challenge08.md

Procedure

Follow the instructions below. You can download them as pdf here. Build upon your solution from the previous challenge and upload everything for peer assessment after you finish.

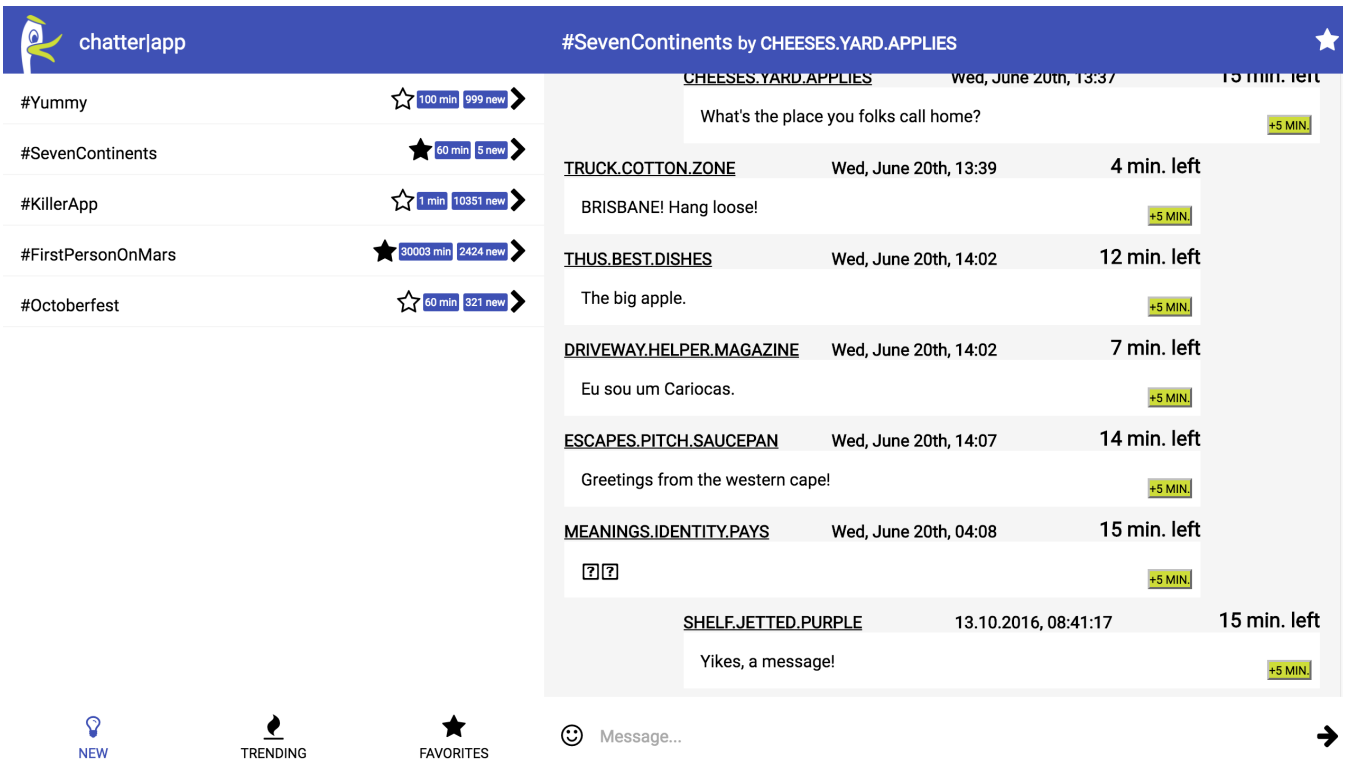
There are different exercise types:

- These exercises are important and you should tackle them
- (*) These asterisk-marked exercises marked are a bit more difficult and thus voluntary. They will improve your skills, tackle them only if you want an extra challenge. However, your app will also work without fulfilling the instruction.

Prioritize the challenges. Don't spend too much time on the voluntary challenges!

Challenge Goal

This is how your app looks like after this **challenge 8**.



It doesn't look very different to what you already have, because this sprint mainly focuses on interaction.

Graded Criteria

`#message & #input` The text message, typed into the input field, appears in the chat view, when clicking on "send". All properties are passed on.

- 3 Pt. :createdBy: Location is retrieved from what3words.
- 3 Pt. :createdOn: e.g. the current date is displayed.
- 3 Pt. :expiresIn: e.g. 15 minutes into the future.
- 2 Pt. :text: Actual text message, typed into the input field is passed on.
- 1 Pt. Message structure is comparable to solution.
- 1 Pt. Send button triggers, that input text and properties are sent.

`#input` Input field positioned correctly, contains prefill, text length is limited and message cleared after send.

- 1 Pt. Input field contains prefill.
- 1 Pt. Text length limited.
- 1 Pt. Input field positioned between smiley and send button.
- 1 Pt. Message is cleared after send.

`#channel` Channel list is created dynamically onload.

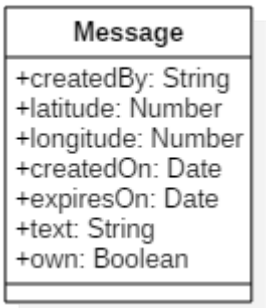
- 2 Pt. Each channel appears in the channel list dynamically. Instead of coded HTML list, channel list is dynamically appended to the
  .
- 2 Pt. Channel names are loaded dynamically, that is, names are loaded from channels.js
- 2 Pt. Channel stars are loaded dynamically. They display whether :starred: is false or true.
- 1 Pt. Channels contain a chevron.

