**BACK-END**

The backend is an Express server with a Mongo database managed with mongoose. The server serves a build version of the React frontend. All server response routes have the prefix /api.

**Socket IO**

Socket.io was used for bidirectional communication between server and client, which was used for chatting feature and notifications. You can find more information about socket.io on their website: <https://socket.io/>

socket.on("joinUserSocket", userId)

Join a channel with a name userId (so there is a private channel for each user).

socket.on("newConversation", {conversationData})

Send a new conversation with conversationData to users with ids conversationData.userIdOne and conversationData.userIdTwo. conversationData users must be in following format:

{

"\_id": "5fc691bd1c348e3b545f0e3e",

"userIdOne": "5fcd126e0dc8721dc48c2176",

"userIdTwo": "5fcf17f70cd8754dc4dad067",

"userNameOne": "John, Doe",

"userNameTwo": "Jane, Doe",

"conversationType": "date",

}

socket.on("join", {conversationId, userId})

Join a channel with a name conversationId (i.e., establish a channel, so two users can chat). The method assumes that conversationId and user\_id is valid. Moreover, send the last messages from the chat with id conversationId to the user with id userId (sends maximum of 10 messages).

socket.on("sendMessage", {content, author})

Send a new message with the author author and with the content content to all two users connected to the channel (the method assumes that socket.on("join") was already called, so the user already connected to the channel for chatting, hence conversationId is already available, so socket only sends information to two users).

socket.on("latestMessages", {\_id, callback})

Send the latest messages (maximum of 10 messages) from the chat to the user who request for latest messages (the method assumes that socket.on("join") was already called, so the user is already connected to the channel for chatting, hence conversationId is already available, so messages can be retrieved from chat with id conversationId).The send messages are earlier than the message with id \_id (the method assumes that \_id is valid, that is, the message with such id already exist in the chat.

**Conversations**

api/conversations/:user\_id

Method(s): GET

Gets all chats’ information of the user with id user\_id. The method assumes that user\_id is valid, that is, the user with such user\_id exists. If the user with user\_id was successfully found, the response's status code is 200 and the response’s data is JSON format as a list of chats:

[

{

"\_id": "5fc691bd1c348e3b545f0e3e",

"userIdOne": "5fcd126e0dc8721dc48c2176",

"userIdTwo": "5fcf17f70cd8754dc4dad067",

"userNameOne": "John, Doe",

"userNameTwo": "Jane, Doe",

"conversationType": "date",

},

...

]

Otherwise, the response’s status code is 500 and and the response’s data is JSON format in form:

{

"errors": [

{"msg": "Something went wrong, try again."}

]

}

api/conversations/:conversation\_id

Method(s): DELETE

Delete the chat from the database with id conversation\_id. The method assumes that conversation\_id is valid, that is, the chat with such conversation\_id exists. If the chat was successfully found and deleted, the response's status code is 200 and the response’s data is JSON format in form:

{

"msg": 'Conversation was deleted"

}

Otherwise, response’s status code is 500 and and the response’s data is JSON format in form:

{

"errors": [

{"msg": "Something went wrong, try again."}

]

}

**User**

/api/auth/login

Method(s): POST

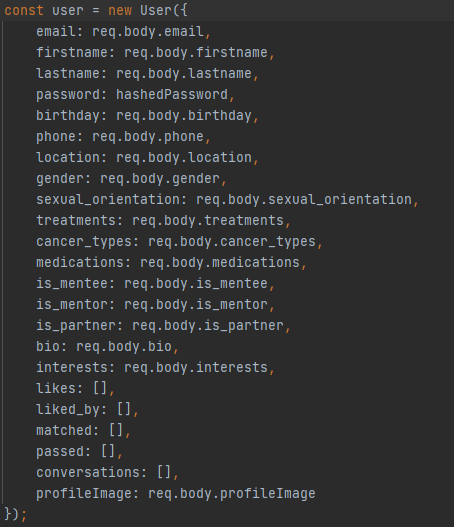
Body: JSON Object containing password and email

Checks the login credentials against database and returns a JSON object containing the userId and accessToken if they match, if not it returns an error.

/api/auth/signup

Method(s): POST

Body:



Registers a new user by taking the input fields (the fields required for the user model) and adding a user to the database, also checks to see if the email is unique.

**Data**

/api/data/cancer-data

Method(s): GET

Gets data that is used later for options in registration and filtering. Most of this data was provided to us by the client.

/api/data/interests

Method(s): GET

This retrieves the data that is used in the interests field in the registration page, this can be used to filter users and find possible matches according to users’ interests.

**Other**

/api/user/:userId

Method(s): GET

Userid: ID (from database)

Finds user in database from the provided userId.

/api/suggestions/:userId

Method(s): GET

Userid: ID (from database)

Returns users based on cancer types and filters.

/api/matches/connect/:currentUser&:UserthatwasLiked

Method(s): POST

currentUser, UserthatwasPassed: IDs (From database)

Called when currentUser presses connect on UserthatwasLiked, there is a match created between them when both of them like each other. This call also adds respective ids’ to their respective lists (liked, liked\_by) user lists for both users.

/api/matches/pass/:currentUser&:UserthatwasPassed

Method(s): POST

currentUser, UserthatwasPassed: IDs (From database)

Called when currentUser passes on UserthatwasPassed, this action is then reflected on respective user lists (passed).

/api/match-by-location/:uid

Method(s): GET

uid: ID (from database)

Find other users that can be shown to current user for them to connect / pass on. Returns a list of all possible user ids, where the first ids are of those that have liked the current user and ones after those are based on location (users closest to current user within 30 kilometers).

/api/admin

Method(s): GET

Body: Includes all conditions and their filter requirements in a json format.

Possible conditions are: gender, sexual Orientation, cancer types, treatments, medications, isMentor (boolean), isMentee (boolean), isPartner (boolean), age range [lower, higher]

Only add conditions that need to be filtered by in the json object

Example:

{

gender: “male”,

treatments : [“A”, “B”]

}

Filters the database based on the conditions given and returns an array of emails that match those filters.