Starting with AJAX

TABLE OF CONTENTS

PREFACE

Welcome to the Getting Started with Ajax cheatsheet! This cheatsheet provides a quick reference guide for working with Ajax, covering everything from making AJAX requests with XHR to using popular Ajax toolkits like jQuery and React. Whether you're a beginner looking to get started with Ajax or an experienced developer looking for a quick reference guide, this cheatsheet has something for you. So, let's dive in and start building dynamic web applications with Ajax!

INTRODUCTION

AJAX (Asynchronous JavaScript and XML) is a technique for building dynamic web applications that can update content on a web page without requiring a full page reload. It uses a combination of JavaScript and XML (or other data formats like JSON) to send and receive data from a web server asynchronously, without disrupting the user's experience on the page.

AJAX has become a popular technique for building modern web applications that require real-time updates, such as chat apps, social media feeds, and e-commerce sites. It allows developers to build more responsive and interactive user interfaces, by enabling them to update the page content without requiring the user to manually refresh the page.

This cheatsheet provides an overview of some of the key concepts and techniques involved in building AJAX applications, including how to make AJAX requests, how to handle responses from the server, and how to modify the DOM dynamically. It also includes information on some of the tools and frameworks commonly used in AJAX development, to help developers work more efficiently and effectively.

XHR

XHR stands for "XMLHttpRequest". It is a built-in web API in JavaScript that allows you to make HTTP requests to a server without having to reload the page. With XHR, you can retrieve data from a server, send data to a server, and perform other types of HTTP requests such as POST and PUT.

METHODS

Method	Description
`open()`	Initializes a request. Takes in the HTTP method (e.g. GET, POST), URL to send the request to, and whether the request should be asynchronous (true or false).
`setRequestHeader()`	Sets the value of an HTTP request header. This method should be called after `open()` and before `send()`.
`send()`	Sends the request to the server. This method should be called after `open()` and optionally after `setRequestHeader()`. If sending data, it should be passed as an argument to the `send()` method.
`abort()`	Aborts the request if it has already been sent.
`getResponseHeader()`	Returns the value of the specified response header.
`getAllResponseHeaders()`	Returns all the response headers as a string.
`onreadystatechange`	An event handler that is called whenever the `readyState` property changes.
`readyState`	Holds the status of the XMLHttpRequest object. Its value is an integer representing the state of the request.
`status`	Holds the HTTP status code of the response.
`statusText`	Holds the HTTP status message of the response.

Method	Description
responseType	Determines the type of response. Possible values include "text", "json", "blob", "document", and "arraybuffer".

XHR READYSTATE VALUES

Readystate Value	Description
0	The request has not been initialized.
1	The request has been set up but not sent.
2	The request has been sent but the server has not yet responded.
3	The server is processing the request and has started sending a response.
4	The server has finished sending the response and the request has been completed.

CREATE AN XHR OBJECT

To create an XMLHttpRequest object for an AJAX request, you can use the XMLHttpRequest constructor function.

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

OPEN A CONNECTION

To open a connection for an AJAX request using the XMLHttpRequest object, you can use the open() method.

```
xhr.open("GET", "your_api_endpoint",
true);
```

The third parameter specifies whether the request should be asynchronous or not.

SET HEADERS

To set headers for an AJAX request using the XMLHttpRequest object, you can use the setRequestHeader() method.

```
xhr.setRequestHeader("Content-type",
"application/json");
xhr.setRequestHeader("Authorization"
, "Bearer your_access_token");
```

SEND THE REQUEST

To send an AJAX request using the XMLHttpRequest object, you can use the send() method.

```
xhr.send();
```

LISTEN FOR THE RESPONSE

To listen for the response of an AJAX request using the XMLHttpRequest object, you can set the onreadystatechange event handler and check the readyState and status properties of the XMLHttpRequest object.

```
xhr.onload = function() {
   // Handle successful response
   var data =
JSON.parse(xhr.responseText);
};

xhr.onerror = function() {
   // Handle error
};
```

SEND DATA WITH THE REQUEST

To send data with an AJAX request using the XMLHttpRequest object, you can set the request body using the send() method.

```
xhr.open("POST",
"your_api_endpoint", true);
xhr.setRequestHeader("Content-type",
"application/json");
```

```
xhr.send(JSON.stringify({ key1:
value1, key2: value2 }));
```

