Praxisaufgabe

Dokumentation des Projekts TravelPoster für das Wahlfach Microservices

von

Felix Waldbach & Marius Kiessling

Abgabedatum:	11. Juni 2018
Matrikelnummer, Kurs:	8215018, /*1234567*/, TINF16B
Dozent:	Dr. Ingolf Buttig

Abkürzungsverzeichnis

MVC Model-View-Controller

UI User Interface

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Struktur der Arbeit

Kapitel 2

Aufgabenstellung

Kapitel 3

Architektur und Grundlagen

3.1 Projektaufbau

3.2 MongoDB

MongoDB ist ein dokument-orientiertes, Open-Source Datenbank-Programm. Dabei speichert MongoDB Daten in JSON-Dokumenten, sodass Datenstrukturen sich von Zeit zu Zeit verändern können und von Dokument zu Dokument unterschiedlich sind. Ein JSON-Dokument sieht beispielsweise so aus:

```
{
ebene11: "Dieser Text befindet sich auf oberster Ebene",
ebene12: {
            ebene21: "Dieser Text befindet sich eine Ebene weiter unten"
          },
ebene13: "Dieses Dokument kann beliebig viele Elemente enthalten und das auch
           von anderen Datentypen"
}
```

Ein wichtiges Feature ermöglicht es, Attribute ineinander zu verschachteln. Im Idealfall können die Dokumente also stets auf den Anwendungscode angepasst werden. Mithilfe von Queries werden diese Daten erreicht und verändert. Dieses Datenbank-Schema zeichnet sich also vor allem durch Flexibilität aus und dadurch, dass es sehr einfach zu lernen ist. Wird MongoDB beispielsweise in einem NodeJS-Projekt verwendet, sieht eine beispielhafte Query so aus:

```
db.collection("testCollection").findOne({id: 0}, function(err, res) {
if(err) throw err;
else console.log("Query erfolgreich ausgefuehrt");
});
```

Hierbei muss jedoch zunächst überhaupt eine Verbindung zu der Datenbank hergestellt werden:

```
const mongoClient = require('mongodb').MongoClient;
const mongoURL = "mongodb://localhost:27017";

mongoClient.connect(mongoURL, function (err, mgo) {
    if (err) callback(err, mgo);
    // Query, um die DB zu durchsuchen oder zu manipulieren
});
```

Grundvoraussetzung ist, dass diese Datenbank und die sogenannte Collection, vergleichbar mit einer Relation bei einem relationalen Datenbankmodell, überhaupt existiert. Abschliessend muss die Verbindung mit der Datenbank wieder beendet werden:

3.3 Express

Bei Express handelt es sich um ein serverseitiges Web-Application-Framework, welches oft als Standard-Server-Framework für Node.js bezeichnet wird. Es enthält eine Menge Features und erleichtert so das schnelle Aufsetzen eines Node.js-Servers und ermöglicht es, diesen Server schnell um Features zu erweitern. So ermöglicht der „Express-Router“ beispielsweise das Definieren von REST-Routen, auf die mithilfe von GET-, POST, PUT und DELETE-Methoden zugegriffen werden kann. Mithilfe des Express-Routers kann das Verhalten definiert werden, das abgearbeitet werden soll, wenn auf eine der Routen zugegriffen wird. Inzwischen bauen viele Node.js-Framework auf Express auf, da es sich für Single-Page, Multi-Page sowie für mobile und webbasierte Applikationen eignet. So gibt es beispielsweise den häufig verwendeten MERN-Stack, bestehend aus MongoDB, Express, React sowie Node.js, wie er auch in diesem Projekt verwendet wird. Weitere Beispiele sind KeystoneJS oder Kraken. Dank Express und Node.js wird Javascript mittlerweile nicht mehr nur für Frontend-Entwicklung von Webseiten verwendet, sondern es werden

auch immer häufiger serverseitige Komponenten mithilfe von Javascript realisiert. Es unterstützt außerdem das MVC-Pattern, um Applikationen nach einem bestimmten Schema zu modellieren und ist betriebssystem unabhängig.

3.4 React

React ist eine Javascript-Bibliothek für das Entwickeln von interaktiven Benutzeroberflächen. Mithilfe von View-Komponenten, die jeweils eine render()-Methode enthalten, werden einzelne Teile dieser React-UI realisiert. React kümmert sich vor allem um effizientes Rendering und das Updaten der Webanwendung. Die render()-Methode verarbeitet Input-Daten und gibt den HTML-Code zurück, der letztendlich angezeigt, also gerendert, wird. Eine hilfreiche Funktionalität von React ist der sogenannte State. Über diesen werden Daten verarbeitet und bei jeglicher Änderung der State-Variable wird die Seite neugeladen, die render()-Methode also erneut ausgeführt. Die state-Variable wird beispielsweise so verwendet:

```
// im constructor der React-Komponente
this.state = {element1: "Dieses Element ist Teil des states"};

//Zugreifen auf Elemente im state
console.log(this.state.element1);
```

