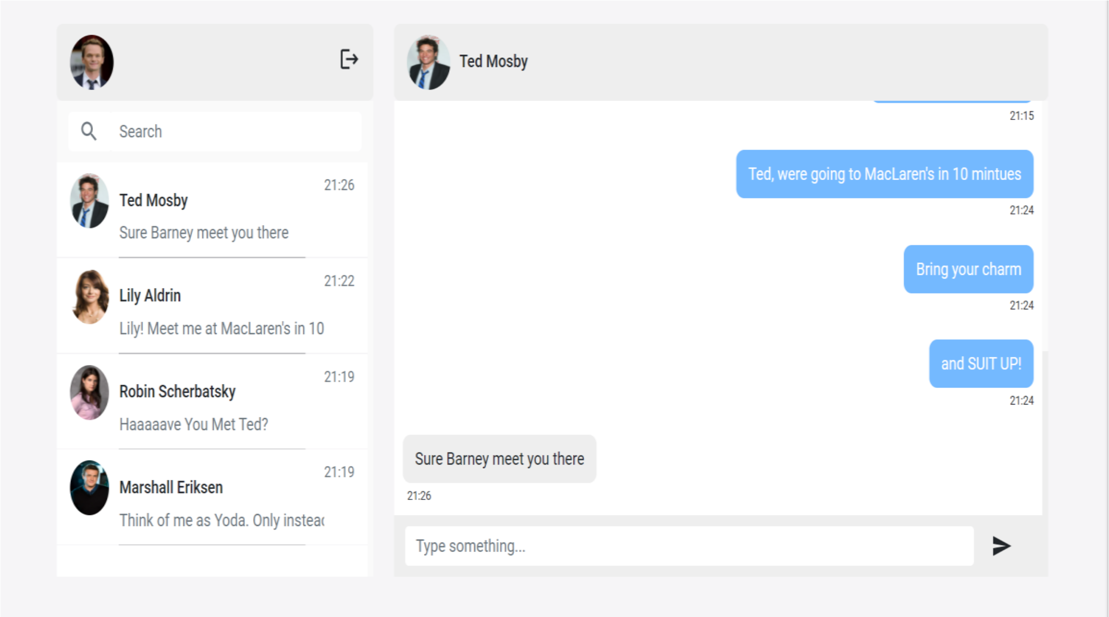
Lesson one: lose the goatee.
21:15

It doesn't go with your suit.
21:15

I'm not wearing a suit.
21:15

Lesson two: get a suit.
21:15

Then Barney can initiate more conversations with friends by searching them. Here how it looks after few conversations are up:



Ted Mosby 21:26
Sure Barney meet you there

Lily Aldrin 21:22
Lily! Meet me at MacLaren's in 10

Robin Scherbatsky 21:19
Haaaaave You Met Ted?

Marshall Eriksen 21:19
Think of me as Yoda. Only insteac

Ted Mosby

21:15

Ted, were going to MacLaren's in 10 mintues
21:24

Bring your charm
21:24

and SUIT UP!
21:24

Sure Barney meet you there
21:26

System Components and Security Notes

Secured Chat consists of backend server, frontend client, and 2 databases: MongoDB and Firebase.

Libraries That I Used for Security Reasons

bcrypt: Used this library to encrypt and hash passwords
https://www.npmjs.com/package/bcrypt

jsonwebtoken: Used this library to grant json web token to a user when he is logged in and verify he is the person sending the server request.
https://www.npmjs.com/package/jsonwebtoken

CryptoJS: Used this library to apply AES Encryption on the messages.
https://www.npmjs.com/package/crypto-js

