

StreamSwipe Tinder für Filme

Leon Gieringer, Robin Meckler, Vincent Schreck

Studienarbeit

6. April 2021

Inhaltsverzeichnis

1	Einleitung	1
1.1	1
1.2	Aufbau der Arbeit	1
2	Motivation	1
3	Theoretische Grundlagen	2
3.1	Netzwerkprotokolle	2
3.1.1	Schichtenmodell	2
3.1.2	HTTP	3
3.2	Framework	6
3.2.1	Node.JS	7
3.2.2	Architektur TODO Aufzählung	7
3.2.3	Module TODO Aufzählung	9
3.2.4	npm TODO Aufzählung	9
3.2.5	Express TODO Aufzählung	10
3.2.6	Mongoose TODO Aufzählung	13
3.2.7	Weitere Module TODO Aufzählung	16
3.3	Language	17
3.3.1	JavaScript	18
3.3.2	Historie TODO Aufzählung!	18
3.3.3	Wesentliche Programmiereigenschaften TODO	18
3.3.4	Anwendungsgebiete TODO	19
3.4	IDE	20
3.5	MongoDB	20
3.6	Representational State Transfer - Application Programming Interface	20
3.7	Firebase	20
3.8	Recommendationssystem	20
3.8.1	Collaborative Filtering	20
3.8.2	Content-based filtering	21
3.8.3	Cold Start Problem	21
4	Konzept?	21
5	Auswahl geeigneter Technologie	21
5.1	Server	21
5.2	Datenbank	22
5.3	Kommunikationsschnittstelle	22
6	Backend-Implementierung	22
6.1	Server	22
6.1.1	Einrichtung	22
6.1.2	Sicherheit	22
6.1.3	Webserver	22
6.2	Datenbank	22
6.2.1	Einrichtung	22
6.3	Kommunikationsschnittstelle	22
6.3.1	Implementierung	23

7 Funktionen/Komponenten	23
7.1 Swipe/Aussuchen/Voting	23
7.2 Matches/Chat	23
7.3 Film-/Serienvorschläge	23
7.4 Gruppenorgien	23
7.5 Gespeicherte Filme/Filmliste	23
7.6 Zugänglichkeit/Behindertenfreundlichkeit	23
8 Benutzeroberflächen	23
8.1 Home-Screen	23
8.2 Gruppen	23
8.3 Chat	23
8.4 Filmliste	23
9 CodeBeispiele	23
10 Probleme	23
11 Fazit	23

1 Einleitung

1.1 ..

1.2 Aufbau der Arbeit

2 Motivation

3 Theoretische Grundlagen

3.1 Netzwerkprotokolle

3.1.1 Schichtenmodell

Eine der gängigsten Arten der Kommunikation findet heutzutage über das Internet statt. Dabei handelt es sich um ein weltweit verbundenes Netz von Rechnern. Zur Gewährleistung einer effizienten und geregelten Datenübertragung der heterogenen Computer im Internet wurden Regelwerke, die sogenannten Netzwerkprotokolle, benötigt. Um das Jahr 1980 wurden daraufhin von verschiedenen Computerherstellern modularisierte Protokolle entwickelt, die fortan als Standard für die digitale Übertragung innerhalb von Rechnernetzen gelten sollen. [1.0] Es musste eine Vielzahl von Aufgaben bewältigt und Anforderung bezüglich Zuverlässigkeit, Sicherheit, Effizienz etc. erfüllt werden. Die Aufgaben reichten dabei von der elektronischen Übertragung der Signale bis zur geregelten Reihenfolge der in der Kommunikation abstrakteren Aufgaben. [1.05] Aus den zu lösenden Problemen und Anforderung kristallisierten sich sieben Schichten bzw. Ebenen heraus. Jede einzelne Schicht setzt dabei separat eine Anforderung um und kann dabei durch verschiedene Protokolle realisiert werden. In dem sich etablierten OSI-Schichtenmodell bauen die einzelnen Schichten aufeinander auf, wobei die unterste Schicht das Fundament ist. Die Open Systems Interconnection (OSI) wurde von der International Organization for Standardization (ISO), der Internationalen Organisation für Normung, als Grundlage für die Bildung von offenen Kommunikationsstandards entworfen.

