Common JS Messenger

Description

“Common JS Messenger” has to be stand alone AngularJS (directive) that will be easy to implement in any javascript application its needed.

The chant and its all related windows will be floating and only a chat icon will be visible in the application view.

User will open the chat window by click on the icon and the chat window will appear, as a custom draggable modal panel.

Basic features will include :

- User to user chat
- Group chat
- Chat rooms (defined by the administrators and custom rooms)
- File sharing capabilities
- Quick image sharing (with preview in the chat)
- User search
- String search (in specific chat session and common search among all chat sessions for the user)
- Text formatting (WYSIWYG editor with text formatting capabilities like **bold**, *italic*, coloring)
- Rich Notifications (see https://developer.chrome.com/extensions/desktop_notifications)

Front end

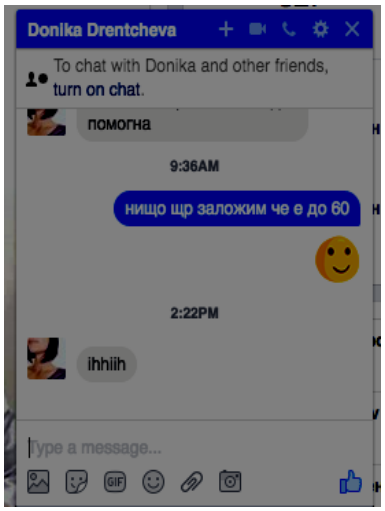
Technologies

- AngularJS
- Angular Material

UI

Feel free to improvise with angular material, but keep it simple and don’t spend too much time on the design.

Example chat window:



UX

There are several important elements that should exists:

Users/rooms search box:

- single line (<input type="text">) with autocomplete function for searching among the users and the chatrooms.

Message input box (with send button) – the place where user types the text

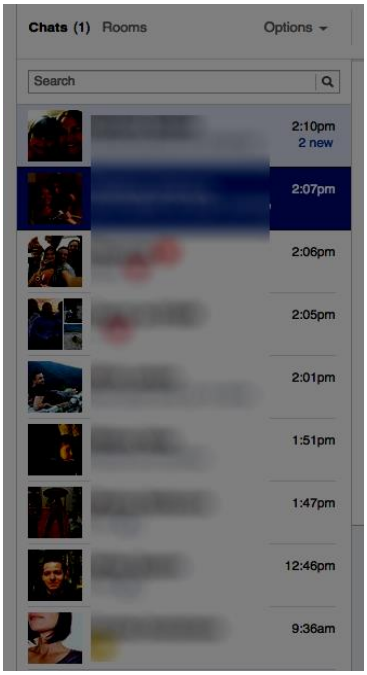
- See examples on https://material.angularjs.org/latest/demo/input
- single line (<input type="text">) at the beginning.
- becomes multiline (WYSIWYG) when the text is longer than # symbols OR the user uses **Ctrl+Enter** (shortcut) to start typing on second line
- user will send the message by hitting **Enter** key on the keyboard OR clicking on the button which will be next to the input.

Messages Log – this is the place where the chat messages will appear:

- Every message might be as ul > li element
- Newest message at the bottom
- When the chat window is opened it will display the last 10 (configurable option) messages and when user scrolls up load another 10 and so on and so – infinitive scroll.
- Format "USERNAME DATE : MESSAGE"
- If message is too long limit the displayed size to 5 lines and add ellipsis and button for expanding the message (check for "css multiline ellipsis")
- Mark message as favorite for quick access.

Chat and rooms browser:

- Side navigation menu (click here) that can be displayed by clicking a button (or moving the mouse pointer to side of the screen) and will be used to display all of the user's chat sessions (like the one in the facebook – see the image below)



Back end

Create RESTful web service with JSON media type responses for non real time activities such a searching thru messages, searching for users, autocomplete, retrieving the old messages in the infinitive scroll, submitting options etc. and use sockets for the chat sessions.

Use NodeJS server for the development purposes and use sockets for send/receive messages.

Instead of using database, write directly in files.

Technologies

- Nodejs
- Express
- Socket.io

References

Check these links for help, ideas and inspiration :

Front

- https://github.com/vogloblinsky/angular-simple-chat
- https://www.facebook.com/messages/
- https://developer.chrome.com/extensions/desktop_notifications
- https://www.tinymce.com/

Back

- http://jsonapi.org/
- http://expressjs.com/
- https://nodejs.org/en/

API

POST/GET SOCKET	Name	Params	Sample Response	Error	Desc
GET	currentUserInfo	accessToken – currentUser accessToken	{ fullName : “Testi Testov” , username:’qwerty’ }	Returns object with error message	Executed onInit() will return current user details

GET	fullNamesByString	searchString – searched user name	[
			{
			fullName:”Ivan Ivanov”,
			Id: 34
			},
			{
			fullName:”Dragan Ivanov”,
			id: 14
			}
			]
				Returns object with error message	Will search for matches in first and last names of all users

GET	chatHistoryBrief	accessToken – current user access token	[
			{
			id: 12345678,
			creatorId: 8910112313,
			isRoom: false,
			lastChatMessageText: ’Zdr kp’,
			lastChatDate: 2017-02-21T13:14:36.864Z,
			lastChatSender: {
			fullName:"Testi Testov"
			id:"589dbe874a1dad6d2e933fce"
			},
			participants: [{fullName: “Pesho”, id: 12345567}],
			userId: 454674746764
			},
			…
			]
				Returns object with error message	will return array of chat history notes () ORDER BY DATE DESC

GET	chatHistory	userID– id related to the specific room
		
		lastMessageId – last loaded messageID (so we can load 20 more for pagination)
			[
			
			{
			
			messageId::string,
			
			authorName::string,
			
			messageDate::Date,
			
			messageBody::string,
			
			messageType::int(
			plain text = 1, image = 2, file = 3)
			
			}
			
			]
				Returns object with error message	will return array of chat messages with last messages – will be used for infinitive scroll ….. chat history messages () ORDER BY DATE DESC

GET	messagesBySearchstring	searchstring – searched string
		chatID
			[
			{
			messageID : 23423
			}
			]
				Returns object with error message	will return array of chat notes () based on searched string ORDER BY

					DATE DESC
GET	chatIdForUsers	Array of participant users IDs [2,4,7,8]	“f4983fj843jf0394kf43” (chat ID)	Returns chat id STRING (or int)	Will return the chat ID for these participants OR false if there is no previous session with same participants

POST	createChat	{
		Creatorid : ‘id’,
		Participants : [
		“part1id”,”part2id”
		]
		}
			“f4983fj843jf0394kf43” (chat ID)	Returns chat id STRING (or int)	Will return the newly created chat id

POST	login	username password	{accessToken: 1234}	Returns object with error message	
SOCKET ON	message	uhatid (string) userid (string) message (text)	will store the message to the DB and will emit : new message new message notification		