USE WITH FORMDATA

FormData is a JavaScript object that provides an easy way to construct a set of key-value pairs to send with an AJAX request. It can be used to construct a form data object from an HTML form or to create a new form data object manually.

```
var formData = new FormData();
formData.append("file",
fileInputElement.files[0]);
formData.append("key1", value1);
formData.append("key2", value2);

xhr.open("POST",
"your_api_endpoint", true);
xhr.send(formData);
```

HANDLING AN HTML RESPONSE

To handle an HTML response from an XHR request, you can use the XMLHttpRequest object's responseText property to retrieve the HTML content of the response.

```
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function()
{
  if (this.readyState == 4 &&
this.status == 200) {
    const htmlResponse =
this.responseText;
    // Do something with the HTML
content
  }
};
xhr.open('GET',
'https://example.com', true);
xhr.send();
```

HANDLING A JSON RESPONSE

To handle a JSON response from an XHR request, you can use the XMLHttpRequest object's responseText property to retrieve the JSON content of the response and then parse it into a JavaScript object using the JSON.parse() method.

```
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function()
{
  if (this.readyState == 4 &&
this.status == 200) {
    const jsonResponse =
JSON.parse(this.responseText);
    // Do something with the
JavaScript object
  }
};
xhr.open('GET',
'https://example.com/data.json',
true);
xhr.send();
```

HANDLING AN XML RESPONSE

To handle an XML response from an XHR request, you can use the XMLHttpRequest object's responseXML property to retrieve the XML content of the response. Here's an example code snippet:

```
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function()
{
  if (this.readyState == 4 &&
this.status == 200) {
    const xmlResponse =
this.responseXML;
    // Do something with the XML
content</code>
    const elements =
xmlResponse.getElementsByTagName('el
ementName');

console.log(elements[0].textContent)
;
  }
};
xhr.open('GET',
'https://example.com/data.xml',
true);
```

```
xhr.send();
```

Note that handling an XML response can be more complex than handling a JSON or HTML response, as XML has a more complex structure and may require more specialized parsing techniques. Additionally, modern web APIs often prefer to use JSON over XML due to its simplicity and ease of use.

TIPS FOR USING XHR

- **Understand the XHR lifecycle**: The XHR object goes through a series of states as it sends a request and receives a response from the server. Understanding the lifecycle of an XHR request can help you write more effective code.
- **Set the `onreadystatechange` event handler**: The `onreadystatechange` event is fired every time the readyState property of the XHR object changes. You should set an event handler for this event so that you can handle the response from the server appropriately.
- **Use asynchronous requests**: By default, XHR requests are asynchronous, meaning that they don't block the execution of other code while waiting for a response from the server. This is generally preferable, as it allows your application to continue to respond to user input while the request is being processed.
- **Use error handling**: XHR requests can fail for a variety of reasons, such as network errors or server errors. You should always include an error handling code in your XHR requests to ensure that your application can handle these errors gracefully.
- **Use the correct HTTP verb**: XHR requests can use different HTTP verbs (such as GET, POST, PUT, DELETE, etc.) depending on the type of request you are making. Make sure to use the correct verb for the action you are performing.
- **Send data in the correct format**: When sending data in an XHR request, make sure to send it in the correct format (such as JSON, XML, or plain text) and include the correct Content-Type header.
- **Use a library**: XHR can be a bit verbose and low-level. Consider using a library such as jQuery or Axios to make XHR requests easier to write and more readable.

HTTP

HTTP stands for "Hypertext Transfer Protocol". It is a protocol used for communication between web servers and web clients (such as web browsers). HTTP is the foundation of data communication for the World Wide Web.

HTTP is a stateless protocol, which means that each request and response is independent of any previous requests and responses. However, web applications often need to maintain state across multiple requests, so web developers have come up with workarounds such as cookies and sessions.

COMMON HTTP VERBS

HTTP verbs, also known as HTTP methods, are used to indicate the type of action to be performed on a resource identified by a URL.

Verb	Description
GET	Used to retrieve a resource from the server. Should not modify the server's state and is considered a "safe" and "idempotent" operation.
POST	Used to submit data to be processed by the server. Can modify the server's state and is considered a "non-idempotent" operation.
PUT	Used to update a resource on the server. Should be idempotent.
DELETE	Used to delete a resource on the server. Should be idempotent.
PATCH	Used to modify a resource on the server. Should be idempotent.
OPTIONS	Used to retrieve the supported HTTP methods for a resource.