Zusätzlich zum OSI-Modell existiert das in den 1960er-Jahren entwickelte TCP/IP-Referenzmodell. Entwickler dieses Schichtmodells war das das Verteidigungsministerium der Vereinigten Staaten, auch bekannt als das Departments of Defense (DoD). Dementsprechend trägt das TCP/IP-Referenzmodell auch den Namen DoD-Schichtenmodell. [1.06]

OSI-Schicht	TCP/IP-Schicht	Beispiel
Anwendungen (7)	Anwendungen	HTTP, UDS, FTP, SMTP, POP, Telnet, DHCP, OPC UA
Darstellung (6)		TLS, SOCKS
Sitzung (5)		
Transport (4)	Transport	TCP, UDP, SCTP
Vermittlung (3)	Internet	IP (IPv4, IPv6), ICMP (über IP)
Sicherung (2)	Netzzugang	Ethernet, Token Bus, Token Ring, FDDI
Bitübertragung (1)		

Abbildung 1: OSILAYER [1.1]

OSI-Schicht	Aufgabe
Anwendungen	Funktionen für Anwendungen, sowie die Dateneingabe und -ausgabe.
Darstellung	Umwandlung der systemabhängigen Daten in ein unabhängiges Format.
Sitzung	Steuerung der Verbindungen und des Datenaustauschs.
Transport	Zuordnung der Datenpakete zu einer Anwendung.
Vermittlung	Routing der Datenpakete zum nächsten Knoten.
Sicherung	Fehlererkennungsmechanismen / Segmentierung der Pakete in Frames und Hinzufügen von Prüfsummen.
Bitübertragung	Umwandlung der Bits in ein zum Medium passendes Signal und physikalische Übertragung.

Tabelle 1: Kurzbeschreibung der OSI-Schichten [1.2]

IP

In der Vermittlungsschicht des OSI-Schichtenmodells findet, unabhängig des Übertragungsmediums und der genutzten Topologie, die logische Adressierung der Endgeräte statt. Das geläufigste Protokoll dafür ist das Internet Protocol (IP). Jedem am Netz verbundenen Teilnehmer wird eine IP-Adresse zugewiesen. Die bekannteste Notation ist die 32 Bit lange IPv4-Adressen und die IPv6-Adressen mit einer Größe von 128 Bit.

TCP/UDP

In der Transportschicht wird eine Ende-zu-Ende-Kommunikation ermöglicht. Sie ist das Bindeglied zwischen den anwendungsorientierten und den transportorientierten Schichten. Die geläufigsten Protokolle sind das verbindungslose, unzuverlässige, aber weniger Overhead belastete User Datagram Protocol (UDP) und das verbindungsorientierte und datentransferzuverlässige Transmission Control Protocol (TCP). Jedes netzwerkfähige Gerät enthält eine Vielzahl von Ports, die primär zur Unterscheidung zwischen Datenströmen aus Anwendungen bei Netzwerkverbindungen genutzt werden. Anhand des genutzten Ports bei Netzwerkanfragen wissen Webserver, welches Protokollverfahren genutzt werden soll.

3.1.2 HTTP

Das Hypertext Transfer Protocol, kurz HTTP, ist ein zustandloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht.

Kommunikation

Unter einer Nachricht versteht man die HTTP die Kommunikationseinheiten zwischen dem Zentralrechner (Server) und dem, der einen Dienst vom Server abrufen (Client). Man unterscheidet dabei zwischen der Anfrage (Request) vom Client an den Server und der Antwort (Response) als Reaktion vom Server zum Client.

Eine Nachricht besteht aus dem Nachrichtenkopf (Message Header, kurz Header) und dem Nachrichtenrumpf (Message Body, kurz Body). Der Header enthält generelle Informationen über die Nachricht wie zum Beispiel den Methodentyp, das Datenformat, den genutzten Kompressionsalgorithmus, die Länge der Nachricht oder die verwendete Kodierung im Body.

Die erste Zeile des Nachrichtenkops ist dreiteilig und besteht bei der Anfrage aus dem Namen der Anfragemethode, dem Pfad zur angeforderten Ressource (Uniform Resource Locator, kurz URL) und der verwendeten HTTP-Version. Die Anfangszeile einer HTTP-Antwort dagegen besteht zunächst aus der verwendeten HTTP-Version, gefolgt von dem zweiteiligen Status-Code. Anschließend folgt eine Reihe von Headerzeilen, wobei jede Zeile aus einem Schlüsselwort/Wert-Paar besteht und die für die Datenübertragung wichtigen Informationen übergibt. Der Nachrichtenrumpf, der mit den Nachrichtenkopf über einen Zeilenumbruch syntaktisch voneinander getrennt wird, enthält schließlich die Nutzdaten. TODO BILD

