Technische Dokumentation zur erstellten Anwendung, Modul: Internetanwendungen für mobile Geräte, Sommersemester 2015

Andreas Willems

4. Juli 2015

Die erstellte Anwendung ermöglicht das Abrufen von Fotos aus einer Fotosession über ein mobiles Endgerät (z.B. Smartphone) über das Internet. Hierzu wurden ein Web-Client, ein Web-Server sowie ein Datenserver erstellt.

Inhaltsverzeichnis

1 Allgemeine Funktionsweise

Der Kunde eines Fotografen bekommt nach einer Fotositzung (Session) eine Karte mit einer Session-ID überreicht. Anhand dieser ID, welche sich in der aufgerufenen Anwendung eingeben lässt, werden die Bilder aus der Session beim Kunden im Endgerät (Desktop-Browser oder Smartphone-Browser) dargestellt.

Der generelle Aufbau lässt sich der folgenden Abbildung 1 entnehmen.

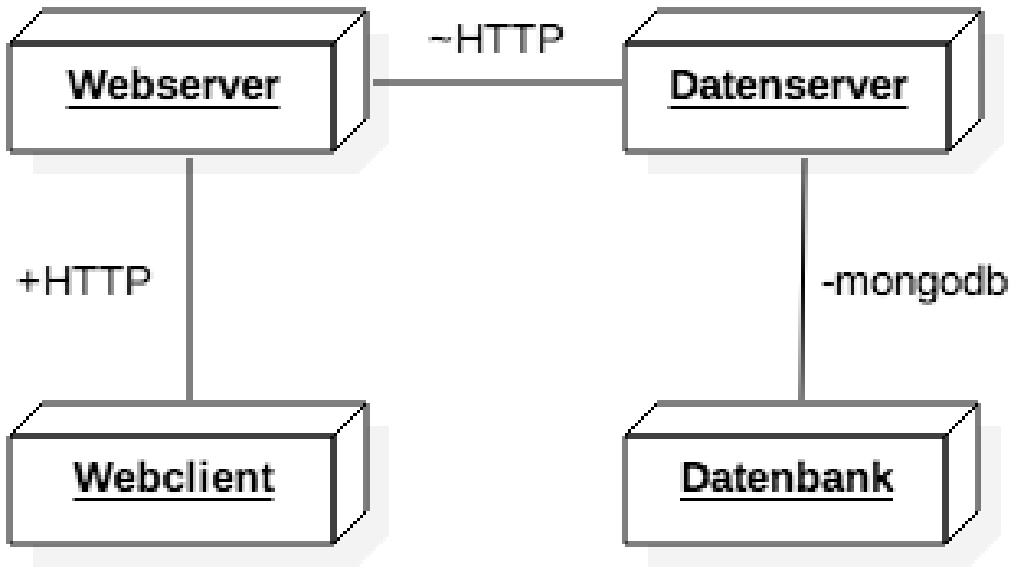

Abbildung 1: Genereller Aufbau der Anwendung

Wie in der Grafik zu sehen ist, besteht die Anwendung aus vier den Komponenten

- **Webclient**: diese Komponente ist für den Nutzer sichtbar und dient der Interaktion mit der Anwendung
- **Webserver**: der Webserver nimmt Anfragen des Clients entgegen, beantwortet sie entweder selber oder leitet sie an den Datenserver weiter und gibt Antworten des Datenservers an den Webclient weiter
- **Datenserver**: der Datenserver nimmt Anfragen des Webservers entgegen und dient der Interaktion mit der Datenbank
- **Datenbank**: die Datenbank dient der persistenten Haltung von Sitzungsdaten und Fotos

Im Folgenden werden die einzelnen Komponenten weiter vorgestellt und ihre Funktionsweise erläutert.

2 Webclient

3 Webserver (AW)

Der Webserver nimmt Anfragen des Clients über HTTP entgegen. Er liefert zum einen die Internetseite an den Client aus, über die der Benutzer mit der Anwendung interagiert und kommuniziert zum anderen mit dem Datenserver.