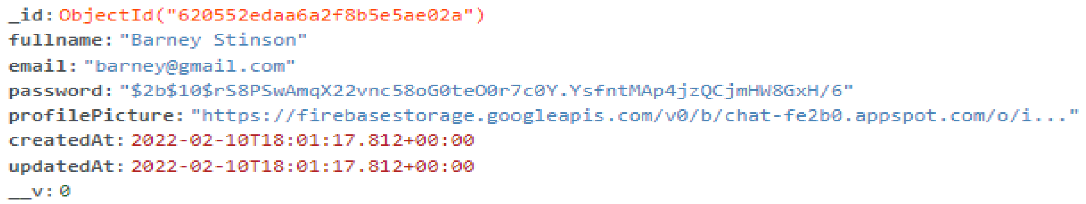
helmet: Used this library to secure Express server.
https://www.npmjs.com/package/helmet

The Backend Server and MongoDB

The server is a NodeJS and Express server. It runs on the Heroku deployment.
The server uses 4 routes: users, messages, conversations and verifyToken.
The server serve the client requests through the routes.
In the users route, the server handles the following events: registration, login, searching for user.

Registration: the user submits his name, email address, desired password and potentially profile picture and make a request to register. The server gets the request, validate the data according to basic rules (email is in email format, password length is at least 6).
If the data is validated, the server encrypts the password of the user using **bcrypt** library, a well-known library to encrypt passwords, and adding the user to the MongoDB database. In the process, the profile picture is being uploaded to the Firebase database.

Here's how the data looks inside MongoDB after registration:



```
_id: ObjectId("620552edaa6a2f8b5e5ae02a")
fullname: "Barney Stinson"
email: "barney@gmail.com"
password: "$2b$10$rS8PSwAmqX22vnc58oG0teO0r7c0Y.YsfntMAp4jzQCjmHW8GxH/6"
profilePicture: "https://firebasestorage.googleapis.com/v0/b/chat-fe2b0.appspot.com/o/i..."
createdAt: 2022-02-10T18:01:17.812+00:00
updatedAt: 2022-02-10T18:01:17.812+00:00
__v: 0
```

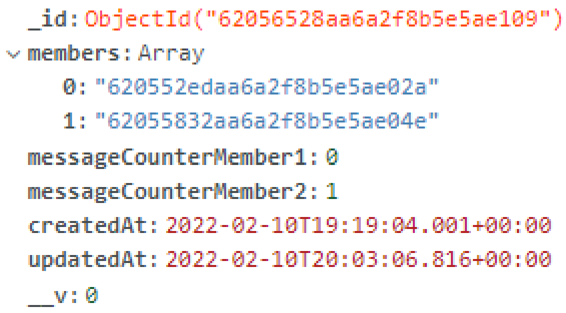
As you can see, the password is saved encrypted in the database.
The profile picture is a URL pointing to the place of the profile picture the user picked is stored at the Firebase database.

Login: the user submits his email and password at the login page and send the data to the server. The server validates the data in the database, and if the email and password matches the email and password that are stored in the database, the user can get access to the chat.
If the credentials are correct, the user is assigned with **JWT** for further actions.
The **JWT** is the key that giving him access to make request to the server that require authorization, such as sending message to another user.
The token is stored in the local storage of the browser to prevent users from logging in again for each session.

Searching*: the user can search for new friends. The server sends back list of all the users that their name starts with the user's input in the search field.

In the messages and conversations route, the server handles the following events: creating new conversation, update certain conversation, creating new message according to specific conversation.

Creating new conversation*: the user can search for friend and if he chooses to send a message to a new friend that is not in his conversation list, a new conversation request is send to the server.
Here's how the data looks inside MongoDB after creating new conversation:



