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)

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.

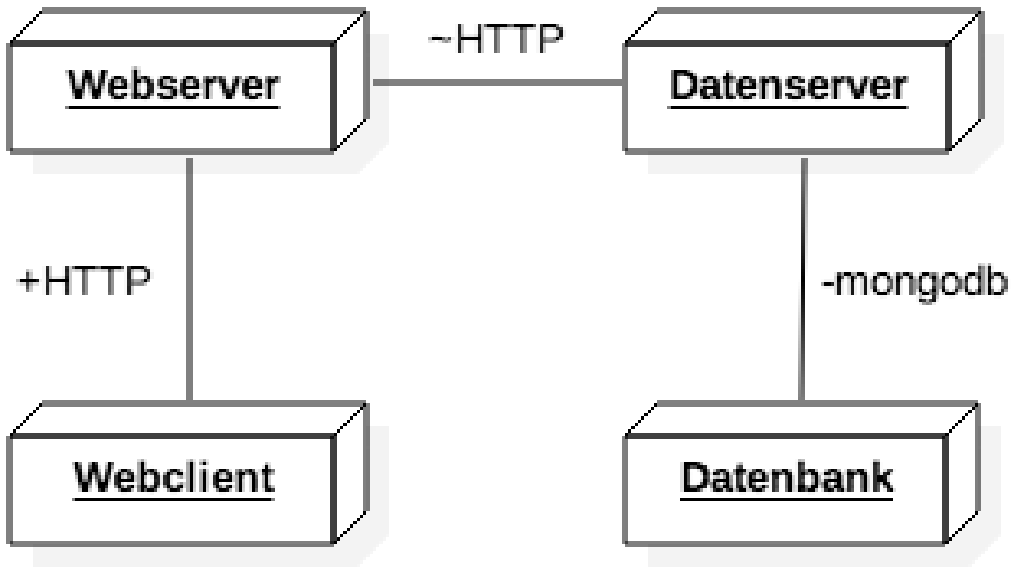

Abbildung 1: Genereller Aufbau der Anwendung

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)

Der Webclient ist die Schnittstelle zwischen Nutzer und Anwendung. Nach Aufruf der Applikation wird der Nutzer aufgefordert seine Session-ID einzugeben. Ist diese gültig, gelangt er in seinen persönlichen Bereich. Dort wird er mit seinem Namen begrüßt und kann die Bilder aus seiner Fotosession betrachten.

Die einzelnen Bilder werden zunächst zur Übersicht in einer Galerie mit quadratischen Thumbnails angezeigt. Durch Klick auf ein Thumbnail wird das entsprechende Bild vergrößert bzw. das Orginal-Bild angezeigt. Hierbei bietet die Galerie vier verschiedene Ansichtsmöglichkeiten. Auf Smartphones fällt die Auswahlmöglichkeit weg und die Bilder werden initial in der Borderless / Fullscreen Ansicht Dargestellt.

