AC11002 – WWW Authoring

Mobile Web Applications

The aim of this lab is to introduce you to the techniques that can be used to create a mobile web application. By the end of this lab you will have created a to-do list application that will work as a web page but also as a Portable Web Application.

Completed code for all of the sections below is available in the module GitHub repository.

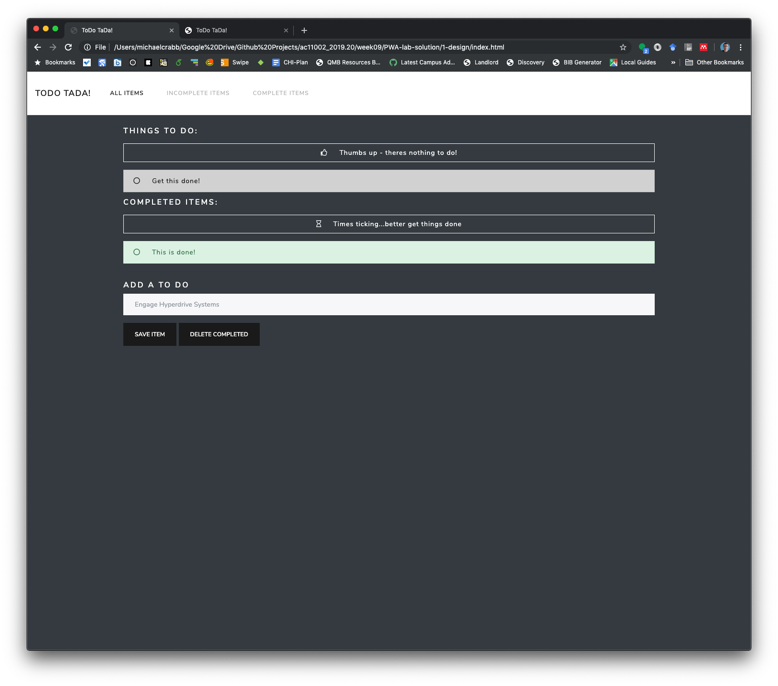
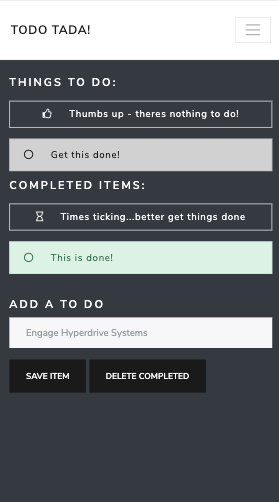
# Designing the Interface

The first thing that we will have to do is create an interface that will work for our application. We’ve been using bootstrap a lot recently, so to make this app look a bit different I’m using a theme from <https://bootswatch.com/> . Feel free to do the same.

We want to create our initial interface so that it looks like a standard to do list:

* We’ll need a nav bar that has links to show all items in our list, only completed items, and only uncompleted items.
* We’ll also want a space to house our to dos
  + I used font awesome to create the to do checkbox circle as well as the thumbs up and hourglass
* We’ll need a space to add in to do items
  + And a button to save them with
* We’ll need a button to delete all completed items.

Go ahead and make this now, your finished interface should look something like the following.



# Saving To Do Items and Displaying in a List

One of the annoying things about localStorage is that it can only hold plain text, so if we want to hold something a bit more complex (say…an array of to-do items) we will need to use the JSON.stringify() function. This takes an array and turns it into a string. We can then parse this any time that we want to read it.

var testObject = { 'one': 1, 'two': 2, 'three': 3 };

localStorage.setItem('testObject', JSON.stringify(testObject));

var retrievedObject = localStorage.getItem('testObject');

retrievedObject = JSON.parse(retrievedObject)

In order for this to work on our page to carry out our functionality, we’re going to have to do the following:

* Any time that an item is added:
  + Get the latest version of the toDo list from local storage
  + Convert that into JSON
  + Add the new item to the JSON array
  + Turn this back into a string
  + Add to local storage
  + Update the to do list
* To Update the to do list:
  + Get the latest version of the toDo list from local Storage
  + Check to see if the array exists (=== null)
    - If so make the array and store it
    - Print out our nothing to do box
  + Check to see if the array is empty ( === “[]”)
    - Print out our nothing to do box
  + Check to see if there is data in the array
    - Convert this into JSON
    - Iterate through array and create a new div for each to show the to do element

You should be able to accomplish all of this through 2 separate functions:

* An event handler attached to the submit button for your form
* A function called updateToDoList()

You will also need to get the toDo list to update when the page is loaded. This should be doable through:

window.onload = updateToDoList();

Another issue that you might run into with data being added to the array is in the use of a form. Forms normally refresh the page when they are submitted. The way you can get round this is to attach event handlers to the save item button (and also the return key if wanted).

