**File and file reader.**

a file object is specialized type of blob, and we have two ways to obtain it that is

1. Constructor: You can create a File object using the File constructor. This constructor takes three parameters.

This constructor takes three parameters.

fileParts: An array of data sources that can be Blob, BufferSource, or String values.

filename: file name string.

Options: An object that can contain additional information about the file including last modified.

2. Using input type or drag and drop.

Using <input type="file"> or Drag-and-Drop: In most cases, you'll obtain a File object from user interactions, such as selecting a file through an <input type="file"> element or dragging and dropping files into a web application

A FileReader in JavaScript is an object specifically designed for reading data from Blob objects, including File objects, which are commonly used to represent files uploaded by users.

Now lets breakdown on how the file reader works:

1. We have constructor: You create a FileReader object using its constructor without any arguments.

2. Then we have reading method: provide several methods for reading the content of a blob or file.

3. The we have events: as the reading process takes time, file reader emits various events to inform you about its progress and status. The common events include:

• loadstart: Indicates that the loading process has started.

• progress: Occurs during reading, providing updates on the progress.

• load: Fired when the reading is successfully completed.

• abort: Triggered when the abort () method is called to cancel the operation.

• error: Indicates an error has occurred during reading.

• loadend: Fired when the reading process is finished, whether it was successful or not.

4. Lastly, we have Accessing results: after reading is complete, you can access results or error as follows.

Reader. results

Reader. error

Commonly used events are load (when reading is successful) and error (when there's an issue). Developers use FileReader to read and process data from files uploaded by users, allowing them to manipulate or display the data as needed within a web application.

**FormData methods**

1. formData.append(name, value): This method adds a new field with the given name and value to the FormData object. If a field with the same name already exists, it appends a new field with the same name.
2. formData.append(name, blob, fileName): This method adds a field as if it were an <input type="file"> input. The name parameter specifies the field name, the blob parameter is the binary data (usually a file), and the fileName parameter sets the name of the file as it would appear in the user's filesystem.
3. formData.delete(name): This method removes the field with the given name from the FormData object.
4. formData.get(name): This method retrieves the value of the field with the given name from the FormData object.
5. formData.has(name): This method checks if there exists a field with the given name in the FormData object. It returns true if a field with the specified name exists, and false otherwise.
6. formData.set(name, value): This method removes all fields with the given name and then appends a new field with the specified name and value.
7. formData.set(name, blob, fileName): Similar to the previous method, this one removes all fields with the given name and then appends a new field as if it were an <input type="file"> input

**fetch**

The fetch method in JavaScript allows you to make HTTP requests, including downloading data from a server. While it doesn't provide a built-in way to track upload progress when sending data to a server, you can indeed track download progress using the response.body property and the ReadableStream.

To initiate a fetch request and receive a response from a server, you typically use the fetch function.

Reading Response as a ReadableStream: When the server responds to your fetch request, the response object provides a body property. This body property is a ReadableStream, which is a special type of object that represents a stream of data that can be read chunk by chunk.

Reading Response as a ReadableStream: When the server responds to your fetch request, the response object provides a body property.

Reading Data Chunks: Once you have a ReadableStream and a reader, you can read data from the stream in chunks.