WebLab: Projektarbeit

1. Technologie

Für das Frontend fiel unser Entscheid auf Angular, weil wir den SPA-Ansatz als ideale Lösung für die Realisierung des Travelblogs sahen. Durch eine SPA wird das Rendering beim Client vorgenommen, wodurch der Server entlastet werden kann und eine bessere Entkopplung entsteht. Der Nachteil einer schlechten SEO ist unserer Ansicht nach nicht weiter tragisch. Ausserdem sind wir beide noch unerfahren im Webbereich und konnten so maximal vom Input aus den Lektionen profitieren, indem wir die Theorie direkt in der Praxis anwenden konnten.

Ebenso entschieden wir uns beim Server für die Variante aus dem Unterricht mit Node.js zusammen mit dem Express Framework. Mithilfe dieses Frameworks haben wir sowohl den Daten-Server (Travelblog-Server) als auch den Authorization-Server (Auth-Server) realisiert, wobei wir beim Letzteren noch zusätzlich JSON Web Tokens verwendet haben. Diese beiden Server bilden die Schnittstelle zwischen Client und Datenbank und stellen den Business-Layer dar.

Als Datenbank verwendet unsere Implementation MongoDB. Ein Grund dafür ist die schlichte Anbindung an unseren Server, welche wir bereits im Unterricht behandelt haben. Weiterhin ist die Verwendung frei von jeglichen Kosten und gibt Aufschluss auf Datenbank-Verbindungen – diese unterstützten uns bei der Fehlersuche, als unsere Ladezeiten für den Travelblog unerklärlich anstiegen.

Architektur

Der Travelblog-Client greift mittels Http-Requests auf den Auth-Server zu, um den User zu authentifizieren und einen JWT-Token zu erhalten. Der Client verwendet diesen Token im weiteren Verlauf als Autorisierungsmethode. Auch über Http-Requests hat der Client Zugriff auf den Travelblog-Server. Von diesem erhält er Zugang zu den Travelblogs und kann diese erstellen, aktualisieren und löschen. Als Persistenz wird eine Mongo-DB mit zwei Collections (User & Travelblogs) verwendet.

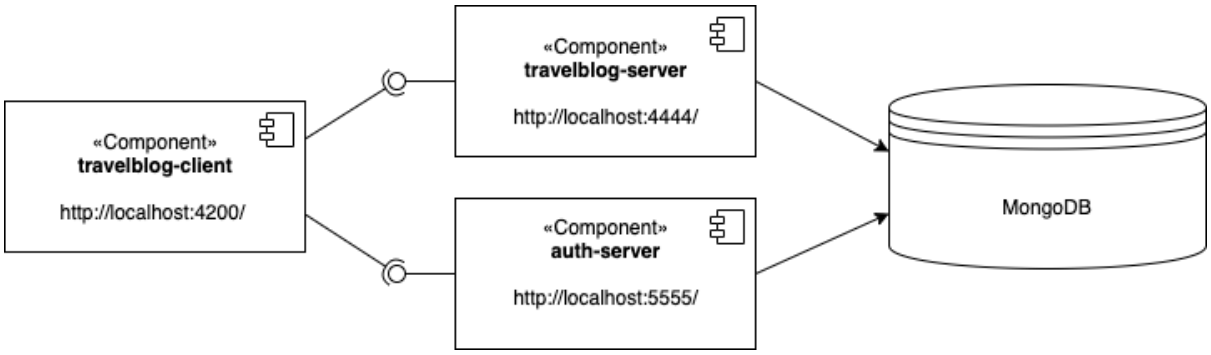

Abbildung 1: Architektur-Diagramm

2. Methodik

Am Anfang des Projektes einigten wir uns auf einige Key-Features, welche als Grundlage des Projekts dienen sollen. Das Resultat dieser Diskussion war die Navigation der Webseite (Routing): Travelblogs, Meine Blogs, Neuer Blog, Authentifizierung und Autorisierung. Anhand dieser Anforderungen konnten wir die benötigten Http-Requests definieren, welche der Travelblog-Server zur Verfügung stellen muss:

Request	URL	Beschreibung
GET	.../travelblog/:id	Liest einen spezifischen Travelblog aus der Datenbank
GET	.../travelblogs	Liest alle vorhandenen Travelblogs aus der Datenbank
POST	.../travelblog	Erstellt einen neuen Travelblog in der Datenbank
DELETE	.../travelblog/:id	Löscht einen spezifischen Travelblog aus der Datenbank
PUT	.../travelblog	Aktualisiert einen Travelblog

