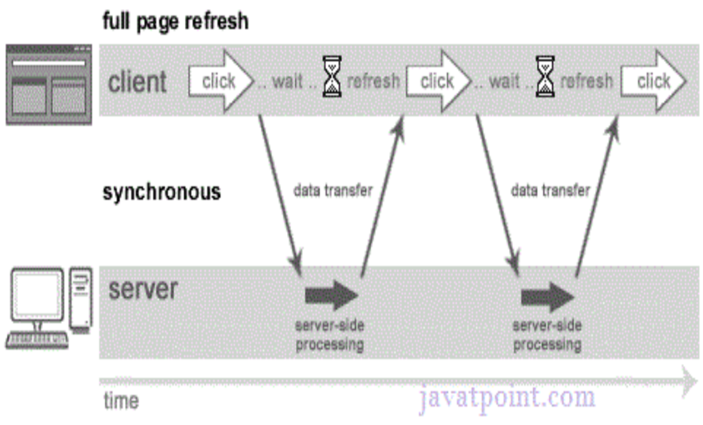

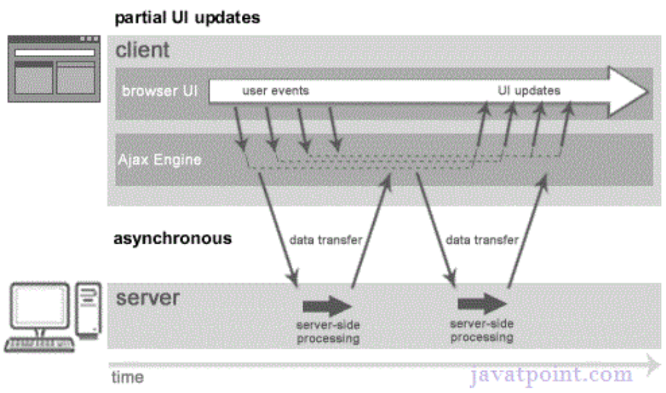

- Javascript engine blocked during synchronous requests
  SPA - Single Page Application -> dynamically rewrites webpage contents with new data based on user's request. Page does not reload or transfer control at any point

XHR - XMLHttpsRequest

User request from UI -> XHR object -> HTTPS request to server -> Data sent back to XHR callback function -> Webpage updated

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Modify Page with responseText</title>
</head>
<body>

    <!-- A div to display the response -->
    <div id="response-container">Loading content...</div>

    <script>
        // Create a new XMLHttpRequest object
        let xmlhttp = new XMLHttpRequest();

        // Initialize the request (GET request, asynchronous)
        let filepath = 'example.txt'; // Replace with your actual file path or URL
        xmlhttp.open("GET", filepath, true);

        // Set the response type as text (default, but specified here for clarity)
        xmlhttp.responseType = "text";

        // Set up the event handler for when the request state changes
        xmlhttp.onreadystatechange = function handler() {
            // Check if the request is complete (readyState 4) and successful (status
200)
            if (this.readyState == 4 && this.status == 200) {
                // Use this.responseText to get the text content of the response
                let responseText = this.responseText;

                // Update the content of a page element with the response
                document.getElementById("response-container").innerText =
responseText;
            }
        };

        // Send the request (null because GET requests don't send data)
        xmlhttp.send(null);
    </script>