# Taking Items off the list

The next thing that you will need to get working is the ability to take items off of your to do list once they have been completed:

* When a to do element is clicked on:
  + Get the latest version of the toDo list from local storage
  + Convert that into JSON
  + Filter out the item from the array that is no longer needed
  + Turn this back into a string
  + Add to local storage
  + Update the to do list

**Hints: you might want to update your HTML for to-do elements to have an onClick event that contains the key index of the item (remember, JSON works with key => value pairs)**

onClick='complete("+index+")'

This code may also help when you are trying to remove items from the array list, it takes an array list, and then returns that list without one item that can be referenced by index

storedList = storedList.filter(function(item) {

return item !== storedList[index];

})

You can manage all of this with one function, you’ll need to pass the index value into this. In the GitHub code this function is called complete(index)

# Implementing a Completed To Do List

The next thing that we want to do is to put completed items into a ‘completed’ to do list. There are a few steps in this and it will be slightly more challenging than you think it will be. You’ll need to create a new updateCompletedToDoList() function that will be similar to the updateToDoList() function and will also have to update your function that runs whenever an item is marked as completed.

You’ll also want to implement a function that will delete all of your completed items. I did this and attached it as an event listener to the delete completed button

function deleteCompleted() {

localStorage.removeItem('completedItems');

updateCompletedToDoList();

}

**Hint: Before carrying out this part of the lab exercise, make a not of all of the things that you will need to get your code to do. It is worth writing this down somewhere and comparing with classmates so that you can be sure you are completing all of the correct steps.**

# Turning this into a PWA

The final main step that we want to do is to turn this into a Progressive Web App. We’ve created the application, now all we need is the service worker and the manifest.

The worker file is very standard and has two parts to it:

* The first is an event listener to install the worker
* The second is an event listener that fetches all pages that are stored by the worker.

Below is what you will need to have in this file:

var CACHE\_NAME = 'simple-PWA-localStorage';

var urlsToCache = [

'./',

'./index.html',

'./manifest.json',

'./bootstrap.css'

];

self.addEventListener('install', function(event) {

// Perform install steps

event.waitUntil(

caches.open(CACHE\_NAME)

.then(function(cache) {

console.log('Opened cache');

return cache.addAll(urlsToCache);

})

);

});

self.addEventListener('fetch', function(event) {

event.respondWith(

caches.match(event.request)

.then(function(response) {

// Cache hit - return response

if (response) {

return response;

}

return fetch(event.request).then(

function(response) {

// Check if we received a valid response

if(!response || response.status !== 200 || response.type !== 'basic') {

return response;

}

var responseToCache = response.clone();

caches.open(CACHE\_NAME)

.then(function(cache) {

cache.put(event.request, responseToCache);

});

return response;

}

);

})

);

});

You’ll then need to refer to this within your application (the index.html file) with the following, placed inside the head:

<script>

if ('serviceWorker' in navigator) {

window.addEventListener('load', function() {

navigator.serviceWorker.register('/sw.js').then(function(registration) {

// Registration was successful

console.log('ServiceWorker registration successful with scope: ', registration.scope);

}, function(err) {

// registration failed :(

console.log('ServiceWorker registration failed: ', err);

});

});

}

</script>

Last is the manifest (which will need to also be refered to in the HTML head)

<link rel="manifest" href="manifest.json">

Information is online on other things that can be put inside of this, but the basics are:

{

"name": "ToDo TaDa",

"short\_name": "ToDo",

"display": "fullscreen",

"theme\_color": "#1a1a1a",

"background\_color": "#1a1a1a"

}

# Lab Extensions

This has been a very full on lab exercise to make this, but hopefully it has given you a bit of an insight into the steps that are required to build a fairly straight forward PWA. This can be extended by:

* Implementing the Incomplete and Complete Items pages
* Finding a way to delete completed items individually
* Sorting items