WEB FRONTEND DEVELOPMENT II

WDD 330

Ponder: Team Activity: Comments

Overview

This activity will have us look one more time at the Great hikes application. This week we will practice using localStorage, and get another opportunity to think about the architecture of our code by adding a comments system for the hikes.

Instructions

Complete the following assignment as a team. Designate one team member as the "main driver" and collaborate on their copy of the code. Everyone on the team should be actively engaged in writing the code and contributing to the solution. Once the solution is working, make sure that everyone on the team gets a copy of the code. Each week let someone else be the "main driver" of the coding.

Core Requirements

01

If you don't have a working copy of the Great hikes app then you can pull down this version of the working code as a starting point (Zip archive).

Review the code to refresh your memory on the structure we have used. Discuss with your group what methods and properties you might need to implement your comments. A comment should consist of at least the name of the hike it is for, a date, and the actual text of the comment. You will also need a comment `type` to use as a key for when we store these to local storage.

> Type? This is just to allow us to re-use this code. Later you might want to expand on this application and add something like favorite parks or climbs, etc. It would be nice to re-use this module for each of those. Adding a **`type`** flag allows us to add comments for different things and grab all of them or keep them separate.

It might help here to write out a list of steps that need to happen in order to capture a comment. A sample starter list can be found below.

1. When the app loads we see a list of all hikes...below we should also see a list of all comments with type:hike.
2. When a hike is touched it will show us the details for that hike. We should also see any comments for that hike, and an input and button to add a new comment.
3. When the submit comment button is touched...

From this we can start to infer some of the methods and properties we will need:

- A **`getAllComments`** and **`renderCommentList`** method.
- A comment **`type`** to use as a key.
- A **`filterCommentsByName`** method.
- An array to hold all of our comments.
- A comment should look something like this:

```
const newComment = {
  name: hikeName,
  date: new Date(),
  content: comment
};
```

- A method to add an event listener to the submit button.
- An **`addComment`** method
- ...

02 Build out the objects and modules decided on in step 1.

Create a new module for our Comments code. Create a class with a constructor to act as the controller. The constructor should set a local property called **`type`** when

the class is instantiated.

> As you start thinking about implementing your module...think back to the Todo list we made. This is in many ways similar to that. We need a form to gather some user input, we need a button with a listener to store that into an array (and eventually LocalStorage), and we need to display a list of our comments. The biggest difference here is that we don't always want to show the entry form, and we want our module to be reusable.
>
> Still there are enough simularities that you could probably use your ToDo code as a starting point for this. A couple of recommendations for changes:
>
> 1. We could end up with more than one type of comments, and more than one list of comments. If you haven't already, take the methods for adding and removing comments and our comment list and build an additional class around them. Call it something like `CommentModel`.
>
> 2. Depending on how you hide/show your comment entry form you could end up with issues with your listeners. If you start getting funny behavior try using `element.ontouchend = callback`, instead of `element.addEventListener('touchend', callback)` when you attach the listener to your button. This will ensure that you only ever have one listener attached to the button at a time.
>
> 3. One additional change (for the stretch) is that we need to include the name of the hike the comment is for in our saved comment. Consider how you will get that information into your listener so that the callback can use it.

Create a method in the class called **`showCommentsList`**. For now it can just return some static text or log something out to the console.

Insert your comment class into the hike class by adding a **`new`** instance of the comment class in the constructor. Call the **`showCommentsList`** method when the list of hikes is shown to make sure that you have things connected up correctly.

When you are looking at a specific hike it should allow you to enter a new comment, and should show all the comments below that form.

Write the rest of the code to implement your commenting system. For now just store your comments in an array.

03 Store It.

Write the model code to store and retrieve the comments to/from local storage.

Stretch Goals

01 Filtering comments

When you are looking at a hike details screen you should only see the comments for that hike. When you are looking at all hikes you should see all comments of type hike. Change your code to support this.

02 Controlling comment entry

When you are looking at a hike details screen...we should be able to enter a comment, when you are looking at all hikes you should not be able to enter a new comment. Change your code to support this.

Instructors Solution

As a part of this team activity, you are expected to look over a solution from the instructor, to compare your approach to that one. One of the questions on the I-Learn submission will ask you to provide insights from this comparison.

Please DO NOT open the solution until you have worked through this activity as a team for the one hour period. At the end of the hour, if you are still struggling with some of the core requirements, you are welcome to view the instructor's solution and use it to help you complete your own code. Even if you use the instructor's code to help you, you are welcome to report that you finished the core requirements, if you code them up yourself.

After working with your team for the one hour activity, click here for the instructor's solution .

Submission

When you have finished this activity, please fill out the assessment in I-Learn. You are welcome to complete any additional parts of this activity by yourself or with others after your meeting before submitting the assessment.