Verteilte Systeme im Sommersemester 2022

Patrick Felschen, Matr. Nr. 932056
Julian Voß, Matr. Nr. 934505 Osnabrück, 14.06.2022

Aufgabenblatt 8

Implementierung

Reduzierung des Datenaufkommens

Long Polling: Ein GET-Request zum Abrufen der Tabelle wird erst dann vom Server beantwortet, wenn eine Datenänderung vorkommt. Die Verbindung wird so lange aufrechterhalten, bis dieser Fall eintritt (oder die Timeout Zeit abläuft). Durch das Long Polling lässt sich die Anzahl der Request verringern.

Umstellung auf Json: Die Antwort des Servers beinhaltet nun nicht mehr die vollständige HTML-Tabelle, sondern nur noch die einzelnen Einträge im Json Format. Die Tabelle wird lokal erstellt. Dadurch verringert sich die Größe der versendeten Daten.

billboard.js

```
function postHttpRequest(url) {
    let name = document.getElementById("contents").value;
    let xhr = getXMLHttpRequest();
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    let data = JSON.stringify({
        "name": name
    });
    console.log("POST " + url + " " + data);
    xhr.send(data);
}
```

Der Inhalt des Textfelds „contents“ wird ausgelesen und in ein Json-Objekt umgewandelt. Dieses Json-Objekt besteht aus einem Feld „name“, welches den Wert von „contents“ beinhaltet. Anschließend wird es über ein POST-Request an den Server geschickt.

```
function getHtmlHttpRequest(url) {
    if (!waiting) {
        waiting = true;
        let xmlhttp = getXMLHttpRequest();
        xmlhttp.onreadystatechange = function () {
            if (xmlhttp.readyState === 4 && xmlhttp.status === 200) {
                waiting = false;
                let postersElement = document.getElementById("posters");
                if (postersElement != null) {
                    let json = JSON.parse(xmlhttp.responseText);
                    console.log(json);

                    let table = "<table><tbody>";

                    for (let i = 0; i < json.length; i++) {
                        table += "<tr>";
                        [Tabellen Zeile aus JSON füllen…]
                        table += "</tr>";
                    }
                    table += "</tbody></table>";
                    postersElement.innerHTML = table;
                }
                $('timestamp').innerHTML = new Date().toString();
            }
        };
        xmlhttp.open("GET", url, true);
        xmlhttp.send();
    }
}
```

Die eintreffenden Daten vom Server liegen als Json-Objekte vor. Aus jedem Objekt werden die ID und der Inhalt ausgelesen und in eine Tabelle verpackt. Ist der Nutzer der Ersteller des Eintrags, hat dieser die Möglichkeit, Einträge zu bearbeiten oder zu löschen.

BillBoardServlet.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws […]{
  AsyncContext asyncContext = request.startAsync(request, response);
  asyncContext.setTimeout(10 * 60 * 1000);
  contexts.add(asyncContext);
}
```

Die „doGet“-Methode startet pro GET-Request einen AsyncContext, welcher alle Request- und Response Daten enthält. Zudem wird eine Timeout-Zeit von 10 Minuten gesetzt. Der Client wartet somit 10 Minuten auf eine Antwort des Servers, welche über die „completeContext“ erstellt wird. Alle asynchronen Contexts werden zu einer Liste hinzugefügt, aus welcher sie zu einem späteren Zeitpunkt ausgelesen werden können.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws […]{
  String caller_ip = request.getRemoteAddr();
  List<AsyncContext> asyncContexts = new ArrayList<>(this.contexts);
  this.contexts.clear();
  JSONObject data = getJSONBody(request.getReader());
  String name = data.getString("name");
  bbJson.createEntry(name, caller_ip);
  completeContext(asyncContexts);
}
```

Die Methode zum Hinzufügen von Einträgen speichert sich zunächst die IP des anfragenden Clients ab. Danach werden alle wartenden Contexts in eine Liste kopiert und die Originalliste geleert, dies stellt sicher, dass jeder Context nur ein einziges Mal abgeschlossen wird. Zuletzt werden die JSON-Daten des Requests ausgelesen und inklusiv der IP im BillBoard gespeichert.

Um alle Änderungen auf allen wartenden Clients sichtbar zu machen, wird die „completeContext“-Methode mit der kopierten Context-Liste ausgeführt.

```
private void completeContext(List<AsyncContext> asyncContexts) {
  for (AsyncContext asyncContext : asyncContexts) {
    try (PrintWriter writer = asyncContext.getResponse().getWriter()) {
      String table = bbJson.readEntries(asyncContext.getRequest().getRemoteAddr());
      writer.println(table);
      writer.flush();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      asyncContext.complete();
    }
  }
}
```

Die Methode zum Abschließen der Contexts, iteriert über alle wartenden Clients und liest jeweils die IP aus, um die passenden JSON-Daten vom BillBoardAdapter zu erhalten. Danach werden diese über einen PrintWriter zum Client gesendet. Nach der vollständigen Übertragung wird der jeweilige Context abgeschlossen und somit kann der Client einen neuen GET-Request senden.