Your `#syntax` will be graded automatically. Overall, 24 Points (Pt.) can be achieved.

Instructions

Yikes! A `#message`!

- When creating new chat messages, we need to send them in object-form to our server later on. Write a `#constructor` function `Message(text)` to create new messages conveniently.

  Messages are based on the following model:
  - `createdBy`, `latitude` und `longitude` are assigned the respective values from the global variable `currentLocation`.
  - `createdOn` is the current date which you get with `Date.now()`.
  - `expiresOn` is a future date: in 15 minutes. Google how to set a future Date in JavaScript.
  - A message's text is passed via the (only) text parameter and is stored in the object's respective property.
  - Finally, `own` is simply `true`, since we'll only create own messages via the `Message()` constructor - all other messages will come from the server later on.
- Add a `#send` button to your chat bar, including a font-awesome arrow pointing to the right. Create a `sendMessage()` function and attach it to the `#send` button's onclick event. Create a `new` message object within the declaration of `sendMessage()`, for now with the **text** "Hello Chatter", using the keyword `new` and your `Message()` constructor. Log the created message object.

We need those message objects to post them to the server and thereby to other chatter|app users. We also need to keep them in memory (e.g., a global variable) to modify them. Chat messages will change - not their text, but their properties such as time out (hence they have a time to live), we need their identification to extend their time to live, and so on.

But besides the chat message objects, we do also need their element representation, since we want to render them in the browser.

- Let's target the latter and get prepared to write them into our HTML document. Write a `createMessageElement(messageObject)` function, which takes a message object and returns a `String` representation of an HTML message `#element`. It basically shall make an HTML element, like the one below, out of message objects (which we then can easily add to the container's innerHTML). The words marked with `::` are placeholders, we tackle them later.

```
<div class="message">
    <h3><a href=":createdBy:" target="_blank"><strong>:createdBy:</strong></a>
        :createdOn: <em>:expiresIn: min. left</em></h3>
    <p>:text:</p>
    <button>+5 min.</button>
</div>
```

- Now instead of placeholders we want to pass an actual message object. Replace the : placeholders: in the return string of `createMessageElement()` with the `#properties` of the received messageObject. Do not change `:expiresIn:`, yet, leave it for the next task. You will need to format the date locally.
- Before you can use `:expiresIn:`, calculate a corresponding variable based on *expiresOn*. Learn, how to calculate with dates, since they are also numbers, namely milliseconds. The Math.round() method may help.
- Call `createMessageElement()` in `sendMessage()` and `#append` it to the end of your #messages div using $.append().
- (*) Right now, new messages are hidden from view. They disappear at the bottom end. Ideally, the #message div should scroll to the bottom. after sending a new message. Try using the (counterintuitive) method $.scrollTop() to `#scroll` the #messages div to the bottom.
- (*) Find a way to add the `#own` class dynamically to messages, depending on the message object's property?

More fun with some `#input`

So, let's ramp up a real messaging system!

- Add an input `#element` between emojis button and send button where the user can edit text.
- `#prefill` it with the placeholder text "Message..."
- `#limit` the input's text length to 140 characters (in HTML)
- (*) Apply `#styles` to
  - remove the input element's border
  - adjust its font size to the one of the messages
  - let it consume the full available chat-bar space (tipp: flexbox).
- Rework your `sendMessage()` so that it reads the input element's value (don't hesitate to assign it an id) and creates a new message based on the `#real` user input.
- `#clear` the input after the message has been sent.

You should now be able to type new messages and write them to the messages container.

Some more `#channel` manipulations

We'll now dynamically build our channels list from the JavaScript objects in the `channels.js`. Creating the corresponding HTML elements works pretty much like our `createMessageElement()` but this time we will use jQuery.

- Write a function `listChannels()` and let it append a new `<li>` to the channel's `<ul>`. Call `listChannels()` in the body's `#onload`. Our approach was to call `listChannels()` in the body's `onload`, because, as you just learned in section 8, it is not always possible to dynamically add HTML elements to others, if the other HTML has not been loaded yet. This problem might also occur when you want to add `<li>`s to your `<ul>`.
- Create a new `createChannelElement(channelObject)` function and call it in your `listChannels()` five times, once for each channel. Similar to `createMessageElement()`, it shall create a `#new` channel (list) elements by taking advantage of jQuery's element building features.
  Return this jQuery object to the caller function, i.e. `listChannels()`. Append the returned element to the `<ul>` in `listChannels()`.

- `#Remove` the current (static) channels `<li>`s from your HTML document.
- (*) Dynamically add two little `#boxes` to the channel's meta pane between star and chevron in `createMessageElement()`. Those should display the channel's properties **messageCount** and **expiresIn**. This task requires creating the boxes using jQuery and styling them in css:

  - style/address them without assigning additional ids/classes
  - primary-colored background
  - adjust margin and padding to resemble the example above
  - small text
  - a 2px border-radius.
- (*) Find a way to flex the blue boxes and their content to be perfectly `#centered`?

`#cleanup`

- Structure and comment your CSS.
- Check code & syntax. Using W3C Validator helps.

Done?

Do not forget to **save** your work, **push** it to Github, and paste the URL in the submission mask.