Der typische Ablauf der Kommunikation zwischen Webclient und Webserver ist in der folgenden Abbildung 2 dargestellt:

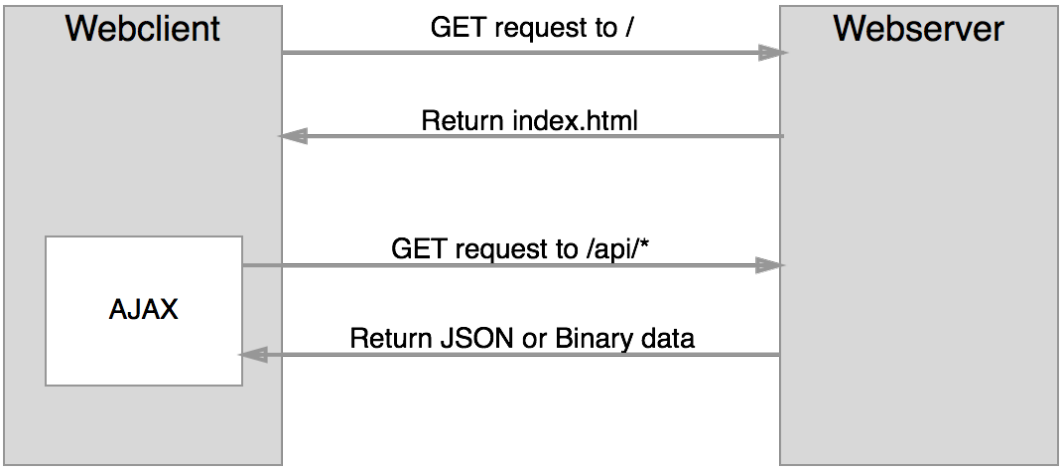

Abbildung 2: Kommunikation zwischen Webclient und Webserver

Die Abbildung zeigt, wie der Webserver eine Anfrage an den Webserver stellt und der mit der Rückgabe der Datei *index.html* antwortet. Da es sich um eine Single-Page-Anwendung handelt, liefert der Webserver nur dieses eine Seite / Page aus.

In der Abbildung 2 ist weiter vereinfacht dargestellt, wie die weitere Kommunikation zwischen Webclient und Webserver über die im Webclient integrierte AJAX[1]-Engine ausgeführt wird.

[1] AJAX (Asynchronous JavaScript and XML): Konzept zur asynchronen Datenübertragung

3.1 Schnittstellen

Der Webserver hält die in der folgende Tabelle 1 dargestellten Schnittstelle bereit, an die der Webclient Anfragen richten kann:

Methode	**Pfad**	**Funktion**
GET	/	ruft die Datei index.html auf und bewirkt den Start der Anwendung
GET	/api/sessions/:id	ruft die Session mit gegebenen ID auf
GET	/api/thumbnails/:id	ruft ein Thumbnail des Bildes mit gegebenen ID ab
GET	/api/images/:id	ruft ein Bild mit gegebenen ID ab

Tabelle 1: Die zu realisierende Schnittstelle

Die in Tabelle angegeben Endpunkte werden wie folgt verwendet:

Der Pfad „/ “ wird durch den Browser des Clients über die Eingabe des Domainnamens aufgerufen.

Der Pfad „/api/sessions/:id “ wird über einen AJAX-Request aufgerufen, wenn auf der Startseite eine Session-ID eingeben wird und der Benutzer den Knopf „Senden “ drückt.

Die Pfade „/api/thumbnails/:id “ und „/api/images/:id “ werden in der Datei index.html als Atrribute des Tags <*img*> verwendet.

3.2 Realisierung

Der Webserver ist in JavaScript auf der Node.js-Plattform implementiert. Zur vereinfachten Installation und bequemeren Handhabung von Anfragen an den Server wird das Framework *Express.js* verwendet.

Die Darstellung der Ordnerstruktur enthält folgende Abbildung 3:

Die Datei *package.json* enthält die Konfiguration einer Node.js-Anwendung und definiert unter anderem Abhängigkeiten zu Paketen.

Die Datei *config.js* enthält die URL und Ports der beteiligten Server.

