**Download Your Daily Notes**

**My own views on Sending a simple form**

 The above is a form that submits data from its input fields.

**Daily Notes - Sending a simple form**

 The above is a form that submits data from its input fields.

**Daily Notes - FormData Methods**

 We can modify fields in FormData with methods:  
  
formData.append(name, value) – add a form field with the given name and value,  
formData.append(name, blob, fileName) – add a field as if it were <input type="file">, the third argument fileName sets file name (not form field name), as it it were a name of the file in user’s filesystem,  
formData.delete(name) – remove the field with the given name,  
formData.get(name) – get the value of the field with the given name,  
formData.has(name) – if there exists a field with the given name, returns true, otherwise false  
  
There’s also method set, with the same syntax as append. The difference is that .set removes all fields with the given name, and then appends a new field. So it makes sure there’s only field with such name:  
  
formData.set(name, value),  
formData.set(name, blob, fileName).

**Daily Notes - Sending a form with a file**

 In this we have added a name input as well as a section for the user to select an image from the system and then a submit button to submit the form.

**Daily Notes - Fetch: Download progress**

 The fetch method allows to track download progress.  
  
Please note: there’s currently no way for fetch to track upload progress. For that purpose, please use XMLHttpRequest, we’ll cover it later.  
  
To track download progress, we can use response.body property. It’s a “readable stream” – a special object that provides body chunk-by-chunk, as it comes.  
  
Unlike response.text(), response.json() and other methods, response.body gives full control over the reading process, and we can count how much is consumed at any moment.

**Daily Notes - Sending a form with Blob data**

 Please note how the image Blob is added:  
  
formData.append("image", imageBlob, "image.png");  
  
That’s same as if there were <input type="file" name="image"> in the form, and the visitor submitted a file named image.png (3rd argument) from their filesystem.  
  
We can either create new FormData(form) from an HTML form, or create an empty object, and then append fields with methods:  
  
formData.append(name, value)  
formData.append(name, blob, fileName)  
formData.set(name, value)  
formData.set(name, blob, fileName)  
Two peculiarities here:  
  
The set method removes fields with the same name, append doesn’t.  
To send a file, 3-argument syntax is needed, the last argument is a file name, that normally is taken from user filesystem for <input type="file">.  
Other methods are:  
  
formData.delete(name)  
formData.get(name)  
formData.has(name)

**Daily Notes - Fetch: Abort**

 There’s a special built-in object for such purposes: AbortController.  
  
Step 1: create a controller:  
let controller = new AbortController();  
  
A controller is an extremely simple object. It has a single method abort(), and a single property signal. When abort() is called, the abortevent triggers on controller.signal  
  
Step 2: pass the signal property to fetch option:  
let controller = new AbortController();  
fetch(url, {  
signal: controller.signal  
  
});  
  
Now fetch listens to the signal.  
  
Step 3: to abort, call controller.abort():  
controller.abort();  
  
We’re done: fetch gets the event from signal and aborts the request.  
  
When a fetch is aborted, its promise rejects with an error named AbortError, so we should handle it.  
  
AbortController is scalable, it allows to cancel multiple fetches at once.

**My Views on the Day**

 1. Learning about the fetch, in terms of capabilities like download and abort, etc.  
  
2. None.  
  
3. None.  
  
4. None.

**Daily Notes - Day 2 Reflections**

 1. Learning about the fetch, in terms of capabilities like download and abort, etc.  
  
2. None.  
  
3. None.  
  
4. None.