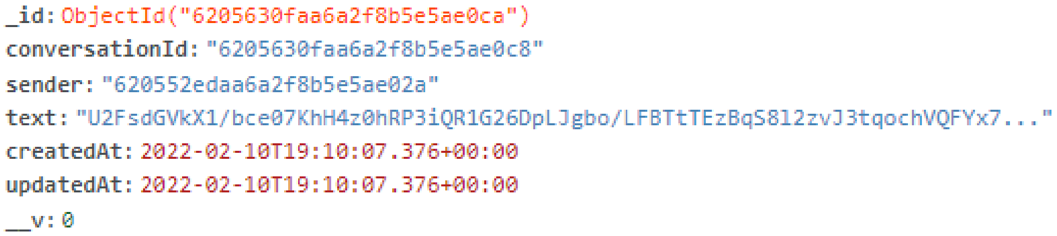
```
 _id: ObjectId("62056528aa6a2f8b5e5ae109")
v members: Array
    0: "620552edaa6a2f8b5e5ae02a"
    1: "62055832aa6a2f8b5e5ae04e"
  messageCounterMember1: 0
  messageCounterMember2: 1
  createdAt: 2022-02-10T19:19:04.001+00:00
  updatedAt: 2022-02-10T20:03:06.816+00:00
  __v: 0
```

The members array holds the users that are members in this conversation.
The message counters are in Conversation model also.
In this case, user1 sent to user2 a message, and then the message counter of member is set to 1.

Creating new message*: the user can send message to friends or even in a new conversation. The message is then encrypted using **CryptoJS** with AES Encryption using a secret key that stored in the .env secret file in the server.

Note on encryption on server side: I assume that my website with funds will use SSL and TSL to encrypt the request from the client to the server, so the encryption happens only on server side.

Here's how the data looks inside MongoDB after creating new message:



```
_id: ObjectId("6205630faa6a2f8b5e5ae0ca")
conversationId: "6205630faa6a2f8b5e5ae0c8"
sender: "620552edaa6a2f8b5e5ae02a"
text: "U2FsdGVkX1/bce07KhH4z0hRP3iQR1G26DpLJgbo/LFBTtTEzBqS812zvJ3tqochVQFYx7..."
createdAt: 2022-02-10T19:10:07.376+00:00
updatedAt: 2022-02-10T19:10:07.376+00:00
__v: 0
```

The database contains who send the message, the conversation id and the text of the message **ENCRYPYED**.

Update certain conversation*: when the user sends message, there needs to be update on the conversation updating the counter of the messages. This event happens also when a user enters a conversation, resetting the counter to 0.

In the verifyToken route there is a single authorization function that function as middleware the gets the JWT from the user when he makes the request, verify that the JWT is legit, and if so, continues to the next function (processing the real request).

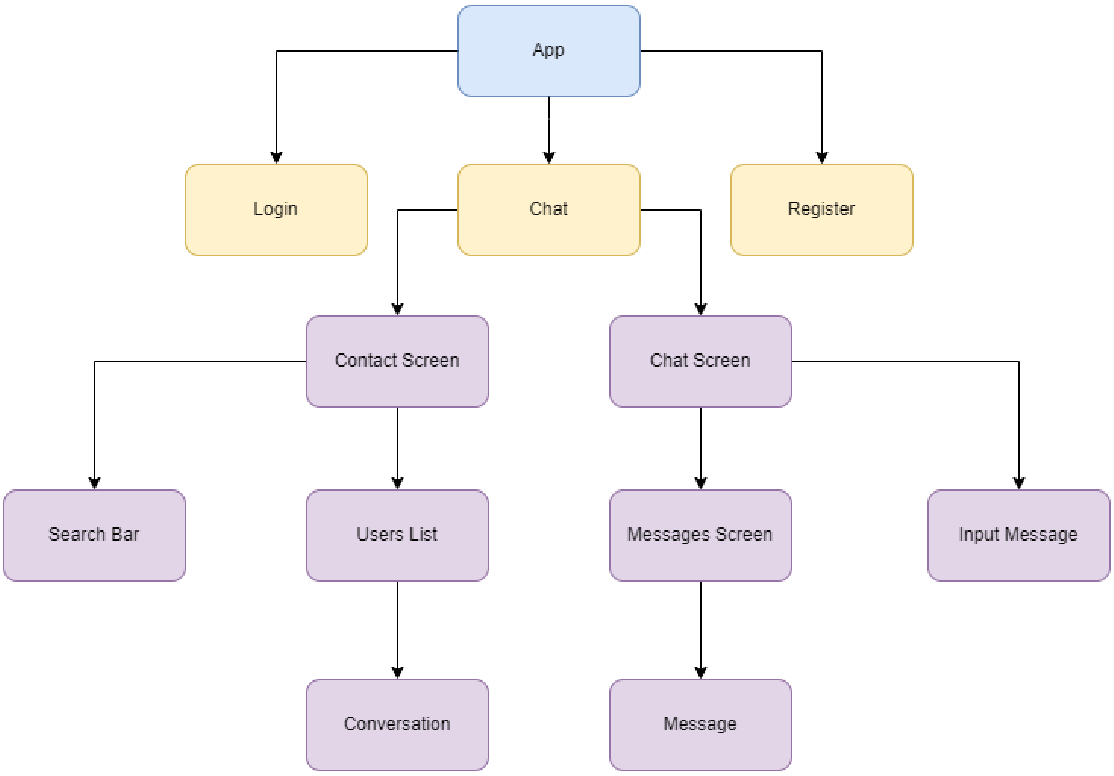
*** This request require authorization meaning the server first check if the user is verified (his JWT) through the verifyToken route, and if he is verified, he serves his request.**

