Collaborative Data Visualizer

Rolf Oberhänsli
Benjamin Kern

May 13, 2022

1 Sprachen, Tools, Frameworks

- Frontend
  - Handlebars / HTML
  - CSS
  - D3.js for data visualization
- Backend
  - Node
  - Express
  - Socket.io
- Datenbank
  - Mongoose
  - MongoDB

2 Architektur

Die Applikation implementiert das MVC Pattern.

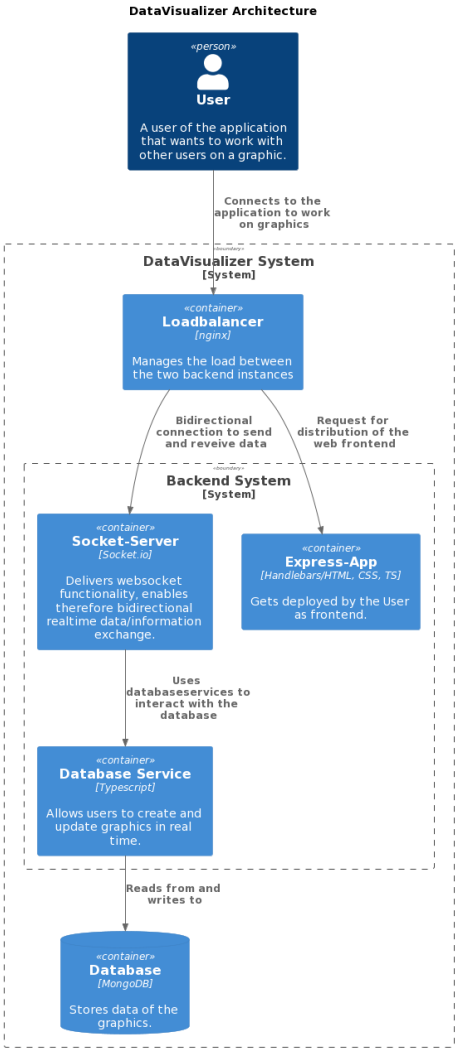

Figure 1: Architektur Datavisualizer

3 Komponenten

3.1 Loadbalancer

Als Loadbalancer verwenden wir einen NGinx-Webserver mit einer Konfiguration, die der Architektur entspricht. Dabei wurden zwei Backendinstanzen als Upstream definiert. Zudem wurde die Konfiguration so vorgenommen, dass Sticky-Sessions verwendet werden.

3.2 Backend

Im Backend wird die Verteilung der Frontendressourcen, der Websocketserver sowie die Interaktion mit der Datenbank gesteuert.

Router Die Verteilung von statischen Files (JS Skripte stylesheets) erfolgt über die traditionelle statische Express Route. Alle anderen Routes werden hier zu einer Methode im Controller gemappt.

Controller

Index-Seite Geht der User auf die Index Seite, so wird die entsprechende Methode im Controller aufgerufen, welche alle zur Verfügung stehenden Dokumente (bzw. dere IDs) aus der Datenbank holt und via Handlebars in die Dropdown-Liste abfüllt. Wählt der Nutzer ein bestehendes Dokument zum Öffnen, so wird die ID entsprechend gesetzt und er auf die Dokumentseite weitergeleitet. Falls das Dokument noch nicht existiert, so wird eine Anfrage auf /createNewDocument abgesetzt, welche das Dokument erstellt und nach Erstellung auf die Dokumentseite weiterleitet. Falls der Nutzer ein Dokument wählt, das unterdessen gelöscht wurde, so verweilt er auf der Index-Seite.

Dokument-Seite Die ID des Dokumentes wird als Query Parameter in der URL übergeben, sodass im Frontend im JS basierend auf dieser dem entsprechenden Socket Raum beigetreten und Updates gesendet/empfangen werden können. Basierend auf der ID des Dokumentes greift der Controller auf den Datenbank-Service zu, um das entsprechende Dokument zu erhalten und via Handlebars in die HTML Tabelle abzufüllen.