In der Datei *app.js* sind die Befehle zur Erzeugung des Web-Servers enthalten . Weiter werden in dieser Datei die übrigen Dateien zur Konfiguration der Datenbank und der Routen importiert. In Listing 1 ist die Datei auszugsweise vorgestellt.

```
1 var express = require("express"); // express importieren
2 var app     = express(); // Anwendung / Web-Server initialisieren
3 var indexRoutes = require("'./routes/index"); // Routen importieren
4 var apiRoutes = require("./routes/apiRoutes"); // Routen importieren
5
```

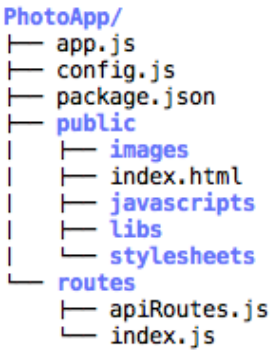

Abbildung 3: Ordnerstruktur des Webservers

```
6  app.use("/", routes); // Routen verwenden für Anfragen an den Server
7  app.listen(60127); // Web-Server starten, auf Anfragen an Port 60127 hören
```

Listing 1: Auszug aus app.js

Das Listing zeigt, wie zunächst die verwendeten Module mit *require* importiert werden und die Anwendung anschließend Anwendung initialisiert wird. Im Anschluss werden die importierten Routen in die Anwendung integriert und zum Schluss der Server gestartet. Der Server hört auf Anfragen an den definierten Port.

Die einzelnen Endpunkte der Schnittstelle sind wie in Listing 2 implementiert:

```
1  /* GET home page. */
2  router.get("/", function(req, res, next) {
3      res.sendFile('index.html');
4  });
5
6  /* GET requests to /api/sessions/:id, /api/thumbnails/:id, /api/images/:id */
7  router.get("/sessions/:id", function(req, res, next) {
8      var sessionId = req.params.id;
9      // create options object for the following request
10     var options = {
11         host: serverData.host,
12         port: serverData.port,
13         path: "/api/sessions/" + sessionId,
14         method: "GET",
15         accept: "application/json"
16     };
17
18     // create and send request
19     http.request(options, function(response) {
20     // check response for error
21         if (response.statusCode == "404") {
22             return res.status(404).send();
23         }
24         // in case of images pipe incoming stream to outgoing stream
25         // response.pipe(res);
26
```

```
27          var resData = "";
28          // react to the server's response...
29          response.on("data", function(data) {
30          // ... by passing the responded data to variable resData...
31              resData += data;
32          });
33
34          // ...and returning it to the client in the end
35          response.on("end", function() {
36              // ...as JSON if dealing with session data...
37              res.json(JSON.parse(resData));
38              // ...or as binary data in case of images
39              // res.set("Content-Type", contentType);
40              // res.end(resData, "binary");
41          });
42      }).end();
43  });
```

Listing 2: Auszug aus den Dateien routes/index.js und routes/apiRoutes.js

Die erste Route in den Zeilen 2-4 reagiert auf die Anfrage an den Pfad „/ “. Sie sendet daraufhin die HTML-Datei an den Client zurück.

Die zweite Route in den Zeilen 7 bis 40 ist verantwortlich für Anfragen an den Pfad „/api/sessions/:id “, die Funktionsweise ist jedoch identisch zur Anfragen an „/api/images/:id “ und „/api/thumbnails/:id grqq.

Wenn eine Anfrage an diese Pfade eingeht, wird eine entsprechende HTTP-Anfrage an den Datenserver gerichtet und die zurückerhaltene Antwort an den Webclient weitergeleitet.

Zunächst wird aus den Request-Parametern die ID ausgelesen (Zeile 8). Anschließend wird ein Objekt erzeugt, das die Art der ausgehenden HTTP steuert (Zeilen 10-16). Der HTTP Request wird mit den gegebenen Optionen an den Datenserver gesandt (Zeile 19 bis 42). Die zurückkommende Antwort wird verarbeitet und an den Webclient weitergeleitet (Zeilen 21 bis 40).

4 Datenserver

Der Datenserver nimmt über eine Schnittstelle im REST-Design[2] Anfragen über HTTP[3] entgegen und wandelt diese in Datenbankanfragen um. Nach erfolgter Antwort gibt der Datenserver die Daten aus der Datenbank an den Client über HTTP zurück.

[2]REST (Representational State Transfer): Programmierparadigma, um Ressourcen im Web über definierte und eindeutige Schnittstellen zu erreichen

[3]HTTP (Hyper Text Transfer Prototcol)

5 Datenbank