formation.tech

Network APIs

Network APIs - XMLHttpRequest

- Microsoft introduced in 1999 with IE5 the possibility the make AJAX request through the XMLHTTP ActiveX.
- It allowed its Outlook Web Access web app to get new emails without reloading the complete page
- Mozilla then introduced an API called XMLHttpRequest which is 100% compatible with the Microsoft ActiveX
- Request can be send asynchronously or synchronously (non sense as the browser will freeze during the request)
- The XMLHttpRequest class got new functionalities in 2011 (cross-origin requests, progress events...)

Network APIs - XMLHttpRequest

- XMLHttpRequest is a constructor function (class)
- Then open method is used to set the HTTP method/verb and URL
- We can listen to event like : load, progress, error
- The send method is used to send the request

```
// contact.json
{ "firstName": "Bill", "lastName": "Gates" }
```

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'contact.json');
xhr.addEventListener('load', (event) => {
  const contact = JSON.parse(xhr.response);
  console.log(contact.firstName + ' ' + contact.lastName);
  // Bill Gates
});
xhr.send();
```

Network APIs - Fetch

- Fetch is a new modern API to make HTTP Request in the browser
- It is a promised based API
- Fetch doesn't exists in some old browsers (Internet Explorer...) : https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- There is a polyfill created by Github to bring the compatibility in those browsers https://github.com/github/fetch
- Fetch doesn't exist in Node.js but some implementation exists if needed (Server Side Rendering / SSR) : https://github.com/bitinn/node-fetch
- If you need an implementation that works in the browser and server you can use : https://github.com/matthew-andrews/isomorphic-fetch
- Since fetch is promise based, some functionality doesn't work yet (progress events)

Network APIs - Fetch

- The fetch method will return a promise that will resolve a Response object : https://developer.mozilla.org/en-US/docs/Web/API/Response
- The response is a stream that will be read asynchronously thought methods like json, blob or text

```
fetch('contact.json')
  .then((res) => res.json())
  .then((contact) => {
    console.log(contact.firstName + ' ' + contact.lastName);
  });
```

Network APIs - Node.js HTTP Server

- Node.js can create HTTP servers very easily, in particular with libraries like Express
- Install :
  `npm install express`

```
const express = require('express');

const app = express();

app.get('/contact/1', (req, res) => {
  res.json({ firstName: 'Bill', lastName: 'Gates' });
});

app.listen(3000, () => {
  console.log('Server started');
});
```

Network APIs - JSON Web Tokens

- JSON Web Token or JWT is a token system that doesn't need to be stored
- It contains a signature that will be verified on subsequent requests
- To manipulate JWT we can use :
  - in Node.js (to generate and verify) : https://github.com/auth0/node-jsonwebtoken
  - in the browser (to decode the payload) : https://github.com/auth0/jwt-decode
- You can try it on https://jwt.io

Network APIs - JSON Web Tokens

- Generate a JWT token

```
app.post('/user/signin', express.json(), (req, res) => {
  if (req.body.username !== user.username || req.body.username !==
user.username) {
    return res.status(401).json({ error: 'Wrong credentials' });
  }
  const token = jwt.sign({ id: user.id, username: user.username }, secret);
  res.json({ token });
});
```

- Verify a JWT token

```
function jwtMiddleware(req, res, next) {
  const token = req.headers.authorization;
  try {
    jwt.verify(token, secret);
  } catch(err) {
    return res.status(401).json({ error: 'Unauthorized' });
  }
  return next();
}
```

Network APIs - WebSocket

- XHR and Fetch are based on HTTP
- HTTP until version 1.1 is a client pool only protocol, that means that request are coming from the client
- HTTP 2.0 allows server push
- WebSockets is a bidirectional communication system which is permanently open between the client and server

Network APIs - WebSocket

- Example with the echo from websocket.org (returns the same message sent)

```
const ws = new WebSocket('wss://echo.websocket.org');
ws.addEventListener('open', () => {
  console.log('Connected');
  ws.send('Hello');
  ws.send('Bye');
  setTimeout(() => {
    ws.close();
  }, 1000);
});
ws.addEventListener('close', () => {
  console.log('Disconnected');
});
ws.addEventListener('message', (event) =>  {
  console.log('Message : ' + event.data);
});
// Connected
// Message : Hello
// Message : Bye
// Disconnected
```

Network APIs - WebSocket

- In Node.js we can use 2 principal implementations :
  - ws
  - socket.io
- Example : an echo server

```
const { WebSocketServer } = require('ws');

const server = new WebSocketServer({ port: 8080 });

server.on('connection', (ws) => {
  ws.on('message', (message) => {
    ws.send(message);
  });
});
```

Network APIs - EventSource

- If the network communication is only from the server to the client we could use a new API : EventSource

```
const es = new EventSource('http://localhost:3000/event-stream');
es.addEventListener('open', () => {
  console.log('Connected');
});
es.addEventListener('message', (event) => {
  console.log(event.data);
});
es.addEventListener('close', () => {
  console.log('Disconnected');
});
```

Network APIs - EventSource

- In server side, we will create a response that is kept alive :

```
const express = require('express');

const app = express();

app.get('/event-stream', (req, res) => {
  res.writeHead(200, {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive',
    'Access-Control-Allow-Origin': '*',
  });
  res.write('\n');

  let id = 0;
  const timeout = setInterval(() => {
    res.write(`id: ${++id}\n`);
    res.write(`data: ${Date.now()}\n\n`);
  }, 1000);

  req.on('close', () => clearInterval(timeout));
});

app.listen(3000);
```