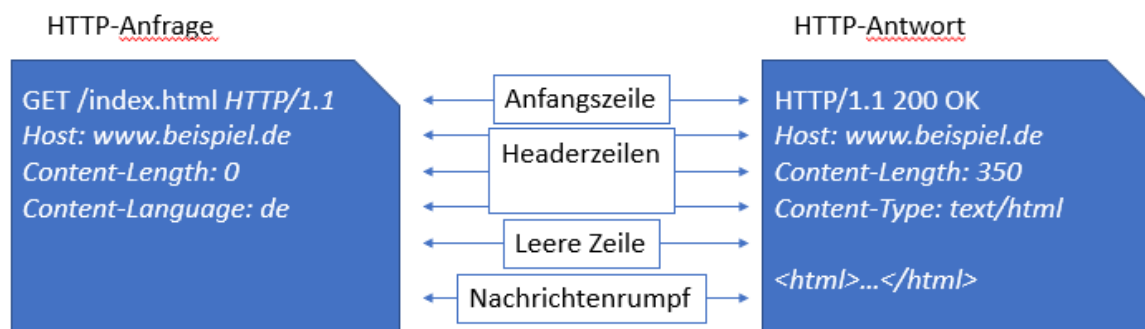


Abbildung 2: HTTP-Nachrichtenaufbau

Methoden

HTTP bietet fest definierte Standard-Methoden für Anfragen, die für verschiedene Aufgaben gedacht sind. Im Folgenden werden die wichtigsten Methoden beschrieben:

1. *GET* ist die gebräuchlichste Methode. Sie fordert vom Server eine Ressource, die bei Erfolg in der Antwort im Body zurückgegeben wird.
2. *POST* ist für die Änderung oder Erzeugung einer Ressource vorgesehen. Dafür werden bei der Anfrage zusätzlich Daten im Body der Nachricht übertragen.
3. *PUT* dient dazu, eine Ressource zu verändern, oder bei Nichtexistenz zu erstellen.
4. *PATCH* ändert eine bestehende Ressource ohne diese wie bei PUT vollständig zu ersetzen.
5. *DELETE* löscht die angegebene Ressource auf dem Server.
6. *OPTIONS* liefert eine Liste von Methoden und Merkmale, die vom Server unterstützt werden.

Statuscodes

HTTP-Antworten senden in der Anfangszeile ihrer Nachricht Statuscodes. Die Angabe ist zweiteilig und besteht aus einer standardisierten Statuskennzahl sowie einer kurzen textuellen Beschreibung, die zusammen Auskunft über den Bearbeitungszustand der zugehörigen Anfrage geben.

<i>The following are the list of HTTP response status codes</i>		
Type	Status Codes	Examples
Informational	1xx	100 Continue, 101 Switching Protocols
Success	2xx	200 - OK , 201 - Created, 202 Accepted
Redirection	3xx	300 Multiple Choices, 301 Moved Permanently, 302 - Found
Client Error	4xx	400 Bad Request, 403 - Forbidden , 404 - Not Found , 422 - Unprocessable Entity
Server Error	5xx	500 - Internal Server Error , 503 - Service Unavailable

Abbildung 3: HTTP-Statuscodes

HTTPS

Das HTTP-Protokoll hat den großen Nachteil, dass die Nachrichten unverschlüsselt und ungesichert übertragen werden. Die Daten können bei der Übertragung von Dritten empfangen, gelesen und verändert werden. Hypertext Transfer Protocol Secure, kurz HTTPS, dem entgegenwirken und die Sicherheit bei der Kommunikation gewährleisten. Dafür dienen zwei Konzepte:

Ersteres ist das Verschlüsseln der Kommunikation von Sender und Empfänger. Die zugrundeliegende Technik nennt sich Transport Layer Security (TLS), ist aber auch als Secure Sockets Layer (SSL) bekannt. Die Idee dahinter ist, dass jeder Teilnehmer der Kommunikation einen öffentlich bekannten Schlüssel (Public Key) und einen geheimen, nicht-öffentlichen Schlüssel (Private Key) enthalten. Über den Public-Key des Empfängers verschlüsselt der Sender seine Nachricht. Diese kann nur über den Private-Key des Empfängers entschlüsselt werden, der vom Empfänger nicht weitergegeben werden sollte.

Das zweite Konzept von HTTPS ist die Webserver-Authentifizierung. Ein Zertifikat, dass zu Beginn der Kommunikation an den Webclient gesendet wird, bescheinigt die Vertrauenswürdigkeit des Senders. Dafür vertrauen Browser- und Betriebssystemhersteller bestimmten Zertifizierungsstellen, deren Zertifikate sie in ihrem Browser bzw. Betriebssystem hinterlegen. Die Kommunikation zwischen Webserver und Webclient findet folglich erst nach vollständiger Authentifizierung.

3.2 Framework

Hier steht mein Framework Text

3.2.1 Node.JS

Im Jahr 2009 veröffentlichte Ryan Dahl das Framework Node.js, das auf Googles V8-Engine, welche auch als JavaScript-Engine in Googles Browser Chrome zum Einsatz kommt, basiert und sich hervorragend für hochperformante, skalierbare und schnelle Webanwendungen eignet. Zudem ermöglicht es Webentwicklern die Entwicklung von serverseitigem JavaScript-Code. - [NodeJS 1.0]

3.2.2 Architektur TODO Aufzählung

Eine wesentliche Eigenschaft von Node.js ist die hohe Performance. Im Folgenden soll der Unterschied der Node.js-Architektur zu traditionellen Webservern und der damit verbundenen höheren Performance dargestellt werden.

