Prof. Dr. Phillip Heidegger
phillip.heidegger@hs-augsburg.de

Version vom: 11.6.2020
Zu Bearbeiten bis: 15.7.2020

Blatt 3

Software Systeme
Webanwendungen

DRAFT

A 3.1: Zeichenprogramm - Eventbasierte Datenquelle 25 Punkte

Modifizieren Sie Ihr Zeichenprogramm, so dass dieses auf eine eventbasierte Architektur aufbaut. In dieser Variante wird die Zeichenfläche durch Events angesprochen, anstelle durch Methodenaufrufe.

- Verwenden Sie das Architektur Prinzip „Event-Sourcing" um den zeitlichen Ablauf Ihres Zeichenprogramm aufzuzeichnen. Zeigen Sie kontinuierlich den Event-Stream in einem Textfeld an.
- Zwischenschritt für die eigene Kontrolle: Fügen Sie einen Knopf zu Ihrem Programm hinzu, mit dem Sie den Event-Stream aus dem Textfeld „laden" können. D.h. Sie können durch Bearbeiten des Event-Streams in Textform „Zeitreisen" in Ihrem Programm realisieren. (Dieses Textfeld könne Sie wieder entfernen, wenn Sie die Multiuser Aufgabe umsetzten.)

A 3.2: Verschieben von Zeichenobjekten 25 Punkte

Ermöglichen Sie dem User das Verschieben der Zeichenobjekten mithilfe von Drag&Drop. Nutzen Sie hierbei die Selektionsmöglichkeit aus dem vorherigen Blatt.
Hinweis: Realisieren sie das Verschieben durch Löschen gefolgt von einem anschließenden Erstellen eines neuen Shapes. Dies erleichtert die folgende Aufgabe.

A 3.3: Multiuser 150 Punkte

Erstellen Sie nun für Ihr Programm ein Backend, so dass mehrere Personen in der Lage sind, gleichzeitig an einer Zeichenfläche zu arbeiten.

- Ihr Backend soll die benötigten Ressourcen, d.h. .html, .js, .css etc. Dateien, an den Client (Browser) ausliefern.
- Ihr Client soll eine Single-Page-Webanwendung sein. Sie sollen für alle gültigen URLs den Inhalt einer generischen HTML-Datei ausgeliefert. Auf dem Client soll dann abhängig von dem Pfad eine von zwei Seiten angezeigt werden.

  - Eine Übersichtsseite `http://.../`
  - Eine Seite, die die Zeichenflächen darstellt. `http://.../canvas/<ID>`

  Auf jeder dieser Seiten wird der Client direkt eine Websocketverbindung zum Server aufbauen (URL: `ws://.../channel/`). Sobald eine Websocket Verbindung zwischen Client und Server hergestellt ist, sollte der Server dem Client eine eindeutig ID zuordnen, und dem Client diese mitteilen, z.B. durch Schicken von:

  ```
  { "clientId": "XYZ......" }
  ```

  Diese ID sollte der Client nutzen, um IDs für Shapes zu generieren. Auf diese Art und Weise kann sichergesestellt werden, dann nicht zwei Clients auf der selben Canvas zwei Shapes mit den selben IDs erzeugen.

  Falls Sie dies wünschen, können Sie diese ID zusätzlich noch in einem JWT ablegen (wie genau, dürfen Sie selber entscheiden), so dass diese ID dauerhaft den Clients zugeordnet werden kann. Der Server sollte in diesem Fall beim Aufbau des Websockets die ID, die vom Client vorgeschlagen wird, übernehmen und zusätzlich in der oben beschriebenen Form dem Client mitteilen. Sie müssen hier keinerlei Integritätssicherheit umsetzten.

- Ihr Backend werden mehrere Zeichenflächen, die durch IDs identifiziert werden, unterscheiden. Diese werden jeweils unter URLs der folgenden Form bereitgestellt:

  `http://<IhrHost>/canvas/<ID>`

  D.h. wenn ein Nutzer die URL `http://.../canvas/ganzLangeId` im Browser öffnet, sollte ein Zeichenprogramm sichtbar werden, dass ähnlich aussieht wie das Zeichenprogramm aus der vorherigen Aufgabe.

  Ruft aber nun ein weiterer Nutzer die URL `http://.../canvas/ganzLangeId` auf, so können diese beiden auf der selben Zeichenfläche gemeinsam zusammenarbeiten.

  Damit niemand die IDs erraten kann, sollten diese IDs zufällig generiert werden, und möglichst nicht zu kurz sein.

  Es sollte eine Navigationsmöglichkeit von den Zeichenfläche zu der Übersichtsseite geschaffen werden. Diese sollte ohne "reset" der JS-VM umgesetzt werden (bitte eine SPWA).

