WBE-Praktikum 10

Client-Server-Interaktion

Aufgabe 1: Spielzustand auf dem Server (Miniprojekt)

Das im letzten Praktikum implementierte Spiel «Vier gewinnt» soll nun so umgebaut werden, dass der aktuelle Zustand des Spiels auf einem Server gespeichert werden kann.

Damit es nicht zu kompliziert wird, verwenden wir als Server unseren REST Service aus Praktikum 7. Wir speichern einfach den gesamten Spielzustand (Variable *state*) auf dem Server.

Damit wir keine Probleme wegen *Cross Origin Requests* bekommen, laden wir die HTML- und CSS-Dateien vom gleichen Server, der auch die REST-API zur Verfügung stellt. Hinweise zu *Cross Origin* Problemen finden Sie am Ende dieser Praktikumsbeschreibung.

Aufgaben:

- Zunächst zum Server. Wie schon im früheren Praktikum verwenden wir *Express* für den Server. Um auch statische Dateien wie unsere HTML- und CSS-Dateien übertragen zu können, ist eine zusätzliche Middleware nötig. Ausserdem erlauben wir einen zusätzlichen API-Key *c4game*. Sehen Sie sich *index.js* mit den zugehörigen Änderungen an.
- Ergänzen Sie die Datenbank des Servers (Variable *data*) um den Anfangszustand für unser Spiel. Wählen Sie einen geeigneten Schlüssel für den Zugriff.
- Unter *public* ist bereits ein Gerüst von *connect4.html* und das Stylesheet. Ergänzen Sie die Dateien um die nötigen Scripts aus den letzten beiden Praktika.
- Installieren Sie die zusätzlich benötigten Module (*npm install*). Wenn Sie den Server nun starten (*node index.js*) sollten Sie das Spiel laden können (*http://localhost:3000/connect4.html*).
- Nun zum Client. Ergänzen Sie die Seite um zwei Buttons «Laden» und «Speichern».
- Beim Klick auf «Speichern» soll der aktuelle Spielzustand mit PUT auf dem Server gespeichert werden. Dazu können Sie *fetch* verwenden, welches ein Promise-Objekt zurückgibt:

  ```
  fetch(url + "api/data/" + datakey + "?api-key=c4game", {
    method: 'PUT',
    headers: { 'Content-type': 'application/json' },
    body: data
  })
  ```

  Im *body* müssen Sie den zu einem JSON-String konvertierten Zustand übergeben.

- Beim Klick auf «Laden» wird der auf dem Server gespeicherte Spielzustand geladen und das Spielfeld entsprechend neu aufgebaut:

```
fetch(...)
  .then(response => response.json())
  .then(data => { /* use data */ })
```

- Das Spiel sollte nun wieder spielbar sein. Zusätzlich können Sie den Spielzustand auf dem Server speichern und – zum Beispiel in einem anderen Browser – wieder laden.

Aufgabe 2: Spiel-Ende erkennen (Miniprojekt, Abgabe)

Schreiben Sie eine Funktion *connect4Winner*, welche überprüft, ob einer der Spieler gewonnen hat. Sie erhält das Element, für das die Gewinnsituation überprüft werden soll, sowie das Board als zweidimensionales Array. Beispiel:

```
let testBoard = [
    [ '_', '_', '_', '_', '_', '_', '_' ],
    [ '_', '_', '_', '_', '_', '_', '_' ],
    [ '_', '_', '_', '_', 'r', '_', '_' ],
    [ '_', '_', '_', 'r', 'r', 'b', 'b' ],
    [ '_', '_', 'r', 'b', 'r', 'r', 'b' ],
    [ 'b', 'b', 'b', 'r', 'r', 'b', 'b' ]
]

connect4Winner('r', testBoard)        // true, da 4 x 'r' in einer Spalte
connect4Winner('b', testBoard)        // false
```

Am Beispiel sehen Sie, dass alle anderen Positionen ignoriert werden sollen, beim ersten Aufruf also alle 'b' und '_', es spielt also auch keine Rolle. ob leere Felder als '_' oder ' ' dargestellt werden.

Abgabe

Geben Sie die Funktion *connect4Winner* ab.

Abgabeserver:	https://radar.zhaw.ch/python/UploadAndCheck.html
Name Praktikum:	WBE9
Dateiname:	connect4-winner.js
Funktionsname:	connect4Winner
Export im Script:	`module.exports = { connect4Winner }`

Projekt

Integrieren Sie die Funktion *connect4Winner* in das Spiel. Wenn ein Spieler gewonnen hat, soll ein Hinweis ausgegeben und weitere Klicks blockiert werden.

Hinweis: Same Origin Policy

Aus Sicherheitsgründen darf ein Script normalerweise nur mit Ressourcen auf derselben Quelle interagieren, von der es selbst heruntergeladen wurde. Sonst könnte ein Script einer Website, nennen wir sie http://malicous.attacks/you-have-won.html, schlimmstenfalls mit anderen Seiten interagieren, in die wir gerade eingeloggt sind (Facebook, Bank). Der Server der anderen Seite kann einen solchen Zugriff jedoch zulassen, indem er mit einem *Access-Control-Allow-Origin*-Header alle oder bestimmte Quellen zulässt. Dies wird als **Cross-Origin Resource Sharing (CORS)** bezeichnet.

Seite zu CORS des Mozilla Developer Networks:
https://developer.mozilla.org/de/docs/Web/HTTP/CORS