- Lightbox
- Borderless
- Lightbox / Fullscreen
- Borderless / Fullscreen

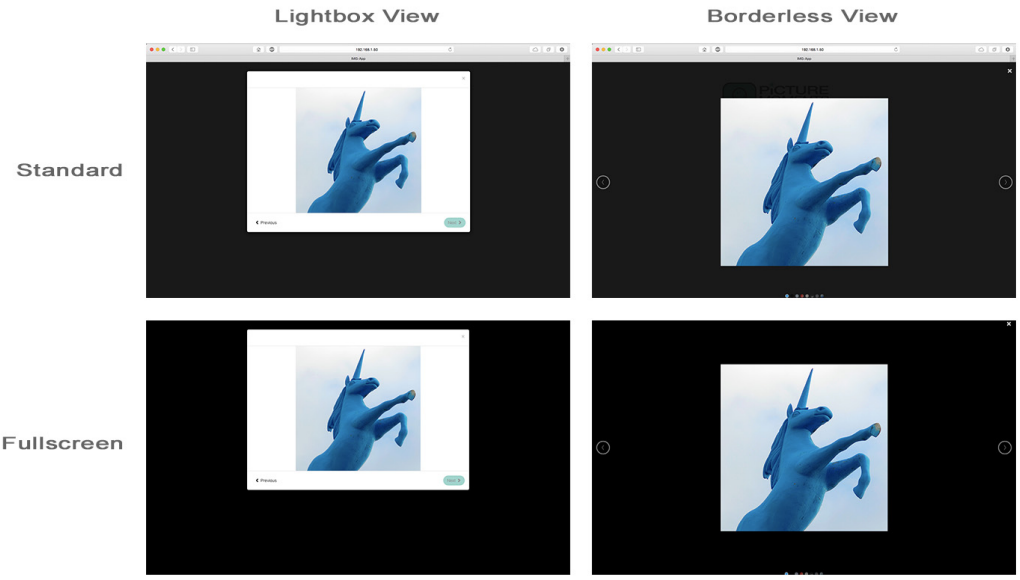

Abbildung 2: Galerie Ansichten

In der Galerieansicht lässt sich jeweils zu dem nächsten oder vorherigen Bild via Button oder Pfeiltasten in Desktop Borwsern oder via Touch-Slide auf mobilen Endgeräten navigieren.

2.1 Schnittstellen

Der Webclient hält folgende Schnittstellen bereit:

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

Mit der Eingabe der URL wird zunächst ein GET Request an den Webserver gestellt, der Webserver gib daraufhin die *index.html* zurück.

Die weitere Kommunikation zwischen Client und Webserver verläuft über die im Webclient integrierte AJAX-Engine. Der Client stellt einen GET Request und bekommt die entsprechende Antwort in Form eines JSON Strings oder als Binärdaten zurück (Siehe dazu auch Abschnitt 3.1 und Abbildung 3)

2.2 Realisierung

Der Webclient basiert auf einer Single-Page Anwendung und wurde mittels der Grundelemente HTML (*index.html*), CSS (*style.css*), JavaScript (*app.js*) implementiert.

Neben diesen drei Elementen wurde zur Realisierung der Website das Framework Bootstrap und die JavaScript Bibliothek jQuery genutzt. Außerdem wurde zur Erstellung der Galerie auf die Bootstarp Image Gallery blueimp zurückgegriffen.

Entsprechende Codezeilen des Bootstrap Stylesheets werden durch das Stylesheet *style.css*, was individuelle Designvorgaben enthält, überschrieben.

Die *index.html* gibt die Struktur der Web-Anwendung vor. Sie beinhaltet das Logo, ein Jumbotron als Textfeld, das Formular zur Eingabe der Session ID und die entsprechenden Button. Die Darstellung der Seite wird durch die *app.js* gesteuert. Die Elemente werden je nach Bedarf ein- oder ausgeblendet. Dialogtexte sowie die Galerie werden dynamisch erzeugt aus der *app.js* erzeugt.

Des Weiteren stellt die *app.js* die Funktion requestSession zur AJAX Kommunikation zwischen Webclient und Webserver bereit.

```
1     function requestSession(evt) {
2         var id = $('#inputSessionId').val();
3         if (id !== '') {
4             var jqxhr = $.ajax('/api/sessions/' + id)
```

```
5               .done(function() {})
6               .success(function(data, textStatus) {
7                   appendSessionView(data);
8               })
9               .fail(function(err) {
10                  $('#greeterHeading').css({"color": "#C1121C"}).html(
11                      'Die eingegebene Session ID ist nicht vergeben.'
12                      + 'Bitte kontrolliere Deine Eingabe oder versuche '
13                      + 'es zu einem späteren Zeitpunkt noch einmal.');
14              })
15              .always(function() {});
16          } else {
17              $('#greeterHeading').css({"color": "#C1121C"}).html(
18                  'Es wurde keine Eingabe getätigt. '
19                  + 'Bitte gib Deine Session-ID ein.');
20          }
21      }
```

Listing 1: Auszug aus app.js (Webclient)

Der Client ruft hierbei Daten vom Webserver, die dort unter einer bestimmten Session ID gespeichert sind, ab. Bei einer ungültigen ID sendet der Webserver den Fehlercode 404 zurück an den Webclient und der Fail-Zweig der If-Abfrage wird ausgeführt. Bei einer gültigen Session ID wird der Success-Zweig ausgeführt und die Daten vom Webserver an die Funktion appendSessionView übergeben.