Herkömmliche Webserver erstellen zunächst für jede ankommende Anfrage einen neuen Thread. Dieses Vorgehen ist eng mit steigendem Speicher- und Rechenaufwand verbunden. Um sich Rechenzeit, die durch die Erstellung und Zerstörung von Threads entstanden, zu sparen, wurden Threadpools eingerichtet. Dieser Threadpool enthält mehrere Threads, denen Aufgaben zugewiesen werden können. Nach erfolgreicher Abarbeitung einer Operation kann einem Thread eine weitere Aufgabe zugeordnet werden.

Es bleibt aber ein weiteres Problem: Bei der Anfragenabarbeitung kann es zu einer Form von blockierender Ein- und Ausgabe (Blocking Input/Output kurz Blocking I/O) kommen: zum Beispiel beim Suchen in einer Datenbank oder dem Laden einer Datei im Dateisystem. Während der Abarbeitung wartet der Thread solange, bis die Operation ein Ergebnis zurückwirft und belegt dabei weiterhin Speicherplatz. Bei hohem Anfragen-aufkommen kommt es dadurch zu einer hohen Speicherauslastung des Servers. Zudem kosten die Kontextwechsel zwischen den Threads im Betriebssystem weitere Rechenzeit. [Node.js 1.05] Man spricht bei diesem Architekturkonzept auch vom Multi-Threaded Server.

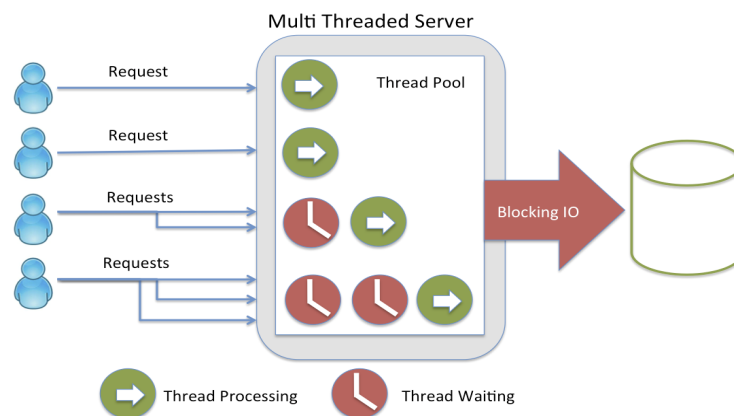


Abbildung 4: Multithreaded / Blocking I/O [Nodejs 1.1]

Node.js verfolgt einen anderen Ansatz: Anfragen werden nur in einem einzigen Thread, dem Hauptthread, abgearbeitet und in einer Warteschlange verwaltet. Dadurch bleiben Kontextwechsel zwischen Threads erspart. Hierbei handelt es sich also um einen Single-Threaded Server. Der Hauptthread verwaltet eine Schleife, die sogenannte Event Loop, die permanent Anfragen aus der Event-Warteschlange überprüft und Ereignisse, die von Ein- und Ausgangsoperationen ausgerufen werden, verarbeitet.

Bei Ankommen einer Nutzeranfrage an einen Node.js Server wird zunächst in der Event Loop geprüft, ob diese Anfrage Blocking I/O benötigt. Falls nicht, kann die Anfrage direkt bearbeitet werden und die Antwort an den Nutzer zurückgesendet werden.

Im anderen Fall wird einer von Node.js interner Workern, welche prinzipiell auch Threads sind, aufgerufen, um die jeweilige Operation auszuführen. Dabei wird eine Callback-Funktion mitgegeben, die vom Worker aufgerufen wird, sobald die Operation ausgeführt wurde. Diese Callback-Funktion kann anschließend als Ereignis von der Event Loop registriert werden. Man spricht hierbei auch von ereignisgesteuerter Architektur. [1.4]

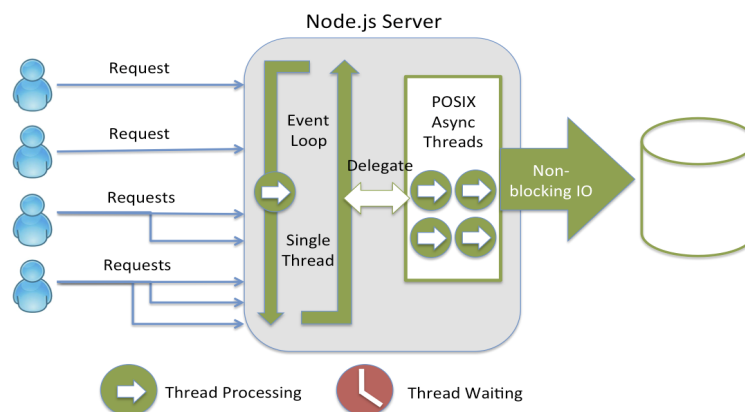


Abbildung 5: Single Threaded / Non Blocking I/O [Nodejs 1.1]

