**Task 4: Adding Items (posts or products**)

Description

For this task, we'll be

1. Creating a class to manage the items (products or posts)

2. Adding a method to the class to keep track of items in our application

Walkthrough

**Step 1: The Setup**

In this step, we'll re-organise our folder structure in preparation for the next few steps.

1. Create a js folder in your project if one does not already exist

2. Copy the existing js file into your js folder, and rename it to *index.js*

3. Update the <script> tag in your html file to use the new location of the js/index.js file.

4. Create a *itemsController.js file* in the js folder

5. Add a <script> tag pointing to the js/itemsController.js file before the <script> tag pointing to the js/index.js file.

**Step 2: The ItemsController Class**

In this step, we'll create a ItemsController class that will be responsible for managing the items(posts or products) in the application.

Always aim to use meaningful and long names to make your code more readable. We encourage you to replace the word Items with a more meaningful one depending on your project domain:

• PostsController

* ProductsController

Useful Resources for this step

• ECMAScript 2015 Classes

1. Create a ItemsController class in js/itemsController.js

2. Within the constructor of the ItemsController class, initialize a *this.items* property on the class, Make the property equal to an empty array.

Test Your Code!

Now is a good chance to test your code, head over to js/itemsController.js and do the following

1. Initialize a new instance of ItemsController

2. console.log() the items property

Expected Result You should see an empty array logged to the browser.

**Step 3: Adding A New Item Programmatically**

In this step, we'll add a method to the ItemsController class that will allow us to add new items to array.

As part of this process, we're going to create a unique id for each item.

For each item to have a unique id, we will need to keep track of what id the latest task was created with, so that we can increment that id for the next item.

For example, pay attention to the two items objects below:

//E-commerce project

const product1 = {

id: 1,

name: ''Tayto'',

description: 'Cheese & Onion Chips',

img: 'https://www.irishtimes.com/polopoly\_fs/1.4078148!/image/image.jpg'

createdAt: '2020-09-20'

};

const product2 = {

id: 2,

name: ''Water'',

description: 'Mineral water',

img: 'http://www.mazalv.com/wp-content/uploads/2016/11/Bottled-Water1-979x1024-1-979x1024.png'

createdAt: '2020-09-20'

};

//Social Media project

const post1 = {

id: 1,

name: 'My first post',

text: 'This is my first post',

img: "https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/Post-greenland-uummannaq.jpg/1200px-Post-greenland-uummannaq.jpg"

author: 'Andres Lowles',

createdAt: '2020-09-20'

};

const post2 = {

id: 2,

name: 'My second post',

text: 'This is my second post',

img: "https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/Post-greenland-uummannaq.jpg/1200px-Post-greenland-uummannaq.jpg"

author: 'Andres Lowles',

createdAt: '2020-09-20'

};

Notice how each item has a unique id? We will be using this id in future steps to keep track of the different items.

Useful Resources for this step

**• Array.prototype.push()**

1. In the ItemsController's constructor, accept a currentId parameter, with a default value of 0.

2. Assign the currentId to a new property on the class, this.currentId.

3. Create a method on the class, addItem. This method should accept all the nessecary information from the form to create an item as parameters.

o Product

o name

o description

o img

o createdAt

o Post

o name

o text

o img

o author

o createdAt

4. Within the addItem method, increment the this.currentId

5. push a new item into the this.items array, with the correct properties of the item, using the values passed in as parameters as well as the new this.currentId Note Make sure to include the id

Test Your Code!

Now is a good chance to test your code, head over to js/itemsController.js and do the following

1. Initialize a new instance of ItemsController

2. Use the addItem method to add a new item

3. console.log() the items property

Expected Result You should see an array containing the added item logged to the browser.

Results

We've now set up the ItemsController class, create an addItem and you are able to see the output on the browser console!

Test out your code by hardcoding items and checking the ItemsController instance's items array for the tasks.

**Task 5: Displaying Items List(posts or products)**

Description

For this task, we'll be creating the feature to display the objects list of the selected project:

Products List

Posts List

Walkthrough

**Step 1: Define the item card layout**

In this step, we'll create the item representation using the card component

Read the documentation and understand how to use the card components

Define the HTML structure of the item card representation.

Add a div element with id list-items to add your list items dynamically with JavaScript.

Add 3 different sample item cards inside your div element.

Expected Result You should see the items list displayed correctly

**Step 2: Adding your items cards programmatically**

In this step, we'll create a function addItemCard(item) that will be responsible for adding new items to the list.

Useful Resources for this step

Document.getElementById()

Define a function called addItemCard(item) inside your in js/items.js

Write the code so your function can create the same HTML structure of your item card representation replacing the item's information.

Test Your Code!

Now is a good chance to test your code, head over to you HTML page and open it on the browser:

Open the console from the Developer Tools.

Excecute the addItemCard(item) function from the console

Expected Result You should see a new item card added to the DOM.

**Step 3: Store and read Items from the LocalStorage**

In this step, we'll connect the ItemsController class and items.js with the local storage to persist your items data.

Modify the items.js adding a new function to save sample items data in the local storage. Use the following JSON structure as reference (make sure you save the data as a String).

const sampleItems = [{'name':'juice',

'img':'https://www.gs1india.org/media/Juice\_pack.jpg',

'description':'Orange and Apple juice fresh and delicious'},

{'name':'Tayto',

'img':'https://www.irishtimes.com/polopoly\_fs/1.4078148!/image/image.jpg',

'description':'Cheese & Onion Chips'}];

localStorage.setItem("items", JSON.stringify(sampleItems));

Modify the ItemsController so it loads the data from the storage implementing a new function items.js

loadItemsFromLocalStorage() {

const storageItems = localStorage.getItem("items")

if (storageItems) {

const items = JSON.parse(storageItems)

//TODO load the items into the local items structure (this.items)

}

}

Implement a new function in the items.js that loads the items from the ItemsController using the function you already implemented addItemCard(item).

Modify the items.js so it calls the loadItemsFromLocalStorage() and then iterate the ItemsController.items list to load the items into your items.html page using the function implemented from addItemCard(item)

Results

We've now implemented a basic version of your application that persist the data on the local storage.

Test out your code by calling the funciton that loads the data from the local storage and verify the items are displayed correctly. You can also use the developer tools and navigate to the Application tab to verify that the items are saved in the local storage.

Example

**Task 6: Creating new items using the form**

Description

For this task, we'll be creating new Items using a form and validating the input data.

Walkthrough

**Step 1: The Setup**

1. Create a new file called item-form.js inside the js folder.

2. Add a <script> tag in your item-form.html file to include the js/item-form.js file.

Step 2: Adding Items With The Form

In this step, we will use the ItemsController class to keep track of items we add with the New Item form.

Useful Resources for this step

• Document.querySelector()

• EventTarget.addEventListener()

• Event Reference

1. Make sure a new ItemsController is initialized near the top of the item-form.js file.

2. In item-form.js, add an event listener to the New Item form, listening to the submit event. If there is already an event listener used for validation, use that one.

3. When the submit event fires, if validation of the form is successful, use the values of each input in the form to call the itemsController's addItem method.

o Note: Make sure to prevent the default action of the form!

4. Update the items list in local storage so the data is also reflected on the items.html.

5. Clear the values from each form input, ready for the next submission.

Results

We've now set up the ItemsController class, used addItem and hooked it up to our New Item form!

Test out your code by adding some items using the New Item form, and checking the ItemsController instance's items array for the items.