Mandatory Exercise 1 - Advanced JavaScript with React

The deadline for this exercise is Friday, March 15, 08:59.

For this **mandatory exercise** you should work on **master branch only**.

Preparation

1. Create a new repository on GitHub called **mandatory-advanced-js1**.
2. Follow the instructions that GitHub gives you; Create a local repository and add a remote or clone the newly created repository.

Submission

When you submit the exercise in PingPong, before the deadline, you will enter a link to your repository, such as:

https://github.com/mygithubusername/mandatory-advanced-js1

The teacher will look in the **master branch**. If any commits are done to the branch after the deadline, the teacher will look at the last commit before the deadline.

You will get one of the grades **G** or **IG**.

Instructions

In this exercise you will create a simple chat application using React. The backend server is provided.

Socket.io

Socket.io is a library that enables real-time communication between a client and a server. The client and server communicates by sending and receiving events.

Please refer to the documentation for more information.

https://socket.io/docs/

Socket.io server

A socket.io server is provided at
http://ec2-13-53-66-202.eu-north-1.compute.amazonaws.com:3000

This URL could change. Ask the teacher if you are unable to access it.

Events

The server sends the following events

- `messages` - This event is sent automatically when a client connects and will give a list of all messages on the server
- `new_message` - This event is sent to all clients (except the sender) when a new message is sent to the server

A message sent from the server has the following structure

```
{
  username: "A username",
  content: "A message",
  timestamp: 1551191228686, // A timestamp in milliseconds
  id: "message-120", // A unique ID
}
```

To send a new message to the server an event called "message" is sent from the client. The message should have the following form:

```
{
  username: "A username",
  content: "A message",
}
```

The client

Your task is to implement a client for this server. The client should be implemented using React and contain two views:

- A "login" screen where the user inputs a username
- A "chat" screen that shows all the messages and contains a text input field where the user can add new messages

When sending a new message to the server it will not be returned to the sender (it's only sent to all other connected clients) but it must still be shown in the client.

Validation

The server has some limitations on the username and content

- The username can only contain alphanumeric characters, "-", "_" and spaces and must be between 1 and 12 characters long
- The content must be between 1 and 200 characters long

This validation should be added to the client.

Emojis

The client should handle emojis, similar to how it works in Slack.

E.g. if the message contains the string ":heart:" it should be replaced with a heart emoji. You are required to support at least three different emojis.

Links

If a message contains a URL it should be automatically converted to a link.

Requirements

The client has the following requirements

- It should contain a "login" screen with a text input and a button. This screen is shown when the app is opened.
- When a username is submitted the user is shown a "chat" screen with a text input for sending messages and a list of messages
- The message list should be populated with the messages received from the server and new messages should be added automatically (including messages sent from the client)
- The chat screen should contain a "close" button so the user can return to the "login" screen
- Support for emojis
- Automatically convert URLs into links

Tips

- Try to connect to the server and play around to see how it works with a simple Node program before attempting to integrate it with your React application
- Use regular expressions (regex) to find URLs in the messages
- Ask questions if there is something you are unsure about