Open-Source Software Practice

12. Client-Server Model

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab, College of Computing
Sungkyunkwan University

Schedule

- No Lecture for Weeks 13, 14, and 15

- Final Exam (Week 13, ***Next Week***) @ classroom (85718)
  - Prepare your ID.
  - Bring your laptop or tablet.

- Ask Me Anything (Week 14) @ classroom
  - Ask me anything about what we learned (or what you want to learn in the future).
  - No online session (will not be recorded)

- Presentation (Week 15) @ online
  - Voting

Goals

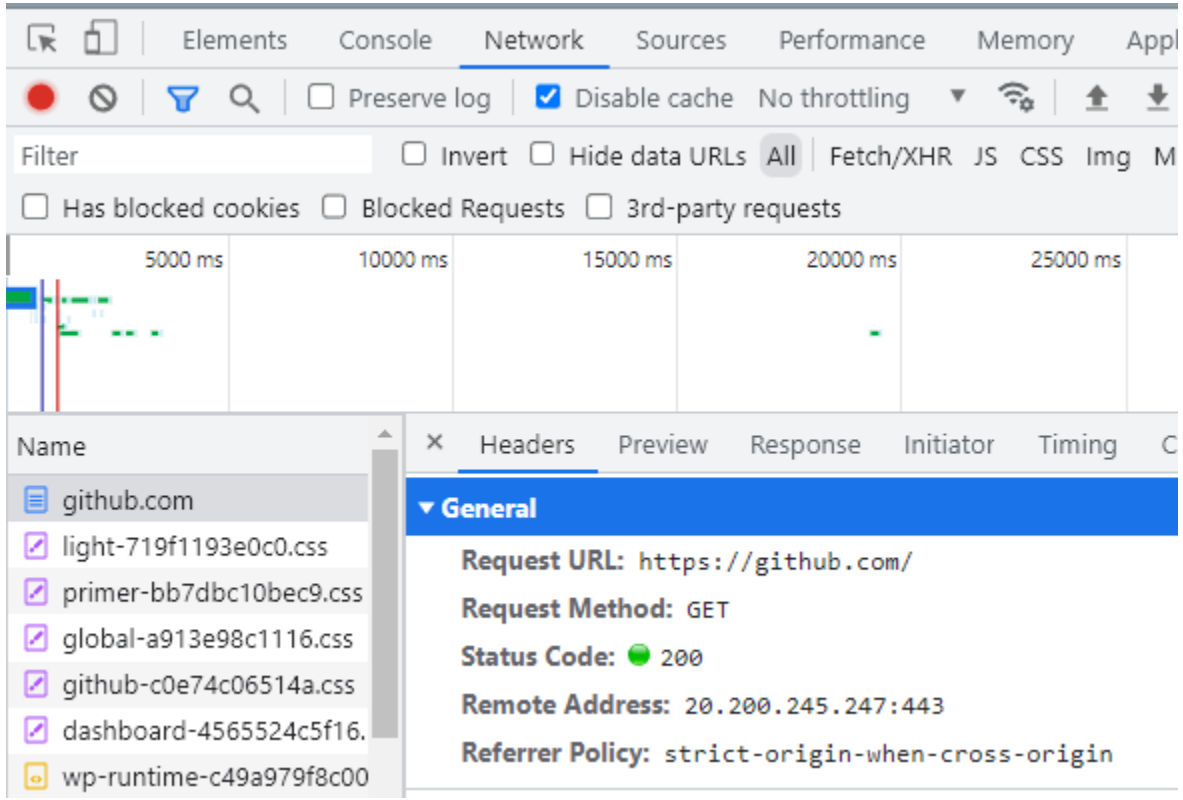
1. Install express.js
2. Looking into an HTTP request
3. Simple Hello World! server
4. Serving a file
5. Visit counter (fetch)
6. A chat server

Let’s Practice – 1 (express.js)

- https://expressjs.com/
- https://github.com/e-/skku-chat

- `npm install express --save`

Let’s Practice – 2 (HTTP)



Let’s Practice – 3 (hello world!)

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

const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello, World!')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

- Import express.js
- Create a new app
- An event-listener-like logic specification
- “If you get a GET request on “/”, send “Hello, World” to the client”
- Run the app on port 3000.
- The process will go into an infinite loop until Ctrl+C.