Websocketserver Der Websocketserver läuft innerhalb derselben Instanz wie die Express App. Es existieren die folgenden Websocket Endpunkte:

- updateTable: Änderungen werden in der Datenbank gespeichert und anschliessend an alle Clients im selben Raum (Clients, die am selben Dokument arbeiten) gesendet.
- deleteDocument: Es wird überprüft, ob niemand anders im Raum des Dokumentes ist. Falls dies der Fall ist, wird ein "deleteSuccessful" emitted. Schlägt der Löschvorgang fehl, so wird ein "roomNotEmpty" emitted.

- joinRoom: Über diesen kann ein Client einem Raum beitreten.

Websocket Raumsynchronisation Leider konnte dies innerhalb der gegebenen Zeit nicht implementiert werden. Da die Sockets nur per Backend (und nicht backend-übergreifend) sind, und beispielsweise für den Löschvorgang relevant ist, ob jemand noch am Dokument arbeitet, müssten die Räume (bzw. mindestens die Anzahl der aktiven Nutzer in einem Raum) von Backend zu Backend synchronisiert werden. Hierfür wurden 2 Ansätze getestet, welche allerdings beide fehlschlugen:

- Change Streams (mehr dazu hier), mit der Änderungen an der Datenbank ein Event auslösen auf welches dann reagiert werden kann. Das Problem hierbei ist, dass die MongoDB innerhalb eines Replica-Sets laufen muss, zu welchem wir die Verbindung innerhalb eines Docker Containers nicht herstellen konnten (evtl. sind auch neuere Versionen von MongoDB bzw deren Replica-Sets zur Zeit inkompatibel mit Docker).
- MongoDB Adapter (mehr dazu hier). Auch dieser Ansatz schlug leider fehl.

Details zu den Ansätzen, sowie die Entwicklungsfortschritte können der Commit-Historie entnommen werden.

3.3 Frontend

3.3.1 Seiten

/ "Index" Hierbei handelt es sich um die Einstiegsseite der Applikation. Der Nutzer hat entweder die Möglichkeit ein neues Dokument durch einen neuen Namen zu erstellen, oder ein bereits bestehendes Objekt aus einer Dropdown-Liste zu wählen. Nachdem er eine der Option ausgewählt hat und das Dokument falls notwendig durch das Backend erstellt wurde, wird der Nutzer auf die Dokumentseite weitergeleitet.

/document Die Seite besteht aus der Tabelle, in der der Nutzer Daten manipulieren eingeben kann, und der gerenderten Grafik. Der User kann Reihen zur Tabelle hinzufügen, aber keine Kolonnen. Grund hierfür ist, dass die Applikation darstellungstechnisch zur Zeit nur 2 dimensionale Bar Charts unterstützt.

3.3.2 Websocket Endpunkte

- updateTable: Änderungen werden in die HTML Tabelle geparst und anschliessen durch d3.js gerendert.
- roomNotEmpty: Mutation (zur Zeit kann dies nur bei einem Löschversuch auftreten) ist fehlgeschlagen. Es wird die Fehlermeldung angezeigt, die vom Backend durchgereicht wird.

- deleteSuccessful: Das Dokument wurde erfolgreich gelöscht. Da dieses nun ja nicht mehr existiert, wird der Client wieder auf die Index-Seite weitergeleitet.

3.3.3 Dokument Mutationen

Der Nutzer kann nun die Daten in der Tabelle anpassen. Sobald der Nutzer mit den Änderungen zufrieden ist, betätigt er den Button "Update Table", wodurch einerseits die Grafik mit den aktuellen Daten neu gerendert wird und andererseits durch den Websocket alle Änderungen an alle anderen gesendet werden, die zur Zeit an diesem Dokument arbeiten. Dadurch, dass dies über den Websocket läuft müssen andere Teilnehmer nicht die Seite aktualisieren, um die neuste Version zu erhalten, sondern sie erhalten Änderungen in Echtzeit. Um sicherzustellen, dass alle Nutzer nur Updates ihres derzeitigen Dokumentes erhalten, wird beim Laden der Seite zuerst eine Anfrage an den Server geschickt, um dem entsprechenden Socket Raum beizutreten. Socket Räume tragen den Namen der Dokumentid.