3.5 Docker

Docker ermöglicht die *containerization* von Anwendungen. Dies bedeutet, dass Anwendung nicht mehr direkt auf dem Host-Betriebssystem des Servers ausgeführt werden, sondern in einer vom Host-System getrennten Ebene ausgeführt werden. Der große Vorteil bei der Verwendung von Docker liegt in der Systemsicherheit und Geschwindigkeit. Docker Container sind, falls nicht gewollt, komplett voneinander getrennt und unabhängig. Es ist somit der einen Systemkomponente nicht ohne explizite Deklaration möglich auf andere Anwendungskomponenten zuzugreifen. Außerdem handelt es sich bei Docker um keine Virtualisierungslösung. Es muss somit nicht pro Container ein eigener Kernel bereitgestellt werden. Auf Grund dessen ist die Verwendung von Docker um einiges performanter als die Verwendung von vollen Virtualisierungslösungen wie *HyperV*.

Kapitel 4

Umsetzung

4.1 Grundlegendes

Die grundlegende Architektur gliedert sich in das Frontend, Backend und Datenbank. Dabei besteht das Backend aus einem NGINX Webserver, der alle Anfragen an die Node.JS Anwendung weiterleitet. Er agiert somit als Proxy. Eine Ausnahme für die Weiterleitung an die Backend-Anwendung besteht in der Abhandlung der Anfragen an *http://host.name/storage/**. Alle Anfragen, die an diesen Pfad werden von NGINX direkt an das Dateisystem weitergeleitet. So können die großen Geschwindigkeitsvorteile von NGINX in der Austeilung von statischen Ressourcen verwendet werden.

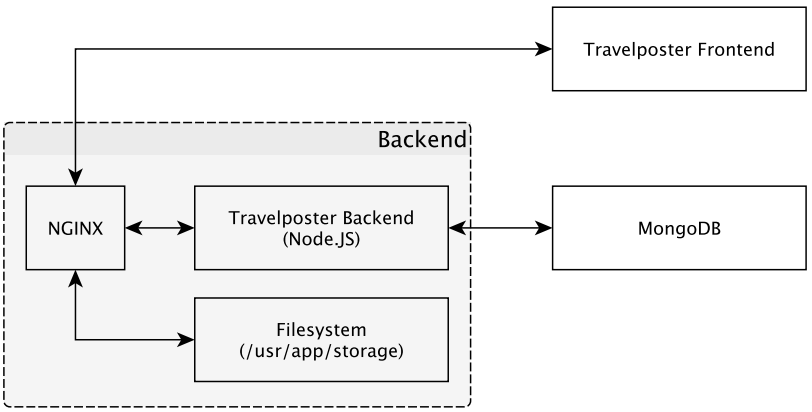

Abbildung 4.1: Architektur

Die Datenbank kommuniziert einzig und allein mit der Node.JS Anwendung. Dies bedeutet, dass jegliche Datenanfragen, die vom Frontend gestellt werden, erst vom NGINX-Server an die Node.JS Anwendung weitergegeben und von dort mit Hilfe der Datenbank beantwortet werden. Eine direkte Kommunikation ist aus Sicherheitsgründen nicht direkt

zwischen dem Frontend und der Datenbank möglich.

4.2 Frontend

4.2.1 Bildupload

4.2.2 Senden der Informationen an das Backend

4.2.3 Bildanzeige

Jedes Mal, wenn die Seite aktualisiert wird, sollen anhand der aktuellen Informationen in der Datenbank alle Posts angezeigt werden. Hierfür wird ein GET-Request an die Route */post/get/all* an das Backend geschickt. Das Resultat enthält den Namen des Verfassers sowie das dazugehörige Bild in einem JSON-Objekt.

4.3 Backend

4.3.1 Speichern der Posts

Die Posts eines Benutzers werden mithilfe von cors()-Headern vom separaten Frontend an das Backend gesendet. Diese bestehen im Grunde lediglich aus dem Namen des Verfassers des Posts und dem Namen der Datei, in der das zugehörige Bild gespeichert wird. Empfängt das Backend einen POST-Request an die Route /post/fileupload", wird der mitgesendete Post in der dahinterliegenden MongoDB-Collection gespeichert. Zusätzlich wird das Bild, welches im base64-Format gesendet wird, in PNG-Format umgewandelt und in einer Datei gespeichert. Der Name dieser Datei wird letztendlich in der Datenbank zusammen mit dem Namen des Verfassers gespeichert. Um die Einträge in der Collection zu speichern, wird eine Query verwendet, die der Collection einen neuen Eintrag mit einem "name und einem "filenameAttribut hinzufügt.

