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

Christian Halfmann (CH)
Andreas Willems (AW)

7. 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 (AW/CH) 2

2 Webclient (CH) 3

3 Webserver (AW) 4
3.1 Schnittstellen 5
3.2 Realisierung 5

4 Datenserver (AW) 9
4.1 Schnittstellen 9
4.2 Realisierung 10

5 Datenbank (AW) 12

6 Lessons learned 12

1 Allgemeine Funktionsweise (AW/CH)

Die FotoApp wurde in Zusammenarbeit mit einer Fotografin erarbeitet und entwickelt. Die Fotografin ist spezialisiert auf Familien Fotografie und möchte den fotografierten Familien zukünftig auch einen Online-Service bieten.

Da Nutzer mobiler Endgeräte ihre Smartphones oder Tablets vermehrt auch zum Speichern und Betrachten ihrer eigenen Fotos und Bildergalerien nutzen, sollen – als zusätzlicher Service für die Kunden der Fotografin – die Bilder einer Foto Session zukünftig auch Online, speziell für den Abruf über mobile Endgeräte zur Verfügung gestellt werden.

Die Kunden bekommen nach einer Fotosession eine individuelle Session-ID ausgehändigt. Mit dieser ID haben sie innerhalb der Anwendung Zugang auf die Fotos ihrer Fotosession und können diese online (über einen Desktop-Browser oder einen Mobile-Browser) betrachten.

Die folgende Abbildung 1 zeigt den generellen Aufbau der Anwendung.

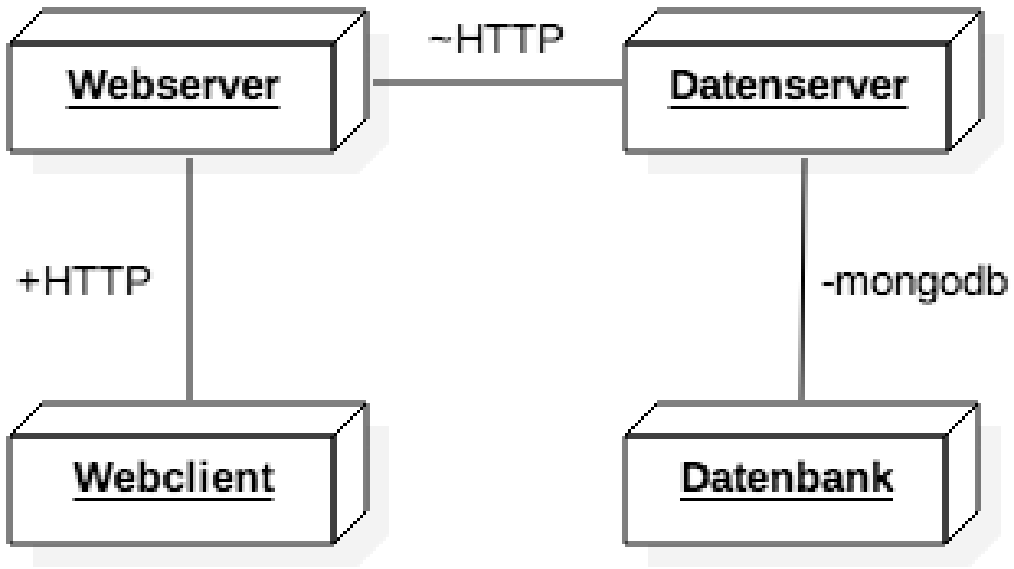

Abbildung 1: Genereller Aufbau der Anwendung

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 im Detail vorgestellt und ihre Funktionsweise erläutert.

2 Webclient (CH)

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:

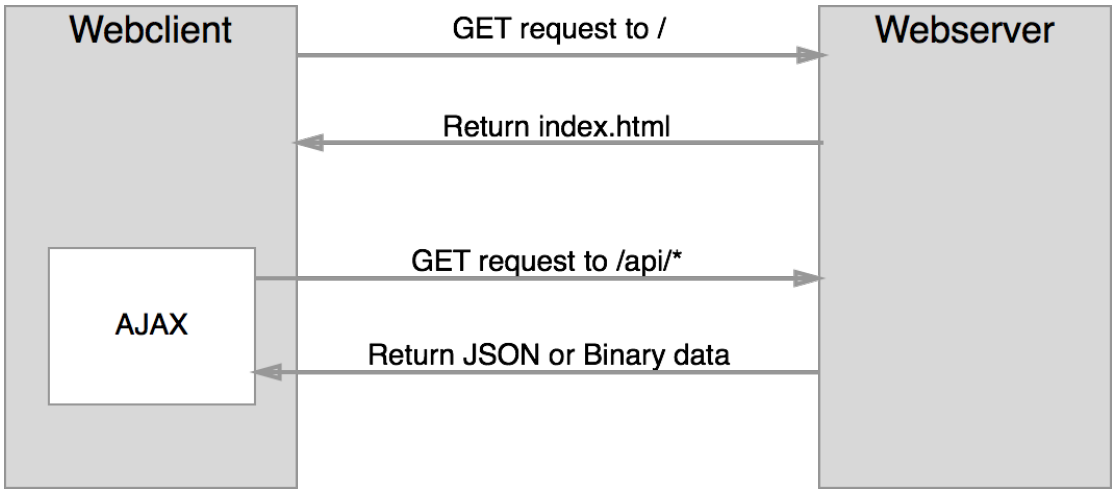

Abbildung 2: Kommunikation zwischen Webclient und Webserver

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 Attribute 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:

