Chapter 36: AJAX

AJAX stands for "Asynchronous JavaScript and XML". Although the name includes XML, JSON is more often used due to its simpler formatting and lower redundancy. AJAX allows the user to communicate with external resources without reloading the webpage.

Section 36.1: Sending and Receiving JSON Data via POST

Version ≥ 6

Fetch request promises initially return Response objects. These will provide response header information, but they don't directly include the response body, which may not have even loaded yet. Methods on the Response object such as `.json()` can be used to wait for the response body to load, then parse it.

```
const requestData = {
  method : 'getUsers'
};

const usersPromise = fetch('/api', {
  method : 'POST',
  body : JSON.stringify(requestData)
}).then(response => {
  if (!response.ok) {
    throw new Error("Got non-2XX response from API server.");
  }
  return response.json();
}).then(responseData => {
  return responseData.users;
});

usersPromise.then(users => {
  console.log("Known users: ", users);
}, error => {
  console.error("Failed to fetch users due to error: ", error);
});
```

Section 36.2: Add an AJAX preloader

Here's a way to show a GIF preloader while an AJAX call is executing. We need to prepare our add and remove preloader functions:

```
function addPreloader() {
  // if the preloader doesn't already exist, add one to the page
  if(!document.querySelector('#preloader')) {
    var preloaderHTML = '<img id="preloader" src="https://goo.gl/cNhyvX" />';
    document.querySelector('body').innerHTML += preloaderHTML;
  }
}

function removePreloader() {
  // select the preloader element
  var preloader = document.querySelector('#preloader');
  // if it exists, remove it from the page
  if(preloader) {
    preloader.remove();
  }
}
```

Now we're going to look at where to use these functions.

```
var request = new XMLHttpRequest();
```

Inside the `onreadystatechange` function you should have an if statement with condition: `request.readyState == 4 && request.status == 200`.

If **true**: the request is finished and response is ready that's where we'll use `removePreloader()`.

Else if **false**: the request is still in progress, in this case we'll run the function `addPreloader()`

```
xmlhttp.onreadystatechange = function() {

  if(request.readyState == 4 && request.status == 200) {
    // the request has come to an end, remove the preloader
    removePreloader();
  } else {
    // the request isn't finished, add the preloader
    addPreloader()
  }

};

xmlhttp.open('GET', your_file.php, true);
xmlhttp.send();
```

Section 36.3: Displaying the top JavaScript questions of the month from Stack Overflow's API

We can make an AJAX request to Stack Exchange's API to retrieve a list of the top JavaScript questions for the month, then present them as a list of links. If the request fails or the returns an API error, our promise error handling displays the error instead.

Version ≥ 6

View live results on HyperWeb.

```
const url =
    'http://api.stackexchange.com/2.2/questions?site=stackoverflow' +
    '&tagged=javascript&sort=month&filter=unsafe&key=gik4BOCMC7J9doavgYteRw((';

fetch(url).then(response => response.json()).then(data => {
  if (data.error_message) {
    throw new Error(data.error_message);
  }

  const list = document.createElement('ol');
  document.body.appendChild(list);

  for (const {title, link} of data.items) {
    const entry = document.createElement('li');
    const hyperlink = document.createElement('a');
    entry.appendChild(hyperlink);
    list.appendChild(entry);

    hyperlink.textContent = title;
    hyperlink.href = link;
  }
}).then(null, error => {
  const message = document.createElement('pre');
```

```
  document.body.appendChild(message);
  message.style.color = 'red';

  message.textContent = String(error);
});
```

Section 36.4: Using GET with parameters

This function runs an AJAX call using GET allowing us to send **parameters** (object) to a **file** (string) and launch a **callback** (function) when the request has been ended.

```
function ajax(file, params, callback) {

  var url = file + '?';

  // loop through object and assemble the url
  var notFirst = false;
  for (var key in params) {
    if (params.hasOwnProperty(key)) {
      url += (notFirst ? '&' : '') + key + "=" + params[key];
    }
    notFirst = true;
  }

  // create a AJAX call with url as parameter
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      callback(xmlhttp.responseText);
    }
  };
  xmlhttp.open('GET', url, true);
  xmlhttp.send();
}
```

Here's how we use it:

```
ajax('cars.php', {type:"Volvo", model:"300", color:"purple"}, function(response) {
  // add here the code to be executed when data comes back to this page
  // for example console.log(response) will show the AJAX response in console
});
```

And the following shows how to retrieve the url parameters in `cars.php`:

```
if(isset($_REQUEST['type'], $_REQUEST['model'], $_REQUEST['color'])) {
  // they are set, we can use them !
  $response = 'The color of your car is ' . $_REQUEST['color'] . '. ';
  $response .= 'It is a ' . $_REQUEST['type'] . ' model ' . $_REQUEST['model'] . '!';
  echo $response;
}
```

If you had `console.log(response)` in callback function the result in console would have been:

> The color of your car is purple. It is a Volvo model 300!

Section 36.5: Check if a file exists via a HEAD request

This function executes an AJAX request using the HEAD method allowing us to **check whether a file exists in the directory** given as an argument. It also enables us to **launch a callback for each case** (success, failure).

```
function fileExists(dir, successCallback, errorCallback) {
    var xhttp = new XMLHttpRequest;

    /* Check the status code of the request */
    xhttp.onreadystatechange = function() {
        return (xhttp.status !== 404) ? successCallback : errorCallback;
    };

    /* Open and send the request */
    xhttp.open('head', dir, false);
    xhttp.send();
};
```

Section 36.6: Using GET and no parameters

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
    if (xhttp.readyState === XMLHttpRequest.DONE && xhttp.status === 200) {
        //parse the response in xhttp.responseText;
    }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Version ≥ 6

The `fetch` API is a newer promise-based way to make asynchronous HTTP requests.

```
fetch('/').then(response => response.text()).then(text => {
  console.log("The home page is " + text.length + " characters long.");
});
```

Section 36.7: Listening to AJAX events at a global level

```
// Store a reference to the native method
let open = XMLHttpRequest.prototype.open;

// Overwrite the native method
XMLHttpRequest.prototype.open = function() {
    // Assign an event listener
    this.addEventListener("load", event => console.log(XHR), false);
    // Call the stored reference to the native method
    open.apply(this, arguments);
};
```