4.3.2 Senden der Informationan an das Frontend

Empfängt das Backend einen GET-Request an die Route */post/get/all*, soll es alle bisher in der Collection gespeicherten Posts im JSON-Format an das Frontend zurückschicken. Hierfür wird eine Query verwendet, die die "postCollection ohne jegliche Bedingung durchsucht, sodass alle Einträge gefunden werden:

```
db.collection("post").find({}, function(err, res) {
    if(err) throw err;
    else console.log("Query erfolgreich ausgefuehrt");
});
```

4.4 Deployment

Die Bereitstellung des Projektes kann komplett mit Hilfe von Docker bewältigt werden. Dazu steht eine *docker-compose.yml* bereit. Jene besitzt folgende Struktur:

```
version: "3"
services:
    frontend:
        build: https://github.com/felixwaldbach/travelposterfrontend.git
        ports:
            - "8000:80"
    backend:
        build: https://github.com/felixwaldbach/travelposter.git
        ports:
            - "5000:80"
    db:
        image: mongo
        entrypoint: mongod --bind_ip_all
        expose:
            - "27017"
```

Zur Bereitstellung des **Frontends** wird ein eigen angepasstes Image verwendet. Dies gilt ebenfalls für die Bereitstellung des **Backends**. Beide Images werden beim ersten Start der docker-compose.yml gebaut. Das **Datenbank**-Image hingegeben wird nicht angepasst. MongoDB stellt bereits ein fertiges Image bereit, welches allen Anforderungen dieser Anwendung genügt. Es wird im Datenbank-Image ebenfalls kein permanentes Volumen eingebunden. Da es sich hier nur um eine Anwendung handelt, die die Möglichkeiten von Node.JS und Docker aufzeigen soll, wurde die Entscheidung getroffen, dass ein permanenter Datenspeicher nicht relevant sei.

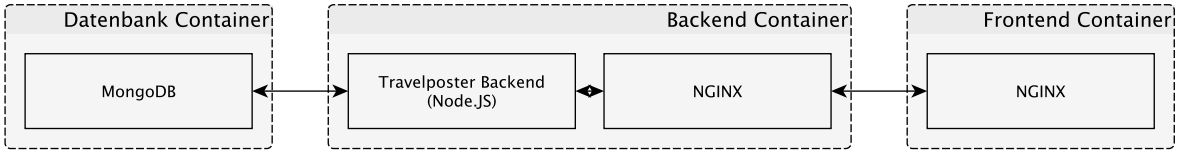

Abbildung 4.2: Docker Aufbau

Der Start aller Anwendungskomponenten kann mit folgendem Befehl ausgeführt werden:

```
docker-compose up
```

Eine Internetverbindung ist zu diesem Schritt notwendig, da der Code für das Backend und Frontend von GitHub und das Image für den MongoDB Container vom offiziellen Docker Hub geladen werden müssen.

Kapitel 5

Fazit

5.1 Erweiterungsmöglichkeiten

Dieses Projekt bietet eine gute Basis für zukünftige, umfangreichere Projekte. Es könnte gar zu einer umfangreichen Blog-Webseite erweitert werden. Hierfür könnte beispielsweise ein User-Authentifizierungssystem implementiert werden. Da MongoDB bereits zum Speichern der Posts verwendet wird, könnte die zugrundeliegende Datenbank einfach um eine User-Collection erweitert werden. Im Zuge dieser Erweiterung würde es sich weiterhin eignen, durch Session-Handling und Cookies das Benutzererlebnis zu verbessern, sodass man sich beispielsweise nur einmalig anmelden muss und dann jederzeit seine eigenen Posts ansehen kann. Weiterhin könnte ein Follower-System implementiert werden, bei dem ein Benutzer in der Lage ist, anderen Benutzern zu folgen und so auch die Posts eben dieser anderen Benutzer anzusehen. Ein Benutzer sieht also schlussendlich nur die Posts von anderen Benutzern, die ihn interessieren. Des Weiteren können die Posts umfangreicher und flexibler gestaltet werden, beispielsweise durch Link-Unterstützung, dem Hinzufügen einer aussagekräftigen Beschreibung oder auch dem Posten von Videos. Travelposter zeichnet sich also in der Umsetzung besonders durch flexible Erweiterbarkeit aus.

Literaturverzeichnis

[1]	MongoDB	https://www.mongodb.com/what-is-mongodb MongoDB, Inc. Stand 03.06.2018
[2]	Frameworks aufbauend auf Express	https://expressjs.com/en/resources/frameworks.html expressjs contributors Stand 03.06.2018
[2]	Express.js	https://www.upwork.com/hiring/development/express-js-a-server-side-javascript-framework/ Upwork Stand 03.06.2018
[2]	ReactJS	https://reactjs.org/ Upwork Stand 03.06.2018