The Client

The client is written in React framework, using JavaScript, html, CSS , bootstrap 5.

Secured Chat client consists of 3 main pages: Register, Login and Chat.

Components Tree:



App: main page in React application.

Login: login page as shown in the picture in page 1.

Register: register page as shown in the picture in page 2.

Chat: the whole page as shown in the picture in page 2.

Contact Screen: The left side of the screen in the Chat page. Black rectangle in the picture below.

Search Bar: The search input in the left side of the screen in Chat page. Red rectangle in the picture below.

Users List: The list of conversations in the left side of the screen in Chat page. Green rectangle in the picture below.

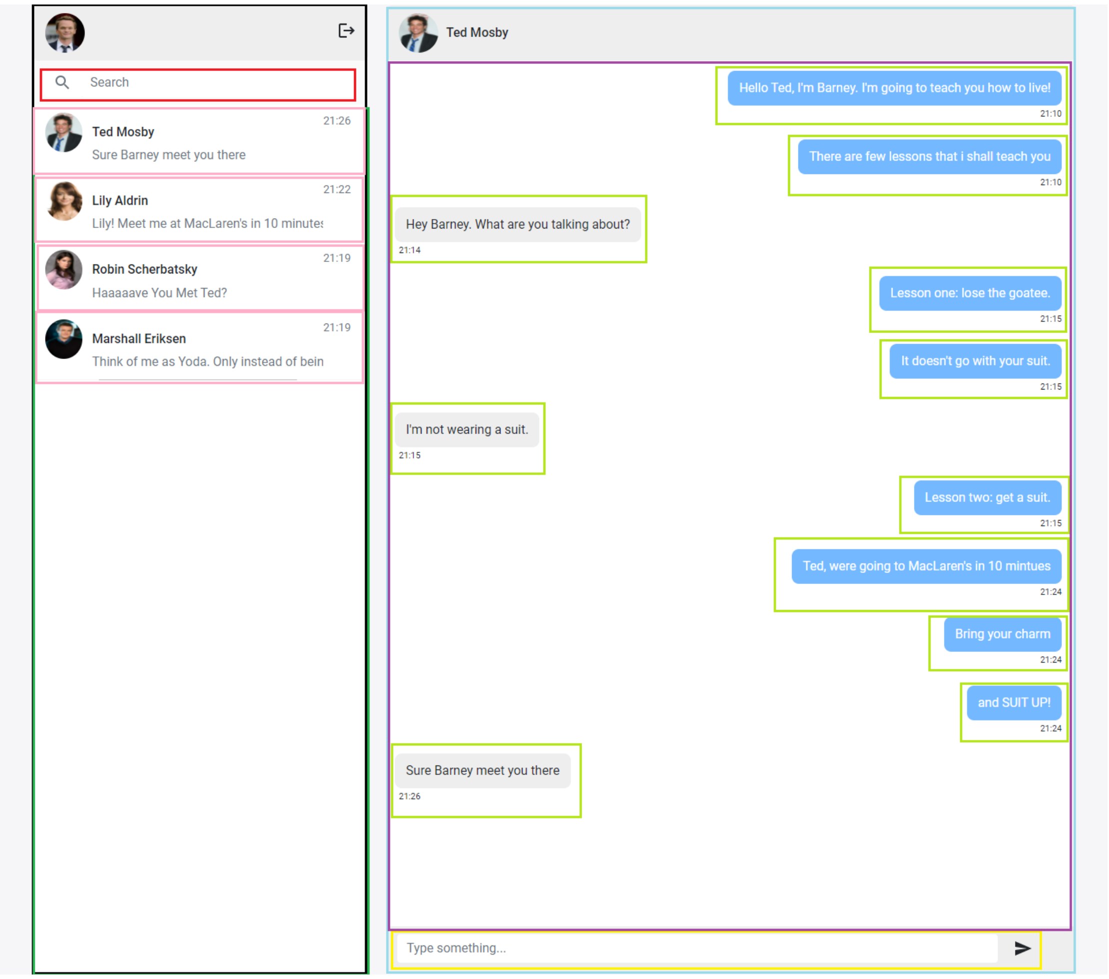
Conversation: Holds the name of the friend conversation, time of last message and last message preview. Pink rectangles in the picture below.