Zu diesem Zeitpunkt sind die Zugriffsrechte noch nicht relevant, weshalb auch keine Unterscheidung dieser gemacht wird. Erst im nächsten Schritt, wenn die Authentifizierung implementiert ist, wird bei den Requests, welche nicht nur lesen, eine Autorisierung durchgeführt. Die API des Auth-Servers wird erst in einem späteren Punkt definiert, weil wir uns über die Implementation noch nicht im Klaren sind. Aufgrund unserer modularen Architektur ist dies auch noch nicht notwendig.

Travelblog-Server

Mangels Erfahrung mit Node.js respektive Servern im Allgemeinen entschieden wir uns für gemeinsames Arbeiten am Server durch Pair Programming. Dadurch erhöhten wir nicht nur die Qualität des Codes, auch beide Teammitglieder konnten grundlegende Praxiserfahrungen mit Node.js und Express sammeln. Nebst diesen Punkten konnten wir auch eine gemeinsame Grundlage für die Anbindung des Clients schaffen.

Aus ähnlichen Gründen wie zuvor implementierten wir die MongoDB parallel dazu. Das Ziel war es, manuelle Http-Requests im Postman abzuschicken und entsprechendes Feedback zu erhalten. Schliesslich erstellen wir simple Tests im Postman, um die verschiedenen Requests auf ihr Verhalten zu überprüfen.

Auth-Server

Anders als beim Travelblog-Server, wurde am Auth-Server autonom gearbeitet. Dies bedeutet, dass ein Teammitglied eigenständig daran gearbeitet hat. Diese Implementation mittels JWT verlangte eine extensive Recherche, weswegen sich Pair Programming ohnehin nicht geeignet hätte. Ausserdem wird dieser Server nicht zwingend für die Kommunikation zwischen Client und Datenbank benötigt und kann daher auch zu einem späteren Zeitpunkt durch Interception in die Http-Requests integriert werden.

Travelblog-Client

Damit möglichst bald beide Teammitglieder am Travelblog-Client autonom arbeiten können, erstellten wir in einem ersten Schritt ein Routing mit den entsprechenden Pages, wodurch das Skelett definiert ist. Jetzt können erste Pages voneinander unabhängig bearbeitet werden. Zur Veranschaulichung der Änderungen wurde ein erstes Layout mit Navigation erstellt, welche auf die verschiedenen Pages verweist.

In einem nächsten Schritt wurden die Model vom Travelblog und BlogEntry als Klassen erstellt, weil diese über die Pages hinweg verwendet werden. Ähnlich übergreifend sind die Services, welche nach Bedarf hinzugefügt beziehungsweise den Bedürfnissen entsprechend erweitert wurden. So wurden auch Auth-Service und Auth-Interceptor bei der Realisierung des Auth-Servers implementiert.

3. Reflexion

Im Rückblick ist uns die Zusammenarbeit äusserst gut gelungen. Durch gleiche Arbeitszeiten konnten wir uns gegenseitig motivieren und bei Bedarf unterstützen. Das Pair Programming war zwar zeitaufwändig, dafür konnten wir beide voneinander profitieren und gemeinsam ein stabiles Fundament legen. ***

Weil wir jedoch möglichst parallel arbeiten wollten, kam oftmals nur der Sonntag in Frage. Aus diesem Grund wurden unsere Arbeitseinheiten lange, was auch in der Qualität des Codes sichtbar war. Hier hätte man einzelne Arbeitspakete besser auf verschiedene Tage verteilen können. Da wir beide jedoch berufsbegleitend studieren, war dies leider nicht immer möglich.

Umso weiter das Projekt fortgeschritten war, desto mehr Aufwand mussten wir aufwenden, um Probleme zu lösen – diesen Aufwand haben wir unterschätzt. Hierbei hätten frühzeitige Implementationen von automatischen Tests Abhilfe geschaffen. Bereits Tests, durchgeführt mit dem Postman, haben die API des Travelblog-Servers erfolgreich überprüft und nach Anpassungen der Logik auftretende Fehler entdeckt.

Trotz den Herausforderungen konnten wir während des gesamten Projekts einen wachsenden Lernerfolg erkennen. Der stetige Austausch miteinander hat unser Verständnis für die Webentwicklung geschärft und wir lernten voneinander neue Herangehensweisen zur Lösung von Problemen.

