**File and FileReader**

 In JavaScript, a File object inherits from the Blob object and is extended with additional properties and methods related to file system information and user-selected files.  
  
Two ways to obtain it.  
1. There is a constructor similar to Blob:  
  
new File(fileParts, fileName, [options])  
  
fileParts - is an array of Blob/BufferSource/String values.  
fileName - file name string.  
options - optional object:  
lastModified - the timestamp (integer date) of last modification.  
  
2. We get a file from <input type="file"> or drag’n’drop or other browser interfaces. In that case, the file gets this information from OS.  
  
As File inherits from Blob, File objects have the same properties, plus:  
  
name – the file name,  
  
lastModified – the timestamp of last modification  
  
That’s how we can get a File object from <input type="file">:

**File Reader**

 The FileReader is a JavaScript API that allows you to read the contents of files asynchronously from the user's local file system. It is commonly used in web applications for tasks such as reading and processing user-selected files, handling file uploads, and more.  
  
The constructor is as follows:  
  
let reader = new FileReader();

Here’s an example of reading a file

 FileReader for blobs  
  
FileReader can 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.  
  
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).

**Fetch**

 The fetch() method is a modern JavaScript API for making network requests, typically used to retrieve data from a remote server or send data to a server. It provides a more flexible and powerful way to work with HTTP requests compared to older techniques.  
  
The basic syntax is:  
  
let promise = fetch(url, [options])

**Post Requests**

 We need to use fetch options to make a POST request, or a request with another method.

 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),

**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.