Chat Screen: The right side of the screen in the Chat page. Light blue rectangle in the picture below.

Message Screen: Contains list of all the messages in selected conversation. Purple rectangle in the picture below.

Message: Holds the text of the message and time that the message has been sent. Light green rectangles in the picture below.

Input Message: The input field for sending messages. Yellow rectangle in the picture below.



Search

Ted Mosby 21:26
Sure Barney meet you there

Lily Aldrin 21:22
Lily! Meet me at MacLaren's in 10 minute:

Robin Scherbatsky 21:19
Haaaaave You Met Ted?

Marshall Eriksen 21:19
Think of me as Yoda. Only instead of bein

Ted Mosby

Hello Ted, I'm Barney. I'm going to teach you how to live! 21:10

There are few lessons that i shall teach you 21:10

Hey Barney. What are you talking about? 21:14

Lesson one: lose the goatee. 21:15

It doesn't go with your suit. 21:15

I'm not wearing a suit. 21:15

Lesson two: get a suit. 21:15

Ted, were going to MacLaren's in 10 mintues 21:24

Bring your charm 21:24

and SUIT UP! 21:24

Sure Barney meet you there 21:26

Type something...

apiCalls file: the client send request to the server using axios framework, all the API calls are in the apiCalls.js file.

Challenges While Development

Throughout the experience of developing SecureChat I encountered several challenges:

- Creating the UI.
- Develop real-time chat features.
- Learning and using cloud databases.

Pros and Cons of SecureChat

Pros:

- Minimalized and clean UI.
- Efficient.
- Real-time chat features.
- Modern chat features.
- The messages are encrypted using AES encryption.
- The data is stored in secure database: MongoDB, and Firebase.
- The user password is encrypted.
- The server authenticates the user and assigning him JWT for authorization.

Cons:

- SecureChat is not responsive yet, not available for mobile phones.
- No option to send images or videos.

Future Features

In the near future I plan to add some more features that I haven't added yet such as:

- Making the site responsive and available for mobile phones.
- Add option to send images and videos.
- Add option to send emojis.

Summary

I had a lot of fun to implement this project, I learned a lot on security topics and developing real-life application. All the goals in the project proposal were met.