4. Benutzeranleitung

Unsere Applikation befindet sich auf zwei verschiedenen Repositories:

Name	URL
Travelblog-Client	https://github.com/reQQuiem/H20_WebLab_Client
Travelblog- und Auth-Server	https://github.com/reQQuiem/H20_WebLab_Server

Folgende Kommandos müssen zum Bereitstellen der Funktionalitäten unserer Travelblog-Applikation in der angegebenen Reihenfolge ausgeführt werden:

H20_WebLab_Server

Zur Installation der benötigten Dependencies:

```
npm install
```

Travelblog-Server:

```
cd .\travelblog-server\
node .\travelblog-server.js
```

Auth-Server (neues Terminal):

```
cd .\travelblog-server\
node .\auth-server.js
```

Bei den Servern kann alternativ auch "npx nodemon" statt "node" verwendet werden. Die beiden Server stehen jetzt unter http://localhost:5555/ (auth) respektive http://localhost:4444/ (travelblog) bereit.

H20_WebLab_Client

```
cd .\travelblog-client\
npm install
ng serve
```

Der Client steht nun unter http://localhost:4200/ bereit. Für das Login auf der Webseite werden folgende zwei User bereitgestellt:

Name	Passwort
Colin	password
Mischa	password

*Zugriff auf die MongoDB können wir auf Anfrage bereitstellen.

5. Arbeitsjournal: Colin Dexheimer

Datum	Tätigkeiten	Dauer [h]
13.02.2021	Besprechung der Http-Requests die vom Client zum Server möglich sein sollen	1
21.02.2021	Umgebung Einrichten / Repo sharen	1
	Aufsetzen MongoDB	0.5
	Collection in MongoDB für Travelblog erstellen	0.5
	Postman installieren / einrichten / Requests erstellen	1
	Aufsetzen Travelblog-Server mit node.js und express	1.5
	Travelblog Repository implementieren / anpassen	2.5
	MongoDB im Travelblog-Server verwenden	1
	Environment-File aufsetzen	1
28.02.2021	Authentifiziereung mittels JWT auf Serverseite implementieren.	7.5
	Implementierung Travelblog-Service Client-Seite	1
	Problembehebung Angular-Materials	0.5
03.03.2021	Implementierung Authentifizierung Client-Seite (Local Storage, Auth-Service, Auth-Interceptor etc.)	5
	Implementierung Angular Login-Component	1
	Delete Travelblog Logik Client-Seitig	3
07.03.2021	Name im Local-Storage speichern (Für Filter der Blogbeiträge)	1
	Implementation Http-Calls Client-Seitig	1
	Url-Parameter für Filterung einbauen	4
	Allgemeines Review / Fixen (GUI, DB-Connection Aufbau etc.)	2
09.03.2021	Dokumentation / Arbeitsjournal	3
	Artefakte zur Abgabe vorbereiten	1
Total		**40**

6. Arbeitsjournal: Mischa Kälin

Datum	Tätigkeiten	Dauer [h]
13.02.2021	Besprechung der Http-Requests die vom Client zum Server möglich sein sollen	1
21.02.2021	Umgebung Einrichten / Repo sharen	1
	Aufsetzen MongoDB	0.5
	Collection in MongoDB für Travelblog erstellen	0.5
	Postman installieren / einrichten / Requests erstellen	1
	Aufsetzen Travelblog-Server mit node.js und express	1.5
	Travelblog Repository implementieren / anpassen	2.5
	MongoDB im Travelblog-Server verwenden	1
	Environment-File aufsetzen	1
28.02.2021	Routing und grundlegende Navigation	2.5
	Model für Travelblog und BlogEntry	0.5
	Page: Travelblogs mit Datagrid & Travelblog-Service: getAll()	5
	Page: Detail-Item, Welcome, NotFound	1.5
	Problembehebung Angular-Materials	2.5
03.03.2021	Code Cleanup	0.5
	Travelblog-Server: POST, PUT, DELETE überarbeitet	4.5
07.03.2021	DELETE Request verbessert	1.5
	GUI Anpassungen für ein-/ausgelogged und allgemeine Anpassungen	4.5
	Allgemeines Review / Fixen (GUI, DB-Connection Aufbau etc.)	3
09.03.2021	Dokumentation / Arbeitsjournal	3
	Artefakte zur Abgabe vorbereiten	1
Total		**40**