In der Funktion appendSessionView werden die Daten vom Webserver verarbeitet und dynamisch in die *index.html* übergeben.

```
1       var imagesHTML = '';
2       sessionData.images.forEach(function(image) {
3           var link = '<div class="col-xs-6 col-sm-5 col-md-4 col-lg-3">'
4               + '<a id="thumbnail" href="/api/images/' + image + '" '
5               + 'class="thumbnail" data-gallery>';
6           link += '<img src="/api/thumbnails/'
7               + image + '" class="img-responsive" width="" height="" alt="'
8               + image + '" /></a>';
9
10          link += '</div>';
11          imagesHTML += link;
12      });
```

Listing 2: Auszug aus app.js (Webclient)

Für jedes Bild wird hier in HTML-Syntax eine Div-Box mit den entsprechenden Attributen erstellt.

```
1       var name = sessionData.client.firstname + ' '
2           + sessionData.client.lastname;
3       $('#greeterHeading')
4           .css({"color": "#8D9091"})
5           .html('Hallo ' + name
6               + '!<br/> Die Bilder Deiner Session liegen hier bereit. '
```

```
7             + 'Du kannst Dich durch alle Bilder durchklicken und '
8             + 'sie online betrachten. Viel Spass!');
```

Listing 3: Auszug aus app.js (Webclient)

Auch der Name wird aus den JSON Daten des Servers hier verarbeitet und mit weiterem Text an die *index.html* übergeben.

Die Anwendung soll sowohl für Mobile- als auch für Desktopdevices geeignet sein. Daher soll es einige Unterschiede in der Darstellung das jeweilige Device geben. Beispielweise die eingangs erwähnte Darstellung der Bilder (initiale Ansicht in der Lightbox bei Desktop-Browsern mit der Option diese zu verändern und die Borderless/Fullscreen Ansicht für mobile Endgeräte).

Die Unterscheidung der Geräte wird durch Funktionen, die die entsprechenden Geräte erkennen und an die Variable isMobile zurück geben. Durch eine If-Abfrage wird die Darstellung der Website für das entsprechende Device angepasst.

```
1     // Variable to detect mobile devices
2     var isMobile = {
3         Android: function() {
4             return navigator.userAgent.match(/Android/i);
5         },
6         BlackBerry: function() {
7             return navigator.userAgent.match(/BlackBerry/i);
8         },
9         iOS: function() {
10            return navigator.userAgent.match(/iPhone|iPad|iPod/i);
11        },
12        Opera: function() {
13            return navigator.userAgent.match(/Opera Mini/i);
14        },
15        Windows: function() {
16            return navigator.userAgent.match(/IEMobile/i);
17        },
18        any: function() {
19            return (isMobile.Android() || isMobile.BlackBerry()
20                || isMobile.iOS() || isMobile.Opera() || isMobile.Windows());
21        }
22    };
23
24    // Gallery view for mobile and desktop devices
25    if(isMobile.any()) {
26        // Fullscreen and borderless gallery view for mobile devices
27        $('#borderless-checkbox').prop('checked', function () {
28            var borderless = $(this).is(':checked');
29            $('#blueimp-gallery')
30                .data('useBootstrapModal', !borderless);
31            $('#blueimp-gallery')
32                .toggleClass('blueimp-gallery-controls', borderless);
33        });
34
35        $('#fullscreen-checkbox').prop('checked', function () {
```

```
36              $('#blueimp-gallery').data('fullScreen', $(this).is(':checked'));
37          });
38      } else {
39          // Initial Lightbox view for desktop devices
40          $('#borderless-checkbox').on('change', function () {
41              var borderless = $(this).is(':checked');
42              $('#blueimp-gallery')
43                  .data('useBootstrapModal', !borderless);
44              $('#blueimp-gallery')
45                  .toggleClass('blueimp-gallery-controls', borderless);
46          });
47          $('#fullscreen-checkbox').on('change', function () {
48              $('#blueimp-gallery')
49                  .data('fullScreen', $(this).is(':checked'));
50          });
51      }
```