Abbildung 3: Ordnerstruktur des Webservers

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 und die Routen importiert. In Listing 1 ist die Datei auszugsweise vorgestellt.

```
1 var express = require("express"); // import express.js module
2 var app     = express(); // initialize app / server
3 var indexRoutes = require("'./routes/index"); // import routes
4 var apiRoutes = require("./routes/apiRoutes"); // import more routes
5
6 app.use("/", routes); // use routes for requests to server
7 app.listen(60127); // start server listening on port 60127
```

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 (Routen) sind wie in Listing 2 abgebildet implementiert:

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".

```
/* GET home page. */
router.get("/", function(req, res, next) {
    res.sendFile('index.html');
});

/* GET requests to /api/sessions/:id, /api/thumbnails/:id, /api/images/:id */
router.get("/sessions/:id", function(req, res, next) {
    var sessionId = req.params.id;
    // create options object for the following request
    var options = {
        host: serverData.host,
        port: serverData.port,
        path: "/api/sessions/" + sessionId,
        method: "GET",
        accept: "application/json"
    };

    // create and send request
    http.request(options, function(response) {
    // check response for error
        if (response.statusCode == "404") {
            return res.status(404).send();
        }
        // in case of images pipe incoming stream to outgoing stream
        // response.pipe(res);

        var resData = "";
        // react to the server's response...
        response.on("data", function(data) {
        // ... by passing the responded data to variable resData...
            resData += data;
        });

        // ...and returning it to the client in the end
        response.on("end", function() {
            // ...as JSON if dealing with session data...
            res.json(JSON.parse(resData));
            // ...or as binary data in case of images
            // res.set("Content-Type", contentType);
            // res.end(resData, "binary");
        });
    }).end();
});
```

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

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).

Die Anwendung enthält im Übrigen den Ordner *public*, der die Dateien des Webclients enthält (vgl. Abschnitt 2).

4 Datenserver (AW)

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.

Der Ablauf der beschriebenen Kommunikation ist in Abbildung 4 ersichtlich:

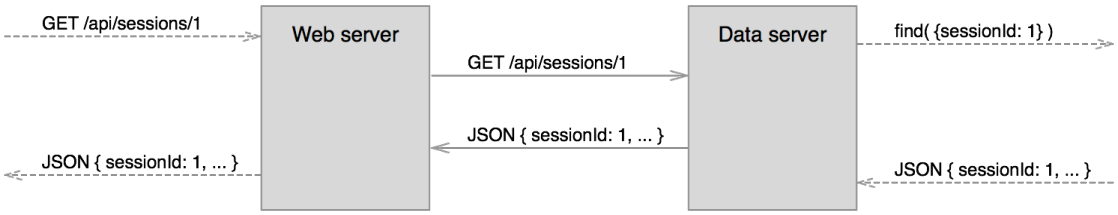

Abbildung 4: Kommunikation zwischen Webserver und Datenserver

Die Abbildung zeigt, wie am Webserver eingehende Requests an den Datenserver weitergeleitet werden. Der Datenserver erstellt daraufhin eine Datenbankanfrage und verarbeitet die erhaltene Antwort. Diese wird an den Webserver gesendet, der sie an den Webclient weiterleitet.

4.1 Schnittstellen

Die Endpunkte der Schnittstelle, die der Datenserver zur Kommunikation anbietet, sind der folgenden Tabelle 2 zu entnehmen:

Methode	**Pfad**	**Funktion**
GET	/api/sessions/:id	ruft die Daten der Session mit der 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 2: Die zu realisierende Schnittstelle

Die angegebenen Routen entsprechen denen des Webservers und dienen der Übermittlung der in der Datenbank hinterlegten Daten an den Webserver.

Der Datenserver hält noch weitere Schnittstellen bereit, um die Daten in die Datenbank einzutragen und zu aktualisieren. Da das Einpflegen neuer Daten auch mit anderen

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

Anwendungen und losgelöst von der hier vorgestellten Anwendung erfolgen kann, wird hier des Umfangs wegen auf die Darstellung verzichtet.

4.2 Realisierung

Die Struktur der Anwendung wird in Abbildung 5 ersichtlich:

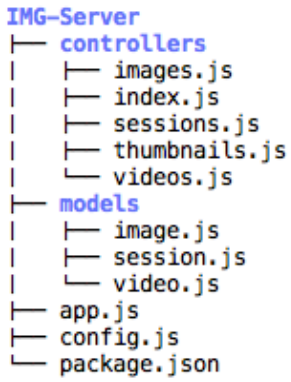

Abbildung 5: Ordnerstruktur Datenserver

Der Aufbau des Datenserver entspricht größtenteils dem in Abschnitt 3 beschriebenen Aufbau des Webservers.

