challenge10.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.

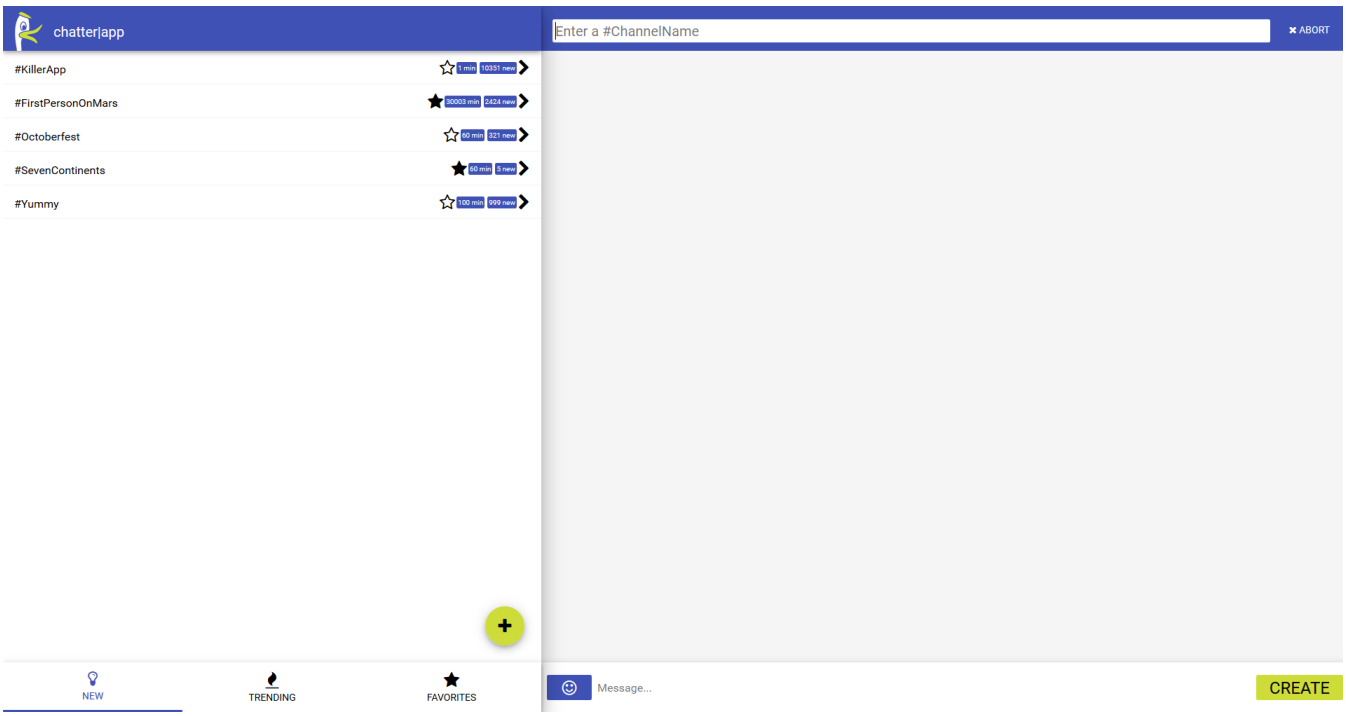
There are different exercise types:

- 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 voluntary challenges!

Challenge Goal

This is how your app looks like after **challenge 10**:



You'll mainly work on channels in this challenge.

Graded Criteria

`#array#for` Channels are stored efficiently and appended with a loop.

- 1 Pt. Channels are stored in a global array.
- 1 Pt. Channels are appended to the channels list with one loop.

`#sort` `listChannels()` sorts channels according to "new", "trending" or "favorite" order.

- 2 Pt. Clicking **new** sorts channels by date.
- 2 Pt. Clicking **trending** sorts channels by number of messages.
- 2 Pt. Clicking **favorite** sorts starred channels before unstarred channels.
- 1 Pt. `listChannels()` does not duplicate channels.

`#emojis` All emojis are loaded.

- 2 Pt. **All** emojis are loaded "onclick".

`#message#empty` Messages with content are send and stored in a channel.

- 1 Pt. It is not possible to send an empty message.
- 2 Pt. sendMessage() shows message and pushes it into a channel object.
- 1 Pt. Sent messages are counted.

`#new` A new channel can be created with the floating action button.

- 1 Pt. Clicking the FAB clears displayed messages.
- 2 Pt. Input field and abort button appear in right app bar.
- 2 Pt. Clicking on the abort button restores the previously selected channel without messages.
- 1 Pt. Validity of new channel name and its first message is checked.
- 2 Pt. A channel object is created, stored in channels array and the currentChannel, and shown in channel list.
- 1 Pt. After successfully creating a channel, the app bar switches back to default and the new channel's data is shown (title, location, and the new message).

Your `#syntax` will be graded automatically. In total, 25 points can be obtained.

Instructions

An `#array` full of channels

- Your channels are currently stored in five different variables in the `channels.js`. Reference these objects in an array which is global and named `channels`. `#arr`
- In your `listChannels()`, use a `#for` loop to append the channels to the channels' list from the array, instead of adding all five channels separately.

