Upload and download data

Be aware that we are showing som code-examples (using *node express*) on the server in this Powerpoint. This is only for background reading. You are not supposed to code anything on the server in this course.

First, a little bit about binary vs. text data

- Digital information are by default binary – it consists of zeroes and ones
- To better process, transmit and use the information, we often code it in different formats.
- For example, we often use a text format (all the data can be recognized as characters) to make it easier for humans to interpret and organize the data, e.g., JSON or XML.
- It is also possible to convert binary data to text, by coding the binary data into for example *base64* – which consists of characters containing 6 bit each. By doing this we can include binary data, like an image, into JSON or XML-text. I.e., (from Wikipedia): Base64 is designed to carry data stored in binary formats across channels that only reliably support text content
- We can send both text and binary data using
- The HTTP protocol supports both binary and text data – using different methods for sending and receiving the data.

Mer om tekst vs. binære data (Norwegian only)

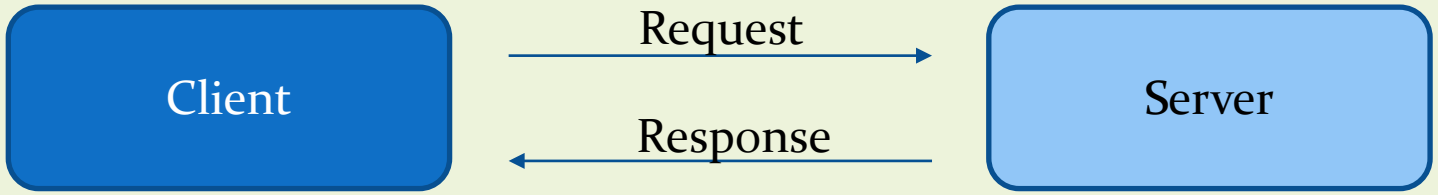
- Alle data vi overfører er binære i utgangspunktet (består av 0'er og 1'ere), men dette trenger bare datamaskinen å bry seg om. Vi kan i stedet bestemme hva dataene representerer.
- Ønsker vi å overføre tekst, vil dataene naturlig nok representere karakterer. Ønsker vi for eks. å overføre et bilde, vil dataene representere fargeverdier for pikslene i bildet.
- For et bilde vil det derfor være naturlig å overføre dataene som tall-verdier (binære data), for eks. RGB-verdiene for hver piksel. En enkel gul piksel kan da for eks. lagres som tallene 255, 255, 0 – dvs. 3 byte. (Dette kan også skrives på heksadesimal form: ffff00).

Konvertere fra binære data til tekst-data (Norwegian only)

- Hvis vi ønsker å overføre binære data som tekst, kan vi konvertere tall-verdiene til karakterer.
- Vi kan for eks. bruke kommaseparerte tallverdier. En piksel vil da kunne representeres om karakterene 255,255,0 – dvs. 9 karakterer. Hver karakter vil som regel lagres i én byte – dvs. at pikselen nå krever 9 bytes med lagringsplass.
- Hvis vi i stedet bruker heksadesimale verdier, ffff00, trenger vi 6 bytes med lagringsplass. Vi ser at dette reduserer «sløsingen», men fortsatt så krever tekst-versjonen av dataene dobbelt så mye plass (100 % *overhead*)
- Det finnes optimaliserte (og standardiserte) måter å konvertere (kode) binære verdier til karakterer – slik at vi får lite overhead. En av de mest vanlige kalles *Base64*. Her blir hver karakter representert med 6 bit (gir et «alfabet» på 64 forskjellige karakterer). En fargeverdi på 3 byte (24 bit) kan da representeres med 4 karakterer i Base64
- Vi ser at Base64 gir 0 % overhead når vi koder 24bits fargeverdier. Vær klar over at vi kan få overhead hvis de binære dataene ikke er organisert i et antall bit som går opp i 6.