Die Dateien *app.js*, *config.js* und *package.json* erfüllen den gleichen Zweck wie auch beim Webserver.

Der Ordner *controllers* enthält die Routen für die einzelnen Endpunkte der Schnittstelle.

Neu im Vergleich zum Webserver ist hier der Ordner *models*, der Abstraktionen für den Zugriff auf die Datenbank bereithält.

Im folgenden Listing 3 ist beispielhaft die Reaktion auf eine Anfrage nach einer bestimmten Session dargestellt:

```
1   (...)
2   /*
3   * GET request to /api/sessions/:id.
4   * Returns the session with the requested id.
5   */
6   router.get('/:id', function(req, res) {
7       // get id from request parameters
8       var id = req.params.sessionId;
9       // use Session model to make db query
10      Session.getOne(parseInt(id), function(result) {
11          // if a result is found, send back the result
12          if (result) {
13              res.json(result);
14          // otherwise send back a 404 status
15          } else {
16              res.status(404).send();
17          }
18      });
19  });
20
21  (...)
22
23  /*
24  * Returns the session with the given id.
25  * @param id - the id you're looking for
26  * @param cb - a callback function
27  * @returns a session object
28  */
29  Session.getOne = function(id, cb) {
30      mongoCrud.read({sessionId: id}, COLLECTIONNAME, function(doc) {
31          cb(doc);
32      });
33  };
```

Listing 3: Auszug aus den Dateien controllers/sessions.js und models/session.js

In den Zeilen 6 - 19 ist der Controller für den Pfad "/api/sessions/:id" zu sehen. Dieser entnimmt den Request-Parametern die zu suchende ID und nutzt das Session-Model, um eine Datenbankanfrage durchzuführen. Eine gefundene Session wird an den Client (in diesem Fall der Webserver) zurückgegeben. Wird keine Session gefunden, wird der Statuscode 404 ("nicht gefunden") zurückgesendet.

Die Zeilen 23 - 33 enthalten die Definition der Methode *getOne* auf dem Modell *Session*. Darin wird mithilfe des Moduls *mongo-crud-layer* (hier: mongoCrud) eine Suchanfrage an die Datenbank gerichtet. Das Resultat dieser Abfrage, die Session-Daten oder ein Fehler, werden in übergebenen Callback-Methode verarbeitet.

5 Datenbank (AW)

Als Datenbank kommt eine Instanz von MongoDB zum Einsatz. Die Datenbank ist für Anwendung über die URI "mongodb://localhost:27017/img-server"zu erreichen.

Die Kommunikation zwischen Datenserver und Datenbank ist in Abbildung 6 dargestellt.

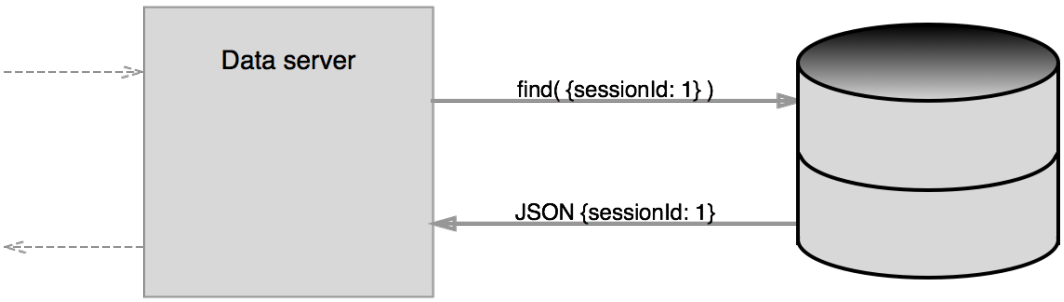

Abbildung 6: Kommunikation zwischen Datenserver und Datenbank

Der Datenserver richtet Anfragen unter Verwendung des Moduls *mongo-crud-layer* an die Instanz von MongoDB. Da die Verkehrssprache von MongoDB JavaScript ist, können Anfragen aus der Datenserver-Anwendung heraus ohne eine weitere erstellt werden.

Das Modul *mongo-crud-layer* stellt eine Abstraktion von CRUD-Operationen[4] zur Verfügung.

6 Lessons learned

Im Folgenden werden einige Punkte genannt, die während dir Realisierung Probleme bereitet haben:

- Die Speicherung von Binärdaten in MongoDB ist kompliziert.
- Die Verwendung der vorliegenden Architektur mit der ausgesuchten Software ist bereits bei einer geringen Anzahl an (größeren) Bildern nicht performant.
- Die korrekte Anwendung von HTTP und die Manipulation von Headern ist zu beachten.

[4]Create, Read, Update, Delete