`#Sorting` the channels

There are three criteria, we want to sort our channels by:

1. *new*: the descending **createdOn** date of channels
2. *trending*: the descending **messageCount** of channels
3. *favorites*: **starred** channels before **unstarred** ones

Luckily, we can `#sort` our channels array by these criteria. However, we need to define corresponding comparison functions. Imagine you want to sort two objects. How should JavaScript know which one comes first? Which property is important in what order? The comparison function, which you can define, will receive two arguments from `sort()`. Read the argument's properties and determine the order: If you return a value `< 0`, the first argument will be listed first, if it's `> 0`, the second. An example: Imagine you've got two ponies:

```
var ponyOne = {name: 'Lilla Gubben', size: 1.49};
var ponyTwo = {name: 'Jolly Jumper', size: 1.86};
function compareSize(ponyOne, ponyTwo) {
   if (ponyOne.size < ponyTwo.size) {
       return -1; //Pony one is smaller and should be sorted first
   } else {
       return 1; //Pony two is smaller (or equal) and should be sorted first
   }
}
```

Your compare function passes the correct sorting order to `array.sort()` . The same can be accomplished by:

```
function compareSize(ponyOne, ponyTwo) {
   return (ponyOne.size - ponyTwo.size);
}
```

This would sort an array of your ponies:

```
var ponys = [ponyOne, ponyTwo];
ponys.sort(compareSize);
```

- Create three `#compare` functions for sorting the channels by **new**, **trending**, or **favorites** in a **descending** order: New above old, more above less, and starred above unstarred. The function should only compare and not yet sort.
- In `listChannels()`, sort the channels array before writing it to the channels list by the *new* criterion. The '#Killerapp' channel should now be listed topmost in **new** and **trending**.
- Add a `#parameter` `criterion` to `listChannels()` and use it to sort the channels array.
- Call the function with a reference to the **new** sort functions (since this is the button selected by default) in the body's `onload` attribute. `#load`
- Also call `listChannels()` with the corresponding sort function in three new `onclick` actions on the tab-bar buttons while keeping the `selectTab()` functionality. `#click`
- Ensure that `listChannels()` doesn't `#duplicate` channels in the list, consider $.empty().

Many, many `#emojis`.

Copying and pasting the emojis from a website is painful. Yet we want them all. Luckily, Kikobeats has done all the work.

- Understand the provided example.html and inspect, download (right-click the file and save it), and `#embed` the emojis-list.js.
- Although it is not very efficient to load all emojis whenever the user clicks the button, `#add` all emojis from the array to your emojis menu when the user clicks `toggleEmojis()`. The example.html shows you how to load the array.
- (*) Find a more `#suitable` place to load your emojis (once).

À propos `#message`

- Currently, our user can send `#empty` messages. We want to block that. Use an if statement to test whether the input value is empty. You can use the string's length to test it.

The time has come to link channels and their messages. This is important for switching channels in the next sprints. To achieve this, each channel needs to know all its messages, so we'll store them in the corresponding channel's object.

- First of all, get rid of the static messages in the HTML file. `#Delete` them (or comment them out).
- Define a property `messages` in all `channel` objects and initialize these properties with empty arrays. We'll put future messages there. `#chl`
- In your `sendMessage()`, instead of only creating a message *element* ouf of the message *object*, now also Array.push the message object into your `currentChannel`'s messages array (the one you've created in #chl). `#push`
- Don't forget to count the number of messages in your channel! `#Increase` the `messageCount` property with every send message. Don't worry about the preset count, in challenge 12 this will be handled by the server.

I'd like to have a `#new` channel!

The following instruction set can be implemented individually. Hence, only the goal is described.

- Clicking the floating action button `#clear`s all messages in the container and swaps the right (messages) app-bar with one showing a text input (for the new channel's `#name`) and an abort button (see image of challenge goal above). In addition, the send button is swapped with a create button.
- All new elements are placed and `#styled` appropriately.

- Clicking on the `#abort` button closes the new app bar and restores the previously selected channel (currentChannel). Switching the channel also aborts this creation mode. Note that switching channels does not work at this moment, we will fix that later. Just make the preparations for the app to behave in the indicated way as soon as switching channels works again. **Don't yet restore the messages when aborting, we'll do that in the next sprint.**
- Clicking on the create button in this "channel creation mode" checks if both a `#valid` message (not empty) and valid channel name (not empty, starting with #, no spaces) has been provided.
- If this is the case, a new channel object will be created and the new message will be attached and `#stored` in the currentChannel variable.
  For creating a new channel object you will need a constructor (remember the constructor function building the message object). The other mechanics should be familiar, too.
- Finally, the app bar switches back and the new channel's data is `#show`n (title, location, and the new message). This is basically the first step reversed.

To implement these instructions isn't easy, we know. But after figuring out a way to handle the "creation mode", some tasks (e.g. placing and creating elements) can be partly copied and pasted from older challenges.

`#cleanup`

- Structure and comment your CSS.
- Check code and 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.