HTTP requests and responses

- The communication between the client and the server happens through requests and responses
- The client sends a request to the server, the server will then process the request and send a response back to the client:



- Both the request and the response may contain data in different parts of the request and response

Sending data in the request

- The request consists of:
  - A *request line* (address field) with an address to the requested resource
  - A *header* with information about the request (metadata)
  - A *body* (optional) with additional information/data

- It is possible to send information in all of the three parts of the request:
  - As URL Query strings (URL variables) in the request line (text only)
  - As parts of a route in the request line (text only)
  - In header-lines in the header (text only)
  - In the body (text and binary data)

Sending data as URL queries (URL variables)

- **On the client**: Add the query to the url, e.g.:

```
http://localhost:3000/person/?name=Anne&age=34
```

- The question mark marks the start of the query variables. You can add several variables using the &

(Tech - background reading) Example on the server. In *node express*, you can retrieve the data like this:

```
// endpoint GET --------------------------------
app.get('/', function (req, res) {

    let name = req.query.name;
    let age = req.query.age;

    // more code…
});
```

Note. You can also retrieve URL queries in other endpoints, e.g. in POST

Sending data as part of a route

- **A value can be sent as part of a route:**

`http://localhost:3000/car/PD12345/red`

(Tech - background reading) Example on the server. In *node express*, you can retrieve the data like this:

```
// endpoint GET --------------------------------
app.get('/car/:regnumber/:color', function (req, res) {

    let regnumber = req.params["regnumber"];
    let color = req.params["color"];

    // more code…
});
```

Note. You can also retrieve URL queries in other endpoints, e.g. in POST

Sending data in a header

- **On the client**: Add a header in the the request, e.g.:

```
let cfg = {
    method: "GET",
    headers: {
        "name": "John",
        "occupation": "nurse"
    }
}

url = "http://localhost:3000";
let resp = await fetch(url, cfg);
// more code...
```

Note! It is not common to send general data in *custom* headers like in this example. We usually use *standardized* headers, e.g.:

- "**content-type**" to describe the type of content, or
- "**authorization**" to send login-information (username and password).

(Tech - background reading) Example on the server. In *node express*, you can retrieve the data like this:

```
// endpoint GET --------------------------------
app.get('/', function (req, res) {

    let name = req.headers['name'];
    let occup = req.headers['occupation'];
    // more code...
});
```

Note. You can also retrieve URL queries in other endpoints, e.g. in POST

Sending json (text) data in the body

- You can send both binary and text data in the body
- Usually, you send text data as JSON.
- We can also send binary data and files (e.g., an image - see later slides).

- **On the client**: In this example we transfer some JSON data

```
let updata = {
    name: "Anne",
    occupation: "Carpenter"
}

let cfg = {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify(updata)
}


url = "http://localhost:3000/person";
let resp = await fetch(url, cfg);

// more code…
```

(Tech - background reading) Example on the server. In *node express*, you can retrieve the data like this:

```
// body parser middleware --------------------
app.use(express.json({limit:'1mb'}));

// endpoint POST -----------------------------
app.post('/', function (req, res) {

    let name = req.body.name;
    let occupation = req.body.occupation;
    // more code...
});
```

Note. You can also retrieve URL queries in other endpoints, e.g. in POST

Sending a file as base64 encoded text

- By converting the data in a file to base64, you can send it as json in the body (see previous slide about sending data in the body).
- To read and convert the data in a file into base64 encoded text (on the **client**), you can use a *FileReader-object*, e.g.:

```
let base64Data;

let freader = new FileReader();

//event handler that is called when reading the file is finished
freader.onload = function() {
    base64Data = freader.result;
    //do something with the data, e.g., send it in the body as json data
}

//start reading (converting) the file as base64 using the readAsDataURL-method
freader.readAsDataURL(theFile);
```

Note. The filereader-object has several methods to read and convert files to different formats. Use *readAsArrayBuffer(theFile)* to read the files as binary data.