Let’s Practice – 4 (static files)

- We could send a text message to the client as a response.
- What about sending an HTML code?

- Create a directory “public”.
  - The files under this directory will be directly served to clients.
  - Don’t put any private files., e.g., *main.js* (source code of the server itself)

- Create *index.html* under *public*, and type the code on the next page.

Client Code

```
<!doctype HTML>

<html>

<head>
    <title>SKKU-CHAT</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
</head>

<body>
    <div class="container">
        <p>
            Visits so far: <span id="counter"></span>
        </p>
    </div>

    <script>
    </script>
</body>

</html>
```

Server Code

```
const express = require('express')
const app = express()
const port = 3000

// app.get('/', (req, res) => {
//     res.send('Hello, World!')
// })

app.use(express.static('public'))

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Visits so far:

Let’s Practice – 5 (visit counter)

- Let’s make something useful: visit counter.
- Once the page is loaded, send a request to the server ***again*** (without a refresh!) to load the # of visitors.

- Example: `fetch(`*`url`*`, `*`options`*`).then(`*`handler`*`)`

```
fetch(
  “localhost:3000/counter”,
  {method: “GET”})
.then((count) => { })
```

Client Code

```
<body>
    <div class="container">
        <p>
            Visits so far: <span id="counter"></span>
        </p>
    </div>

    <script>
        window.addEventListener("load", () => {

            fetch("/counter", { method: "GET" })

                .then((res) => res.text())

                .then((count) => {
                    let counter = document.getElementById("counter");
                    counter.textContent = count;
                })
        })
    </script>
</body>
```

- When the page is loaded,
- fetch the “/counter” resource,
- get the result as text,
- and put the text into the #counter element.

Server Code

```
const express = require('express')
const app = express()
const port = 3000

let counter = 0;

app.get('/counter', (req, res) => {
    counter++;

    res.send(counter.toString())
})

app.use(express.static('public'))

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Visits so far: 8

- When someone requests “/counter”,
- increment the global counter variable,
- and returns the counter as text.

Let’s Practice – 6 (chat app)


```
<h2>Chat</h2>
<ul id="chats">

</ul>
<div class="d-flex">
    <input type="text" id="chat" class="form-control"
placeholder="Enter a chat here">
    <button type="submit" id="submit" class="btn btn-
primary">Submit</button>
</div>
```

Client Code - JS

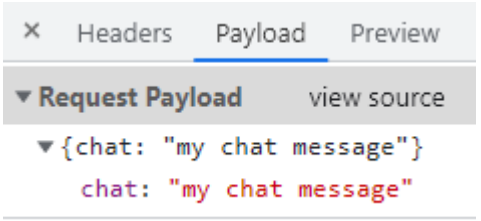
- Getting the messages from the server.
- Clear the list and append the <li> elements for the messages.

```
function loadChats() {
    fetch("/chats", { method: "GET" })
        .then((res) => res.json())
        .then((chats) => {
            let list = document.getElementById("chats");
            list.innerHTML = "";

            chats.forEach(chat => {
                let li = document.createElement("li")
                li.textContent = chat;
                list.appendChild(li);
            })
        })
}
```

Client Code - JS

- When the user clicks on the submit button, read the text in the <input> tag and send it to the server as a JSON object.

```
document.getElementById("submit").addEventListener("click", () => {
    let input = document.getElementById("chat");
    let chat = input.value;

    fetch("/chats", {
        method: "POST",
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ chat })
    })
        .then(() => {
            loadChats();
        })
})
```



Simple Chat Server

```
const express = require('express')
const app = express()
const port = 3000

let counter = 0;

app.get('/counter', (req, res) => {
    counter++;
    res.send(counter.toString())
})

app.use(express.json()) // for parsing application/json

let chats = [];

app.get('/chats', (req, res) => {
    res.send(chats);
})

app.post('/chats', (req, res) => {
    chats.push(req.body.chat);
    res.send(200);
})

app.use(express.static('public'))

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Server Push

- One workaround to this problem is to use “polling”.
- Load the messages from the server every one second.
- Simple but inefficient if there is no update.

```
setInterval(loadChats, 1000); // implements polling
```