Der große Vorteil hierbei ist, dass der Hauptthread trotz der blockierenden Ein- und Ausgabeoperationen nicht anhält, und weitere Anfragen bearbeiten kann. (Non Blocking I/O - Prinzip)

3.2.3 Module TODO Aufzählung

Module stellen in Node.js Software-Komponenten dar, die Objekte und Funktionen nach außen hin bereitstellen sollen. Sie können aus einer Skriptdatei oder einem Verzeichnis von Dateien bestehen. Module können als einzelne Default-Komponente, die den Hauptteil des Moduls repräsentiert, exportiert werden. Bei der anderen Möglichkeit, des sogenannten ‚benannten Exports‘ werden die zu exportierenden Komponenten dagegen explizit angegeben. Letzteres ist in nachfolgender Abbildung dargestellt.

```
function foo() {}  
function bar() {}  
  
// Obige Funktionen exportieren:  
module.exports.foo = foo;  
module.exports.bar = bar;
```

Abbildung 6: Benannter Export von Modulen

Für den Import stehen verschiedene Möglichkeiten zur Verfügung. Im folgender Abbildung ist ein Import über die `require()`-Funktion dargestellt. Mit mitgeliefertem Modul-Pfad als Parameter gibt diese Funktion ein Objekt des Moduls wieder, dass die exportierten Objekte (und Funktionen) enthält.

```
// Importieren der Funktion in einer anderen Datei:  
const foo = require('./module/path');  
const bar = require('./module/path');
```

Abbildung 7: Import von Modulen

Caching TODO Modules are cached after the first time they are loaded. This means (among other things) that every call to `require('foo')` will get exactly the same object returned, if it would resolve to the same file.

Provided `require.cache` is not modified, multiple calls to `require('foo')` will not cause the module code to be executed multiple times. This is an important feature. With it, partially done objects can be returned, thus allowing transitive dependencies to be loaded even when they would cause cycles. [nodejs 1.21]

3.2.4 npm TODO Aufzählung

Ehemals als Node Package Manager bekannt, ist npm ein Paketmanager für Node.js, entwickelt 2010 von Isaac Z. Schlueter [nodejs 1.3] Es verwaltet ein öffentliches Repository (ein digitales Software-Verzeichnis im Internet) unter dem Name npm Registry. In dem Verzeichnis werden weit über 1 Millionen Pakete (Module) angeboten. [1.4] Der Großteil kann unter freier Lizenz verwendet werden. Mit npm können Module installiert, aktualisiert, entfernt und gesucht werden. Node.js liefert seit seiner Version 0.6.3 npm standardmäßig bei der Installation mit. [1.5]

3.2.5 Express TODO Aufzählung

„Express ist ein einfaches und flexibles Node.js-Framework von Webanwendungen, das zahlreiche leistungsfähige Features und Funktionen für Webanwendungen und mobile Anwendungen bereitstellt.“ [nodejs 1.6] Es wurde im November 2010 von Douglas Christopher Wilson und weiteren Entwicklern veröffentlicht und erweitert Node.js um das Abarbeiten verschiedener HTTP-Methoden, das separate Abarbeiten von Anfragen mit verschiedenen URL-Pfaden sowie weiterer nützlicher Möglichkeiten. Im Grunde handelt es sich bei Express um ein Modul, dass durch den npm Package Manager heruntergeladen werden kann. Die aktuelle Version zum Zeitpunkt der Dokumentation [??] ist 4.17.1. [nodejs 1.65]