Listing 4: Auszug aus app.js (Webclient)

In diesem Beispiel wird die Ansichtsart der Bilder für die entsprechenden Devices festgelegt. Im ersten If-Zweig, welchher bei der Nutzung mobiler Geräte aufgerufen wird, werden durch prop() die Checkboxen, welche zur Auswahl der Darstellung dienen, initial als checkt gesetzt und somit die Darstellung auf Borderless/Fullscreen gesetzt.

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 3 dargestellt:

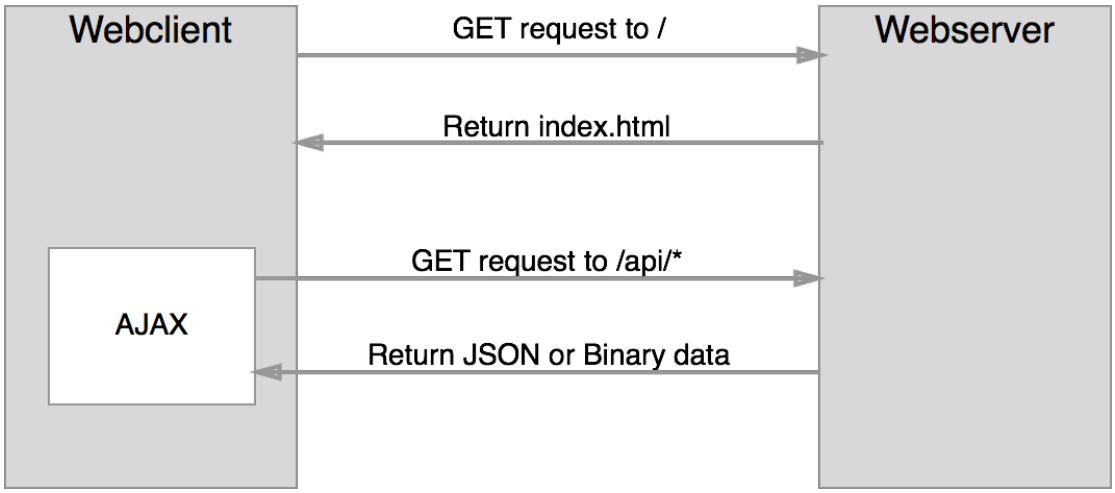

Abbildung 3: Kommunikation zwischen Webclient und Webserver

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 diese eine Seite / Page aus.

In der Abbildung 3 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 2 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 der gegebenen ID auf
GET	/api/thumbnails/:id	ruft ein Thumbnail des Bildes mit der gegebenen ID ab
GET	/api/images/:id	ruft ein Bild mit der gegebenen ID ab

Tabelle 2: 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 4:

Abbildung 4: 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 5 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 5: Auszug aus app.js (Webserver)

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 beispielhaft in wie Listing 6 abgebildet implementiert (die tatsächliche Implementierung ist aufgrund eines Refactorings weniger anschaulich, arbeitet aber identisch):

```
1  /* GET home page. */
2  router.get("/", function(req, res, next) {
3      res.sendFile('index.html');
4  });
5
6  /* GET requests to
7   *        /api/sessions/:id,
8   *        /api/thumbnails/:id and
9   *        /api/images/:id
10  */
11 router.get("/sessions/:id", function(req, res, next) {
12     var sessionId = req.params.id;
13     // create options object for the following request
14     var options = {
15         host: serverData.host,
16         port: serverData.port,
17         path: "/api/sessions/" + sessionId,
18         method: "GET",
19         accept: "application/json"
20     };
21
22     // create and send request
23     http.request(options, function(response) {
24     // check response for error
25         if (response.statusCode == "404") {
26             return res.status(404).send();
27         }
28         // in case of images pipe incoming stream to outgoing stream
29         // response.pipe(res);
30
31         var resData = "";
32         // react to the server's response...
33         response.on("data", function(data) {
34         // ... by passing the responded data to variable resData...
35             resData += data;
36         });
37
38         // ...and returning it to the client in the end
39         response.on("end", function() {
40             // ...as JSON if dealing with session data...
41             res.json(JSON.parse(resData));
42             // ...or as binary data in case of images
43             // res.set("Content-Type", contentType);
44             // res.end(resData, "binary");
45         });
46     }).end();
47 });
```

