Controller-Server Communication

Sridhar Alagar

Controller's role in MVC

Controller must fetch model (in server) for the view

Other needs, such as authentication, to communicate with the server

Asynchronous JavaScript and XML widely used

- Asynchronously send and receive data without requiring full page reload
- Uses HTTP requests and responses
- Can handle data in many formats – JSON is prevalent today
- XMLHttpRequest – earliest method, but outdated
- Fetch – promise-based, modern and elegant method

XMLHttpRequest

Is a built-in browser object that allows to make HTTP requests in JavaScript

```
// 1. Create a new XMLHttpRequest object
let xhr = new XMLHttpRequest();

// 2. Configure it: GET-request for the URL server-url
xhr.open('GET', 'server-url');

// 3. Send the request over the network
xhr.send();
```

XMLHttpRequest – wait for a response

```
// 4. This will be called after the response is received

xhr.onload = function() {
  if (xhr.status != 200) { // analyze HTTP status of the response
    alert(`Error ${xhr.status}: ${xhr.statusText}`); // e.g. 404: Not Found
  } else { // show the result
    alert (JSON.parse(xhr.responseText));
};

xhr.onprogress = function(event) {
      alert(`Received ${event.loaded} bytes`)
};
xhr.onerror = function() {
  alert("Request failed"); //network error, bad url, etc.
};
```

Fetch API

fetch() is a modern and versatile way to send and get network requests

```
// basic syntax
let promise = fetch(url, [options])

• url: the URL to access
• Options: methods, headers, etc.

Without options GET method by default
```

The browser starts the request right away and returns a promise that the caller should use to get the result

Getting a response is a two-step process

Step 1 of response

Promise fulfils with a response that contains status, but don’t have the body yet

```
let response = fetch(url);
```

Step 1 of response

Promise fulfils with a response that contains status, but don’t have the body yet

```
let response = await fetch(url);

if (response.ok) { // if HTTP-status is 200-299
  // get the response body (step 2)

} else {
  alert("HTTP-Error: " + response.status);
}
```

Step 2 of response

Response provides multiple promise-based methods to access the body in various formats

```
let response = await fetch(url);

if (response.ok) { // if HTTP-status is 200-299
  // get the response body
  let json = await response.json();
} else {
  alert("HTTP-Error: " + response.status);
}
```

Example fetch

Get the latest commit from github

```
let url = 'https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits';

let response = await fetch(url);


// read response body and parse as JSON
let commits = await response.json();


alert(commits[0].author.login);
```

Example fetch using then

Get the latest commit from github

```
fetch('https://api.github.com/repos/javascript-
tutorial/en.javascript.info/commits')
  .then(response => response.json())
  .then(commits => alert(commits[0].author.login));
```

Axios – promise based http client

Transforms JSON data automatically; Handles 404 like errors

```
import axios from 'axios';

axios.get(‘url’)
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
```

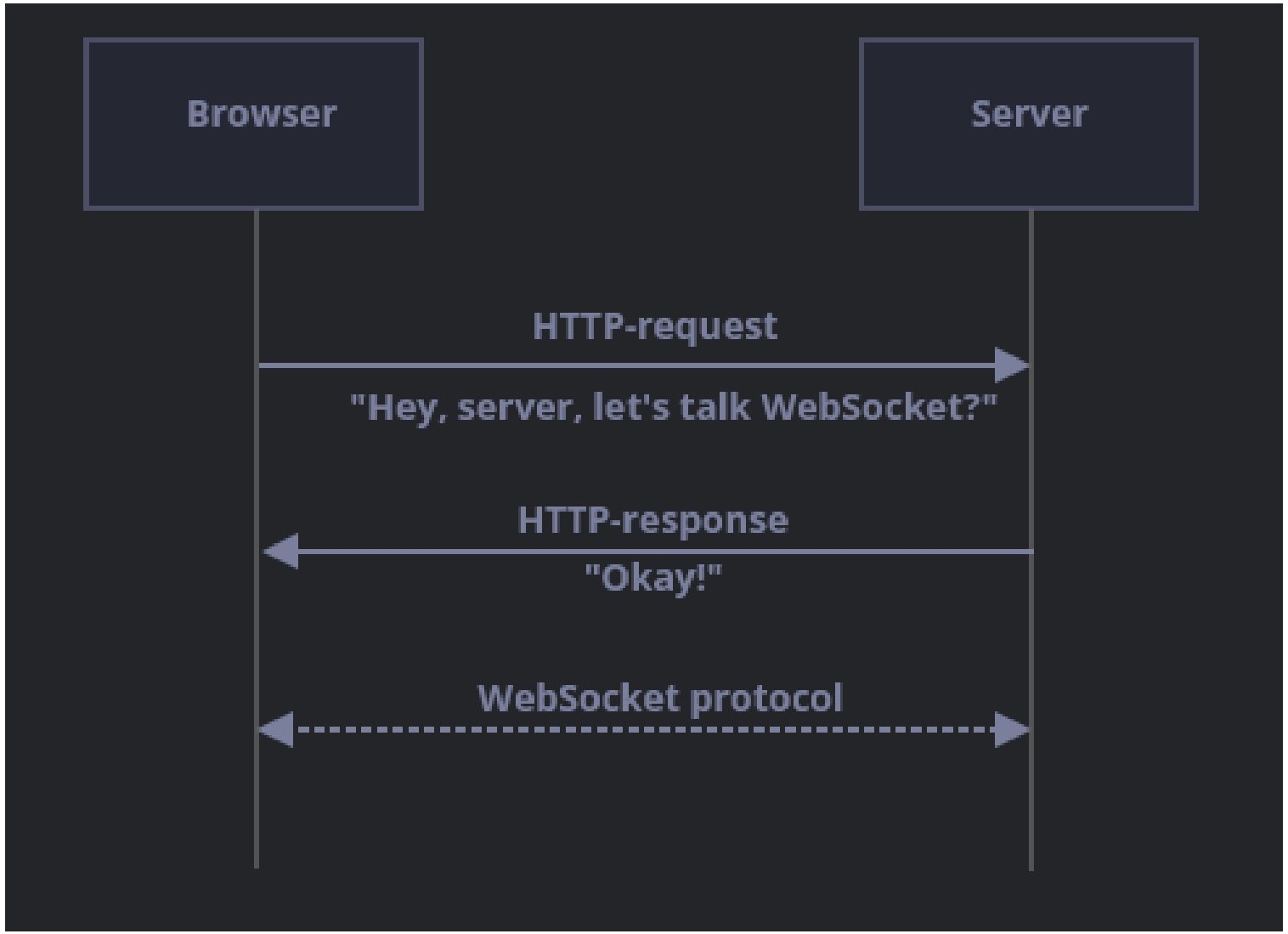
WebSocket

Provides a way to exchange data between browser and server via a persistent connection

The data can be passed in both directions as “packets”, without breaking the connection and the need of additional HTTP-requests

Great for services that require continuous data exchange, e.g. online games, real-time trading systems

Opening a webSocket



Sources

1. Javascript.info - Network requests
2. CS 142 Lectures