Beispiel

Das Erstellen einer einfachen Express-Applikation wird im folgenden Beispiel dargestellt:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World!')
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}!`)
});
```

Abbildung 8: Einfacher Webserver [nodejs 1.8]

Die require()-Funktion importiert das Express-Modul und gibt ein Express-Objekt zurück. Dieses Objekt als Funktion aufgerufen gibt wiederum ein Objekt der Express-Applikation zurück, welche traditionell „app“ genannt wird, das Kernstück des Express-Frameworks ist und sämtliche Methoden wie das Weiterleiten von HTTP Anfragen, das Konfigurieren von Middleware oder das Modifizieren des app-Verhaltens beinhaltet. [nodejs 1.8]

Im mittleren Block befindet sich eine Routendefinition. Die app.get() Funktion spezifiziert eine Callback-Funktion, die ein „request“- und „response“-Objekt als Parameter erhält und aufgerufen wird, sobald eine HTTP Anfrage der Methode GET mit dem Pfad ‚/‘ empfangen wird. Das Request-Objekt enthält sämtliche Informationen über die HTTP-Anfrage. Das Response-Objekt kann dagegen in der Callback-Funktion mit Informationen gefüllt werden und über die send()-Funktion als HTTP-Antwort an den Sender zurückgesendet werden.

Der unterste Block startet den Webserver auf dem mitgegebenen Port über die Funktion app.listen(). Ihr kann auch eine Callback-Funktion mitgegeben werden, die aufgerufen wird, sobald der Server erfolgreich gestartet ist.

Middleware

Express arbeitet nach dem Middleware-Konzept. Darunter versteht man Funktionen, die für die Verarbeitung von Anfragen hintereinandergeschaltet werden können. Jede Middleware hat Zugriff auf das Anfrageobjekt, das Antwortobjekt und die jeweils nächste Middleware-Funktion. [nodejs 1.9] Dabei kann die HTTP-Request direkt terminiert oder an die nächste Middleware gesendet werden. Die Verkettung der Middleware-Funktionen wird in folgender Abbildung illustriert:

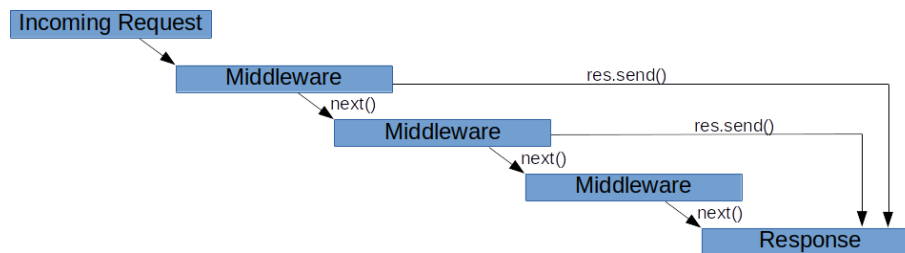


Abbildung 9: Middleware [nodejs 2.0]

Middleware: express.json

Hierbei handelt es sich um eine in express eingebaute Middleware, die die in JSON formatierten Daten im Nachrichtenrumpf aus einer eingehenden HTTP-Anfrage grammatisch analysiert. Dabei ist zu beachten, dass der Nachrichtenrumpf nur dann analysiert wird, wenn bei der Anfrage eine Header-Informationen namens „Content-Type“ mit dem entsprechenden JSON-Typ als Wert übergeben wird. Nach erfolgreicher Analyse erstellt die Middleware aus den JSON-Informationen ein neues body-Objekt innerhalb des übergebenen request-Objekts. [nodejs 2.1]

```
const express = require('express');
const app = express();
app.use(express.json());
```

Abbildung 10: Express.json Middleware benutzen

Middleware: Router

Unter dem Begriff Routing (Weiterleitung) versteht man im Kontext von Express „[...] die Definition von Anwendungsendpunkten (URIs) und deren Antworten auf Clientanforderungen.“ [nodejs 2.15]

Die in express eingebaute Middleware `express.Router` ermöglicht es, modular einbindbare Routenhandler (Weiterleitungsroutinen) zu erstellen. Eine Router-Instanz ist als vollständiges Middleware- und Routingsystem zu sehen und wird deshalb auch als „Mini-App“ angesehen. Der sich durch die Modularität herausziehende Vorteil ist, dass folglich unterschiedliche Anwendungsendpunkte auf entsprechende Dateien ausgelagert werden können.

```
var express = require('express');
var router = express.Router();

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

Abbildung 11: Routinghandler erstellen[nodejs 2.2]

In oberem Beispiel wird ein Routerhandler für das Verzeichnis `/birds` mit eigen implementierter Middleware und zwei Anwendungsendpunkte `/` (bezieht sich auf das Stammverzeichnis) und `/about` erstellt. Der Code wird unter der Datei `birds.js` abgespeichert. Abschließend kann das Routermodul in die Anwendung geladen werden:

```
var birds = require('./birds');
...
app.use('/birds', birds);
```

Abbildung 12: Routinghandler benutzen[nodejs 2.2]

3.2.6 Mongoose TODO Aufzählung

Mongoose ist ein öffentliches Modul, dass zum Zeitpunkt der Dokumentation[??] im npm Package Manager in der Version 5.12.3 zur Verfügung steht. [nodejs 2.4] Bei diesem Modul handelt es sich um ein Object-Document Mapper (ODM), der es ermöglicht, asynchron mit einer NoSql-Datenbank zu kommunizieren. Mongoose ist der populärste und am weitest von MongoDB unterstützte ODM. [nodejs 2.55] Es unterstützt neben transparenter Persistenz auch die Datenvalidierung, das Erstellen von Abfragen (Queries), das Schreiben von logischem Business Code und die Übertragung zwischen Objekten im Code und der Repräsentierung dieser Objekte in der Datenbank.