Listing 6: 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".

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 5 ersichtlich:

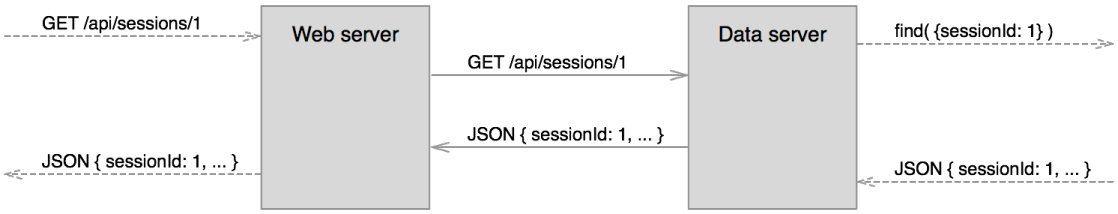

Abbildung 5: Kommunikation zwischen Webserver und Datenserver

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.

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

4.1 Schnittstellen

Die Endpunkte der Schnittstelle, die der Datenserver zur Kommunikation anbietet, sind der folgenden Tabelle 3 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 3: 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 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 6 ersichtlich:

```
IMG-Server
├── controllers
|   ├── images.js
|   ├── index.js
|   ├── sessions.js
|   ├── thumbnails.js
|   └── videos.js
├── models
|   ├── image.js
|   ├── session.js
|   └── video.js
├── app.js
├── config.js
└── package.json
```

Abbildung 6: 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 7 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 7: 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 7 dargestellt.

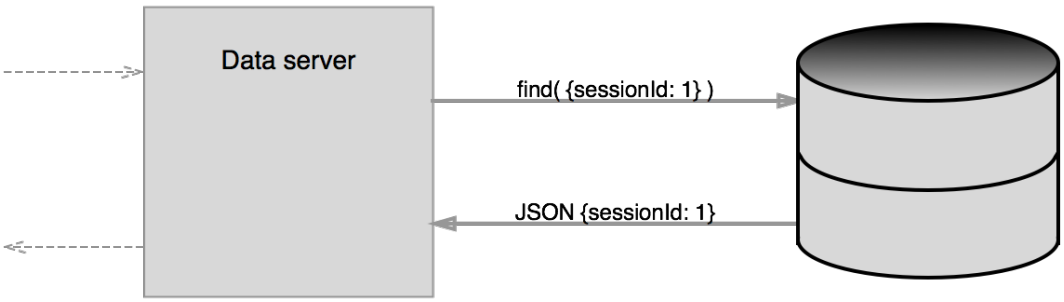

Abbildung 7: Kommunikation zwischen Datenserver und Datenbank

Der Datenserver richtet Anfragen unter Verwendung des selbstgeschriebenen 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.

[4]Create, Read, Update, Delete

6 Lessons learned

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

- Die Speicherung von Binärdaten, die über einen Browser an den Server geschickt werden, erfordert die korrekte Handhabung von HTTP-Requests und deren Codierung.
- Die Verwendung der vorliegenden Architektur mit der ausgesuchten Software ist bereits bei einer geringen Anzahl an (größeren) Bildern nicht performant.
- Insbesondere bei Responses ist auf die korrekte Anwendung von HTTP und die Manipulation von Headern zu achten.
- MongoDB behandelt zu speichernde Objekte / Dokumente bis zu 16 MB gleich. Größere Objekte müssen unter Anwendung des sog. GridFS gespeichert werden.