</body>
</html>
```

`readyState`	Value	Description
0	UNSENT	The `XMLHttpRequest` object has been created, but the `.open()` method has not yet been called.
1	OPENED	The `.open()` method has been called, and the request has been initialized, but `.send()` has not yet been called.
2	HEADERS_RECEIVED	The `send()` method has been called, and the headers and status code of the response are available.
3	LOADING	The response is in progress, and some data has been received, but the request is not yet complete.
4	DONE	The request has been completed, and the full response is available.

AJAX jQuery Methods

$.ajax()

- Regular asynchronous HTTPS request; used when other specific methods fail

Parameter	Type	Description	Default
`url`	`string`	The URL to send the request to.	Current page URL
`type` or `method`	`string`	The HTTP request method: `"GET"`, `"POST"`, `"PUT"`, `"DELETE"`, etc.	`"GET"`
`data`	`object` / `string`	Data to be sent to the server (for `POST` or `GET` requests).	`null`
`dataType`	`string`	The type of data expected from the server: `"json"`, `"xml"`, `"html"`, `"script"`, `"text"`, `"jsonp"`.	`"text"`
`contentType`	`string`	The content type of the request payload (e.g., `"application/json"`).	`"application/x-www-form-urlencoded; charset=UTF-8"`

Parameter	Type	Description	Default
`success`	`function`	Callback function to be executed if the request is successful.	`none`
`error`	`function`	Callback function to be executed if the request fails.	`none`
`beforeSend`	`function`	Function to be executed before the request is sent (useful for modifying request headers or aborting the request).	`none`
`complete`	`function`	Callback function executed after the request completes (whether it succeeds or fails).	`none`
`timeout`	`number`	Time in milliseconds before the request times out.	`0` (no timeout)
`async`	`boolean`	Whether the request is asynchronous or synchronous.	`true`

```
$.ajax({
    url: "https://api.example.com/data",
    method: "POST",
    data: { name: "John", age: 30 },
    dataType: "json",
    success: function(response) {
        console.log("Success:", response);
    },
    error: function(xhr, status, error) {
        console.log("Error:", error);
    },
    complete: function() {
        console.log("Request complete");
    }
});
```

$.get(url [, data] [, success] [, dataType])

```
$.get('/jquery/getjsondata', // url
{name:'Steve'}, // request parameters
function (data, textStatus, jqXHR) { // success callback function
$('p').append(data.firstName);
},
"json");
```

$.post(url [, data] [, success] [, dataType])

```
$.post("https://api.example.com/users", { id: 1 }, function(response) {
    console.log(response); // Handle the response here
}, "json");
```

$.load()

- Makes a GET request and loads the content from the request directly into a HTML element
  Syntax:

```
$(selector).load(url [, data ] [, callback ] [, dataType]);
```

```
// Load content from "about.html" into a <div> and run a callback on success
$("#content").load("about.html", { id: 1 }, function(response, status, xhr) {
    if (status == "success") {
        console.log("Content loaded successfully!");
    } else if (status == "error") {
        console.log("Error loading content: " + xhr.status + " " + xhr.statusText);
    }
});
```

Fetch

- Network requests
- Works by chaining promises
  Syntax: fetch(url [, options])
- `method` : HTTP request method (`"GET"` , `"POST"` , etc.).
- `headers` : Additional headers like `Content-Type` .
- `body` : The request body for POST/PUT requests.

```
fetch('resp.html')
  .then(function(response) {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.text();
  })
  .then(function(text) {
    mydiv.innerHTML = text;
  })
  .catch(function(error) {
    console.error('There was a problem with the fetch operation:', error);
  });
```

```
// Define the data to send in the POST request
const postData = {
    name: "John Doe",
    email: "john.doe@example.com",
    message: "Hello, this is a test message."
};
```

```
// Make the POST request using fetch
fetch("https://example.com/api/submit", {
    method: "POST", // HTTP method
    headers: {
        "Content-Type": "application/json", // Indicate the format of the payload
    },
    body: JSON.stringify(postData) // Convert the data object to a JSON string
})
    .then(response => {
        // Check if the response is ok (status code 200-299)
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }
        return response.json(); // Parse the JSON response body
    })
    .then(data => {
        // Handle the response data
        console.log("Response received:", data);
        document.getElementById("response-container").innerText =
JSON.stringify(data, null, 2);
    })
    .catch(error => {
        // Handle any errors (e.g., network issues or server errors)
        console.error("Error occurred:", error.message);
        document.getElementById("response-container").innerText = `Error:
${error.message}`;
    });
```