3.3.4 Löschen des Dokumentes

Falls der Button zum Löschen des Dokumentes betätigt wird, so wird über den Websocket eine Anfrage zur Löschung geschickt. Falls andere Nutzer noch an dem Dokument arbeiten bzw. noch andere Nutzer im gleichen Raum sind (Vorsicht: Dies funktioniert zur Zeit nur per Backend, nicht backend-übergreifend, wie oben erwähnt), erfolgt die Antwort durch den Websocket woraufhin das Frontend dem Nutzer einen Alert anzeigt, dass das Dokument nicht gelöscht werden kann. Andernfalls wird das Dokument gelöscht und das Frontend leitet den Nutzer wieder zurück auf die Index-Seite.
Anmerkung: Da es sich hierbei um einen Conditionalreroute handelt (der Nutzer soll nur auf die Indexseite geleitet werden, falls das Dokument gelöscht werden konnte) geschieht dies nicht wie alles sonstige Routing durch den Controller im Backend, sondern im Frontend durch Ändern der window location.

3.4 Datenbank

Als Datenbank nutzen wir Mongodb, welche über Mongoose angebunden ist. In der Datenbank werden die Graphdokumente abgespeichert, ausgelesen und aktualisiert. Die Struktur der Datenbank wurde als Model respektive Schema festgelegt.

3.5 Docker

Die gesamte Applikation ist auf Docker-Container verteilt.

- Zwei Backendinstanzen, welche über das Environment eine APPID erhalten. Diese APPID wird anschliesseend als Serverport verwendet.

- Eine Nginx Instanz, welche als Loadbalancer fungiert. Der Loadbalancer ist von aussen über den Port 8080 erreichbar und routet diesen intern auf Port 80, welcher wiederum die Anfragen an eines der Backendinstanzen weiterleitet.
- Eine Datenbankinstanz: Sie wird über den Adapter vom Backend angesteuert
- Backend1, Port: 1111
- Backend 2, Port: 2222
- Nginx Webserver, Port: 8080:80
- Mongodb, Port: 27017

4 Design Entscheidungen

Verwendung von Models Damit Graphdokumente immer dieselbe Struktur haben, nutzen wir ein Model. Über dieses können auf der Datenbank nur identische Dokumente erstellt werden. MongoDB ermöglicht sonst das Speichern von Unstruktierten Dokumenten (wie es auch bei NoSQL der Fall sein soll), allerdings ist dies für uns von Nachteil. Durch das Schmema bleiben die von traditionellen Datenbanksystemen bekannten Vorteile auch in diesem Anwendungsfall erhalten. Mehr zur Wahl der MongoDB im folgenden Abschnitt.

Wahl der Technologien Durch die Wahl von JS (bzw. TS) im Backend sowie im Frontend können Dokumente sehr einfach durchgereicht werden und es muss wenig Aufwand betrieben werden, um die Dokumente/Daten in eine derzeit passende Form zu bringen. Eine weitere solche Entscheidung war die Wahl der MongoDB, da diese ein direktes Abspeichern der Dokumente (die in JSON vorliegen) erlaubt. Hätte man eine relationale Datenbank verwendet, so hätten Objekte bei jedem Schreib- oder Lesezugriff geparst werden müssen.

Websocket vs Controller Jeglicher Informationsaustausch, der nie ein Aktualisieren der Seite im Frontend verlangt, erfolgt über den Websocket. Alle anderen Aktionen, welche zwangsläufig zu einem Seitenwechsel führen, werden durch den Controller kontrolliert. Diese klare Definition hilft der Wartbarkeit des Programmes, sowie der Verständlichkeit