**Download Your Daily Notes**

**Daily Notes - Activity 1 - File and FileReader**

 In this activity I have learned how to make use of code in order to upload files from the system using an input.

**My own views on File Reader**

 FileReader is an object with the sole purpose of reading data from Blob (and hence File too) objects and it delivers the data using events.  
  
Syntax:  
let reader = new FileReader(); // no arguments  
  
The main method is as follows:  
  
readAsArrayBuffer(blob) – read the data in binary format ArrayBuffer.  
readAsText(blob, [encoding]) – read the data as a text string with the given encoding (utf-8by default).  
readAsDataURL(blob) – read the binary data and encode it as base64 data url.  
abort() – cancel the operation.  
  
As the reading proceeds, there are events:  
  
loadstart – loading started.  
progress – occurs during reading.  
load – no errors, reading complete.  
abort – abort() called.  
error – error has occurred.  
loadend – reading finished with either success or failure.

**Daily Notes - Activity 2 - Here’s an example of reading a file**

 FileReader for blobs  
  
As mentioned in the chapter Blob, FileReadercan read not just files, but any blobs.  
  
We can use it to convert a blob to another format:  
  
readAsArrayBuffer(blob) – to ArrayBuffer,  
readAsText(blob, [encoding]) – to string (an alternative to TextDecoder),  
readAsDataURL(blob) – to base64 data url.  
  
FileReaderSync is available inside Web Workers  
  
For Web Workers, there also exists a synchronous variant of FileReader, called FileReaderSync.  
  
FileReader objects can read from a file or a blob, in one of three formats:  
  
String (readAsText).  
ArrayBuffer (readAsArrayBuffer).  
Data url, base-64 encoded (readAsDataURL).

**Daily Notes - fetch**

 JavaScript can send network requests to the server and load new information whenever is needed.  
  
There are multiple ways to send a network request and get information from the server.  
  
The fetch() method is modern and versatile, so we’ll start with it. It evolved for several years and continues to improve, right now the support is pretty solid among browsers.  
  
Syntax:  
  
let promise = fetch(url, [options])  
  
- url – the URL to access.  
  
- options – optional parameters: method, headers etc.  
  
The browser starts the request right away and returns a promise.  
  
Getting a response is usually a two-stage process.  
  
First, the promise resolves with an object of the built-in Response class as soon as the server responds with headers.  
  
So we can check HTTP status, to see whether it is successful or not, check headers, but don’t have the body yet.  
  
The promise rejects if the fetch was unable to make HTTP-request, e.g. network problems, or there’s no such site. HTTP-errors, even such as 404 or 500, are considered a normal flow.  
  
We can see them in response properties:  
  
· ok – boolean, true if the HTTP status code is 200-299.  
  
status – HTTP status code.

**Daily Notes - Post Requests**

 o make a POST request, or a request with another method, we need to use fetch options:  
  
method – HTTP-method, e.g. POST,  
body – one of:  
a string (e.g. JSON),  
FormData object, to submit the data as form/multipart,  
Blob/BufferSource to send binary data,  
URLSearchParams, to submit the data in x-www-form-urlencoded encoding, rarely used.

**Daily Notes - Activity 3**

 Response provides multiple promise-based methods to access the body in various formats:  
  
response.json() – parse the response as JSON object,  
response.text() – return the response as text,  
response.formData() – return the response as FormData object (form/multipart encoding, explained in the next chapter),  
response.blob() – return the response as Blob(binary data with type),  
response.arrayBuffer() – return the response as ArrayBuffer (pure binary data).  
  
We can choose only one body-parsing method.  
If we got the response with response.text(), then response.json() won’t work, as the body content has already been processed.  
  
There’s a Map-like headers object in response.headers.  
  
there’s a list of forbidden HTTP headers that we can’t set:  
· Accept-Charset, Accept-Encoding  
  
· Access-Control-Request-Headers  
  
· Access-Control-Request-Method  
  
· Connection  
  
· Content-Length  
  
· Cookie, Cookie2  
  
· Date  
  
· DNT  
  
· Expect  
  
· Host  
  
· Keep-Alive  
  
· Origin  
  
· Referer  
  
· TE  
  
· Trailer  
  
· Transfer-Encoding  
  
· Upgrade  
  
· Via  
  
· Proxy-\*  
  
· Sec-\*  
  
These headers ensure proper and safe HTTP, so they are controlled exclusively by the browser.

**Daily Notes - Sending an Image**

 Response properties:  
  
response.status – HTTP code of the response,  
  
response.ok – true is the status is 200-299.  
  
response.headers – Map-like object with HTTP headers.  
  
Methods to get response body:  
  
response.json() – parse the response as JSON object,  
  
response.text() – return the response as text,  
  
response.formData() – return the response as FormData object (form/multipart encoding, see the next chapter),  
  
response.blob() – return the response as Blob(binary data with type),  
  
response.arrayBuffer() – return the response as ArrayBuffer (pure binary data),  
  
Fetch options so far:  
  
method – HTTP-method,  
  
headers – an object with request headers (not any header is allowed),  
  
body – string, FormData, BufferSource, Blob or UrlSearchParams object to send.

**Daily Notes - Activity 4 - Day main activity: fetch users from Github**

 async function getUsers(names) {  
  let jobs = [];  
  for(let name of names) {  
    let job = fetch(`https://api.github.com/users/${name}`).then(  
      successResponse => {  
        if (successResponse.status != 200) {  
          return null;  
        } else {  
          return successResponse.json();  
        }  
      },  
      failResponse => {  
        return null;  
      }  
    );  
    jobs.push(job);  
  }  
  let results = await Promise.all(jobs);  
  return results;  
}

**My Views on the Day**

 1. Learning about File Readers, Fetch, and POST.  
  
2. Activity 1, 2, 3 & 4  
  
3. None.  
  
4. None.

**Daily Notes - Day 1 Reflections**

 1. Learning about File Readers, Fetch, and POST.  
  
2. Activity 1, 2, 3 & 4  
  
3. None.  
  
4. None.