- Auf der Übersichtsseite (URL: `http://..../`, d.h. ohne Pfad) sollte eine Liste der existierenden Zeichenflächen sichtbar sein.[1]

[1] Hier wäre später noch denkbar, diese Welcome Page zu personalisieren, so dass Nutzer nur die Zeichenflächen sehen, die diese "kennen". Da die IDs zufällig sein sollen, und lang genug sind, könnte auf diese Art und Weise ein Management der Zeichenflächen umgesetzt werden. Langfristig natürlich noch mit einen Rechtemanagmentsystem. Diese Aspekte dürfen Sie alle ignorieren, da dadurch die Aufgabe zu umfangreich wird.

Außerdem sollten auf der Überssichtsseite eine Möglichkeit vorhanden sein, eine neue Zeichenfläche zu erzeugen. Dies bedeutet, dass der Server eine neue ID generiert, diese dem Client mitteilt, und dieser dann die passende Zeichenfläche anzeigt. (Ohne Refresh! SPWA!)

- Wie oben erwähnt, soll zur Umsetzung der notwendigen Kommunikation zwischen Client und Server während der Sitzung der Websocket (`ws://.../channel/`) eingesetzt werden. Über diesen werden Anfrage vom Client an den Server gestellt, am besten in Form von JSON-Anfragen. Ein Bsp. für eine Anfrage des Clients an den Server mit der Bitte, den Client von nun an über alle Events, die die Zeichenfläche mit der ID `ganzLangeId` betreffen, zu informieren:

```
{
  "command": "registerForCanvas",
  "canvasId": "ganzLangeId" }
```

Im Fall, dass ein Client sich für eine Zeichenfläche registriert, sollte der Server z.B. so antworten:

```
{ "canvasId": "ganzLangeId",
  "eventsForCanvas": [
    { "event": "addShape",
      "shape": { "type": "line",
                 "id": "ID1",
                 "data": { "from": { "x": 5, "y": 5 },
                           "to": { "x": 100, "y": 100 }
                           "zOrder": 5,
                           "bgColor": undefined, // transparent
                           "fgColor": "000000"   // black
                         }
               }
    },
    ...
  ]
}
```

Ab diesem Zeitpunkt muss der Server, jedes Mal wenn ein anderer Client die Zeichenfläche manipuliert, den Client über Änderungen informieren, indem der Server passende Infopacket verteilt.

Wenn im Client der Nutzer dann wieder auf die Welcome Seite wechselt, sollte durch

```
{ "command": "unregisterForCanvas",
  "canvasId": "ganzLangeId" }
```

die Registrierung durch den Client für die Zeichenfläche entfernt werden. Ab diesem Zeitpunkt wird der Server den Client nicht mehr über Änderungen an der Canvas informieren.

Wenn ein Client eine Zeichenfläche anzeigt und der Nutzer Shapes erstellt, verschiebt, etc., muss der Client den Server über alle Änderungen informieren, so dass dieser die Informationen an alle anderen Clients, die die Zeichenfläche auch anzeigen, informieren kann. Ein Bsp. für das Verschieben eines Shapes:

```
{ "canvasId": "ganzLangeId",
  "eventsForCanvas": [
    { "event": "removeShape",
      "shapeId": "ID1",       // s.o., die Linie (5,5) -- (100,100)
    },
    { "event": "addShape",
      "shape": { "type": "line",
                 "id": "ID2",
                 "data": { "from": { "x": 25, "y": 25 },
                           "to": { "x": 120, "y": 120 },
                           "zOrder": 5,
                           "bgColor": ...,
                           "fgColor": ...
                         }
               }
    },
  ]
}
```

Als Konsequenz sollte auf allen Zeichenprogrammen, die die Canvas mit ID `ganzLangeID` anzeigen, die Linie von `(5,5)-(100,100)` um 20 Pixel nach recht und 20 Pixel nach unten bewegt werden (So wird es für den User aussehen, auch wenn es in Wirklichkeit einfach die eine Linie entfernt wird, und eine neue erstellt wird mit den passenden Koordinaten. Dieses Verhalten ist passend zu der API des Zeichenprogramms, da die Canvas nur mit funktionalen Shapes arbeitet. Dies sollten Sie nicht ändern!).

- Empfehlung: Versuchen Sie, die Anzahl der Events auf der Canvas möglichst gering zu halten. Events für die folgenden vier Interaktionen sollten reichen:
  - AddShape — Infos: shape type, und die Daten des shapes
  - RemoveShapeWithId — Infos: die id
  - SelectShape — Infos: shapeId, clientId
  - UnselectShape — Infos: shapeId, clientId