Object Document Mapping (ODM)

Object-Relational Mappers (ORM) finden hauptsächlich Einsatz in objektorientierten Anwendungen, dessen Daten in relationalen Datenbanken sind. Dabei werden die Tabellen in persistente Objekte gemappt. Das Mappen ist aber auch für NoSQL-Datenbanken nützlich. [nodejs 2.56] Die meistverbreiteten NoSQL-Datenbanken basieren auf Dokument-Systemen. Dementsprechend werden für diese Datenbanken Object-Document Mapper für das Mappen zwischen Dokumenten und Objekten genutzt. Einige ODM's sind Mongoose[nodejs 2.7], Morphia [nodejs 2.8], Doctrine[nodejs 2.9] und Mandango[nodejs 3.0]. NoSQL Mapper nutzen vom Entwickler definierte Datenschemata, die das Objekt beschreiben. Ein daraus abgeleitetes Model-Objekt ermöglicht dann die Kommunikation zwischen dem im Schema beschriebenen Objekt und der entsprechenden Datenbank-Collection.

Schema

Mongoose-Schemata definieren die Struktur der gespeicherten Daten einer MongoDB-Collection in der Anwendungsschicht und werden in der JSON-Notation beschrieben. Dokumentenbasierte Datenbanken wie MongoDB enthalten für jede Wurzelentität [??] eine Collection. Mongoose Schemata werden für jede Collection und jede Nichtwurzel-entität[??] definiert. Innerhalb der JSON-notierten Schemabeschreibung können den einzelnen Eigenschaften bestimmtes Verhalten zugeordnet werden. Zum Beispiel lässt sich explizit der Datentyp angeben (type), eine Eigenschaft verpflichtend (required) oder in Kleinbuchstaben einstellen (lowercase).

```
✓ const schema = new Schema({  
  ✓ attributeX: {  
    type: String,    //Datentyp  
    required: true, //Verpflichtend?  
    lowercase: true //Kleinbuchstaben?  
  }  
});
```

Abbildung 13: Mongoose Schema - Beispiel

Model

Ein Model in Mongoose ist ein aus einer Schemadefinition erstellter Konstruktor, aus denen Objekte instanziiert werden können. Diese Instanzen werden auch ‚documents‘ genannt. Sie stehen in direkter Verbindung zu den jeweiligen Collections der verbundenen Datenbank und enthalten Methoden für die persistente Speicherung, Bearbeitung oder Löschung. Beispielsweise wird beim Abspeichern einer Mongoose Instanz eines Models die entsprechende Collection in der Datenbank erzeugt, sofern sie noch nicht vorhanden ist. Eine Konvention in Mongoose sieht vor, dass der Name eines Models dem Singular eines Nomens entspricht, während die Collections nach dem Plural dieses Namens beschrieben werden. [nodejs 3.2] Im folgenden Beispiel wird ein Model über die `mongoose.model()`-Funktion erstellt unter Angabe des Modelnamens und dem zu verwendenden Schema. Dieses Model wird über `module.exports` nach außen zur Verfügung gestellt.

```
const mongoose = require('mongoose')
const testSchema = new mongoose.Schema({
  attributeX: {
    type: String, //Datentyp
    required: true, //Verpflichtend?
    lowercase: true //Kleinbuchstaben?
  }
});

module.exports = mongoose.model('test', testSchema)
```

Abbildung 14: Model erstellen und exportieren

An anderer Stelle kann das Model nun importiert werden. Aus dem Model kann ein Objekt instanziiert werden, welches über die `save()`-Funktion in der Datenbank gespeichert werden kann.

```
const testModel = require(test);

var testX = new testModel(); //Erstellt neue Instanz
await testX.save(); // Speichert in die Datenbank
```

Abbildung 15: Model importieren, Objekt instanziiieren und persistent speichern

Mongoose Models enthalten ohne Instanziierung des Weiteren auch Schnittstellen, um Daten der zugehörigen Collection zu kreieren, abfragen, bearbeiten oder löschen. (Create, Receive, Update, Delete oder auch kurz CRUD).

```
const testModel = require(test);

//Create
testModel.save({ attributeX: "abc"});
//Receive
var testObjects = await testModel.find();
var testObject = await testModel.findOne({ attributeX: "abc"});
//Update
await testModel.updateOne({ attributeX: "abc"}, {attributeX: "cba"});
//Delete
await testModel.deleteMany({ attributeX: "abc"});
```