Sending raw (binary) data in the body

```
let theRawData = someRawData; //e.g., from a file using a FileReader-object (se previous slide)

let cfg = {
    method: "POST",
    headers: {
        "Content-Type": "application/octet-stream"
    },
    body: theRawData
}


url = "http://localhost:3000";
let resp = await fetch(url, cfg);

// more code…
```

(Tech - background reading) Example on the server. In *node express*, you can retrieve the data like this:

```
// body parser middleware --------------------
app.use(express.raw({limit:'1mb'}));

// endpoint POST -----------------------------
app.post('/', function (req, res) {

    let theRawData = req.body;

    // more code...
});
```

Note. You can also retrieve URL queries in other endpoints, e.g. in POST

Sending formdata

- Traditionally, *formdata* was sent from a <form> element on the webpage to a server. The server would then return the result as an HTML-formatted webpage which was shown in the browser.
- By default, the data in the form are sent as query-strings. We can also send binary data (files) by setting the *enctype-attribute* to *Multipart/form-data* encoding.
- When we are using fetch/AJAX, we can't directly send formdata from the <form> element. Instead, we can either):
  - Create a <form> element with name attributes as property-names and then extract the values using a FormData-object.
  - Create a FormData-object and populate it with data in JavaScript.

(see next slides for examples)

Sending file(s) using a <form> element

- In this example we both send a file and some info-text.

HTML:

```
<form enctype="multipart/form-data" id="myForm">
    <input name="image" type="file" />
    <input name="info" type="text" />
    <input type="submit" value="Send data">
</form>
```

Javascript:

```
let myForm = document.getElementById('myForm');
myForm.addEventListener('submit', async function (evt) {

    evt.preventDefault(); //don't use default submit

    let updata = new FormData(myForm); //extract the data

    let cfg = {
        method: "POST",
        body: updata
    }

    url = "http://localhost:3000";
    let resp = await fetch(url, cfg);
    // more code...
});
```

Sending file(s) by using a FormData-object

- This is the same as the last example, but we don't use any <form>-element. Instead, we create the formdata in JavaScript:

HTML:

```
<input id="inpImg" type="file" />
<input name="inpInfo" type="text" />
<button id="btnSend">Send data</button>
```

Javascript:

```
let inpImg = document.getElementById("inpImg");
let inpName = document.getElementById("inpName");
let btnSend = document.getElementById("btnSend");
let theFile;

inpImg.addEventListener('change', function (evt) {
    theFile = inpImg.files[0];
});
```

```
btnSend.addEventListener('click', async function (evt) {

    let updata = new FormData();
    updata.append("image", theFile);
    updata.append("info", inpInfo.value);

    let cfg = {
        method: "POST",
        body: updata
    }

    url = "http://localhost:3000";
    let resp = await fetch(url, cfg);
    // more code...
});
```

(Tech - background reading) Example on the server

Receiving the file and additional formdata

- On the server we can retrieve the form data using a plugin (package) called *multer*.
- Install multer by writing this in a console window: **`npm install multer`**

Must match with the property/name on the client:
`updata.append("image", file);`

```
const multer = require('multer’);
//let mem = multer.memoryStorage();
//let mult = multer({storage: mem}); //save to memory (buffer)
let mult = multer({dest: "uploads/"}); //save file to a folder on the server

// endpoint POST ---------------------------------
app.post('/', mult.single('image'), function (req, res) {
    let info = req.body["info"];
    let file = req.file; //the file
    // let imgData = req.file.buffer; //contains the file-data if saved to memory
    // more code...
}
```

Converting formdata to JSON

- If you are going to send a lot of different values to a server, it can be smart to use a formdata-element in HTML. Then you can use the *name*-attribute instead of retrieving all the values using *id* and *getElementById*. But if the server is configured to only receive JSON, you must first convert the formdata into a JavaScript object, then into JSON. Here is an example of a function that converts a FormData-object to a JavaScript-Object (JSON object literal) :

```
//---------------------------------------------
function formDataToObj(myFormData) {
    var obj = {};
    myFormData.forEach(function (value, key) {
        obj[key] = value;
    });
    return obj;
}
```

- You can the use the *JSON.stringify* method to convert the JavaScript-object to JSON-text.