Verb	Description
HEAD	Similar to GET, but only retrieves the HTTP headers and not the body of the response. Used to retrieve metadata about a resource without actually downloading it.

COMMON MIME TYPES

MIME (Multipurpose Internet Mail Extensions) types are a way of identifying files on the internet according to their nature and format. MIME types were originally designed for email messages, but they are now used in many different contexts, such as HTTP (Hypertext Transfer Protocol) transactions on the World Wide Web.

Mime Type	Description	Example
text/plain	Plain text	.txt files
text/html	HTML document	.html, .htm files
text/css	Cascading Style Sheet (CSS)	.css files
application/javascript	JavaScript	.js files
application/json	JSON data	.json files
image/png	Portable Network Graphics (PNG)	.png files
image/jpeg	Joint Photographic Experts Group (JPEG)	.jpeg, .jpg files
image/gif	Graphics Interchange Format (GIF)	.gif files
application/pdf	Portable Document Format (PDF)	.pdf files
application/xml	Extensible Markup Language (XML)	.xml files
application/zip	ZIP archive	.zip files

AJAX

AJAX (Asynchronous JavaScript and XML) is a web development technique that allows for asynchronous communication between a web browser and a web server, without requiring a page refresh or full reload. AJAX can be used to fetch data from a server and update a web page dynamically, without requiring the user to navigate away from the page.

AJAX ARCHITECTURE

AJAX fits into a broader web development architecture that emphasizes modularity, separation of concerns, and scalability. Specifically, AJAX is often used in conjunction with a client-server architecture, where the client (typically a web browser) sends requests to the server (usually an application server) and receives responses in various formats, such as HTML, XML, or JSON.

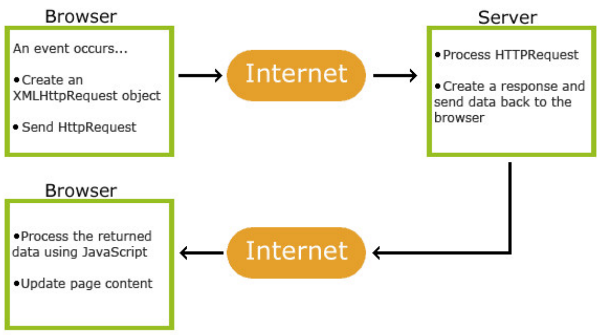
Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

Figure 1. AJAX Architecture.

- **Client**: The client (usually a web browser) sends requests to the server and receives responses in various formats, such as HTML, XML, or JSON.
- **Server**: The server (usually an application server) receives and processes the client's requests, and returns responses containing data or instructions.
- **AJAX**: Asynchronous JavaScript and XML is used to send requests to the server and receive responses asynchronously, without requiring a page refresh or full reload.
- **Model-View-Controller (MVC) pattern**: This pattern separates the application into three parts: the model (data and business logic), the view (user interface), and the controller (mediator between the model and view). AJAX can be used to update the view dynamically.

- **Single Page Application (SPA) pattern**: This pattern contains the entire application within a single web page, and uses AJAX to update the content of the page dynamically as the user interacts with the application.
- **Modularity**: AJAX fits into a broader architecture that emphasizes modularity, separation of concerns, and scalability. By using AJAX in conjunction with other architectural patterns and techniques, developers can create robust and flexible web applications that provide a superior user experience.

FINDING DOM ELEMENTS

To find DOM elements in an HTML document using JavaScript, you can use the `document` object and its various methods.

Method	Description
getElementById	Returns the element with the specified ID.
getElementsByTagName	Returns a collection of elements with the specified tag name.
getElementsByClassName	Returns a collection of elements with the specified class name.
querySelector	Returns the first element that matches the specified CSS selector.
querySelectorAll	Returns a collection of elements that match the specified CSS selector.

Find an element by its ID

```
const element =
document.getElementById('myElement');
```

Find elements by their tag name

```
const elements =
document.getElementsByTagName('div');
```

Find elements by their class name

```
const elements =
document.getElementsByClassName('myC
lass');
```

Find the first element that matches a CSS selector

```
const element =
document.querySelector('#myElement.m
yClass');
```

Find all elements that match a CSS selector

```
const elements =
document.querySelectorAll('.myClass');
```

MODIFYING THE DOM

To modify the DOM in an HTML document using JavaScript, you can use the `document` object and its various methods and properties.

Method	Description
createElement	Creates a new element with the specified tag name.
createTextNode	Creates a new text node with the specified text.
appendChild	Adds a new child element to the end of a parent element’s list of children.
removeChild	Removes a child element from its parent element.
replaceChild	Replaces a child element with a new element.

Method	Description
insertBefore	Inserts a new child element before an existing child element.
setAttribute	Sets the value of an attribute on an element.
getAttribute	Returns the value of an attribute on an element.
removeAttribute	Removes an attribute from an element.

Modify the text content of an element

```
const element =
document.getElementById('myElement');
element.textContent='New text content';
```

Modify the HTML content of an element

```
const element =
document.getElementById('myElement');
element.innerHTML = '<p>New HTML
content';
```

Add a new element to the document

```
const newElement =
document.createElement('div');
newElement.textContent = 'New element';
document.body.appendChild(newElement);
```

Remove an element from the document

```
const element =
document.getElementById('myElement');
element.parentNode
       .removeChild(element);
```

Modify an element's attributes

```
const element =
document.getElementById('myElement');
element.setAttribute('class',
                     'newClass');
```

AJAX TOOLKITS

Ajax toolkits are libraries or frameworks that provide a set of tools and utilities to simplify the process of building AJAX applications. Each of these Ajax toolkits has its own strengths and weaknesses, so it's important to choose the one that best fits your needs and preferences.

Toolkit	Description
jQuery	A fast, small, and feature-rich JavaScript library that simplifies HTML document traversal and manipulation, event handling, and AJAX.
React	A popular JavaScript library for building user interfaces. React uses a virtual DOM and provides a declarative syntax for defining components and updating the view in response to changes in data.
AngularJS	A popular framework for building dynamic web applications. AngularJS provides a declarative syntax for defining HTML templates, and supports two-way data binding, dependency injection, and reusable components.

Toolkit	Description
Vue.js	A progressive JavaScript framework for building user interfaces. Vue.js is designed to be easy to adopt incrementally and scales from small to large applications. It supports reactive data binding, declarative rendering, and component-based architecture.
Ember.js	A framework for building ambitious web applications. Ember.js provides a rich set of features, including templates, routing, controllers, and data persistence. It also has a strong community and ecosystem of add-ons and tools.
Prototype	A JavaScript framework that provides a simple API for performing common tasks, such as DOM manipulation and AJAX. Prototype is known for its concise and readable syntax, and its ability to work with a wide variety of browsers.
Dojo Toolkit	A modular JavaScript toolkit for building dynamic web applications. Dojo provides a comprehensive set of tools for developing complex applications, including data stores, charting libraries, and mobile support.
MooTools	A lightweight JavaScript framework that emphasizes reusable code and extensible classes. MooTools provides a concise and expressive syntax for working with the DOM and AJAX, and supports a variety of browser-specific features.
Backbone.js	A lightweight framework for building single-page applications. Backbone.js provides a simple API for defining models, collections, and views, and supports events and RESTful APIs. It is often used in conjunction with other libraries, such as jQuery and Underscore.js.
Knockout.js	A JavaScript library that simplifies the creation of complex user interfaces with minimal code. Knockout.js uses declarative bindings to connect view elements with data models, and supports two-way data binding and automatic UI updates.

COMMON USEFUL TOOLS

These tools can help developers work more efficiently and effectively when building AJAX applications, by providing useful features such as debugging, automation, collaboration, and testing.

Tool	Description
Developer Tools	Built-in browser tools for inspecting and debugging web pages, including the console for logging and testing JavaScript code.

Tool	Description
Text Editor/IDE	Software for writing and editing code, such as Visual Studio Code, Sublime Text, or IntelliJ IDEA.
Browser Extensions	Extensions for browsers like Chrome or Firefox that provide additional functionality for web development, such as LiveReload or JSONView.
Package Managers	Tools like npm or Yarn for installing and managing dependencies for a project.
Task Runners	Tools like Grunt or Gulp for automating repetitive tasks in the development workflow, such as building, testing, and deploying code.
Version Control	Software like Git or SVN for tracking changes to code over time and collaborating with other developers.
Testing Frameworks	Tools like Jest or Mocha for writing and running automated tests on code.
API Clients	Tools like Postman or Insomnia for testing and interacting with APIs.
Code Quality Tools	Tools like ESLint or Prettier for ensuring consistent code style and preventing common errors.