Abbildung 16: CRUD-Bespielfunktionen eines Mongoose-Models

Verbindung

Verbindung zur Datenbank kann über die `connect()`-Funktion mit Angabe der genutzten Datenbank und des Datenbankpfads. Über das `mongoose.connection`-Objekt können auf Verbindungsereignisse reagiert werden.

```
const mongoose = require('mongoose')
await mongoose.connect("mongodb://127.0.0.1:27017/TestDatenbank");
mongoose.connection.on('error',(error) => console.log(error));
mongoose.connection.on('open',() => console.log('Connected to DB'));
```

Abbildung 17: Mongoose: Verbindung zur Datenbank aufbauen

Für den Verbindungsaufbau können weitere Option übergeben werden. Dafür kann ein Objekt wie in folgendem Beispiel erstellt werden, dass die zugehörigen Optionen als Attribute beinhaltet.

```
const options = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useCreateIndex: true,
  useFindAndModify: false,
  autoIndex: false, // Don't build indexes
  poolSize: 10, // Maintain up to 10 socket connections
  serverSelectionTimeoutMS: 5000, // Keep trying to send operations for 5 seconds
  socketTimeoutMS: 45000, // Close sockets after 45 seconds of inactivity
  family: 4 // Use IPv4, skip trying IPv6
};
mongoose.connect(uri, options);
```

Abbildung 18: Mongoose Verbindungsoptionen[nodejs 3.3]

Express-Modul	Beschreibung
fs	Erlaubt die Interaktion mit dem Dateisystem. Zum Beispiel Schreiben/Lesen von Dateien.
http	Ermöglicht Datentransfer über das Protokoll HTTP und das Abhören eines Ports.
https	Gesicherte Variante zu HTTP mit SSL. Benötigt Private Key und Zertifikat.
firebase-admin	Ermöglicht die Verbindung zu Google Firebase Cloud.
node-cron	Ermöglicht das Einstellen von sich wiederholenden Aufgaben zu bestimmten Zeitintervallen.

Tabelle 2: Module

3.2.7 Weitere Module TODO Aufzählung

TODO warum Tabelle oben?

3.3 Language

Hier steht mein Language Text.

3.3.1 JavaScript

In den nächsten Unterkapiteln soll zunächst ein historischer Überblick über die Programmiersprache JavaScript gegeben werden. Im Anschluss wird auf die Bedeutung und Nutzung von JavaScript eingegangen.

3.3.2 Historie TODO Aufzählung!

Ihren Ursprung findet die Programmiersprache JavaScript im Jahr 1995, als Brendan Eich, ein damaliger Ingenieur des US-amerikanischen Software-Unternehmens „Netscape Communications Corporation“, innerhalb von zehn Tagen diese Sprache für den Browser „Netscape Navigator“ entwickelt hat. [1] Das Ziel dabei war es, eine Skriptsprache zu entwickeln, die es Entwicklern möglich machen sollte, auf ihren Webseiten Skripte umzusetzen. Zunächst noch unter dem Namen Mocha und LiveScript änderte sich der Name zu JavaScript aufgrund der Kooperation von Netscape und Sun, der Firma hinter der Programmiersprache Java, und Marketinggründen. Netscape wollte von der damaligen Popularität von Java profitieren. [1.05]

Netscape's Veröffentlichung des Netscape Navigator 2.0, der erste Browser der JavaScript unterstützte, brachte Microsoft dazu, Netscape als ernstzunehmenden Konkurrenten zu sehen. Microsoft antwortete im August 1995 mit der Veröffentlichung des ersten Internet Explorer zusammen mit der Skriptsprache JScript, die einen Dialekt der Sprache JavaScript darstellt. Dies ist ferner als der Beginn der „Browserkriege“ bekannt. [1.06]

Im Jahre 1997 reichte Netscape JavaScript an die European Computer Manufacturers Association (kurz ECMA[ABK]), einer privaten, internationalen Normungsorganisation zur Normung von Informations- und Kommunikationssystemen und Unterhaltungselektronik ein. Das Ziel war es, von der ECMA einen einheitlichen Standard für die Sprache schaffen zu lassen, die fortan weiterentwickelt werden und von weiteren Browserherstellern genutzt werden soll. Das resultierende Standard nennt sich ECMAScript, wobei JavaScript die bisher bekannteste Implementierung dieses Standards ist. [1.07] Andere Implementierungen sind zum Beispiel ActionScript von Macromedia, JScript von Microsoft und ExtendScript von Adobe.

Jährlich wird dieser Standard seit Juni 2015 erweitert. ECMAScript Version 11 beziehungsweise ECMAScript 2020 bildet zum Zeitraum dieser Dokumentation [??] den aktuellen Standard. [1.08] Im Juni 2021 soll die neueste Version ECMAScript 2021 veröffentlicht werden. [1.09]

3.3.3 Wesentliche Programmieigenschaften TODO

„JavaScript is Not Java“ [1.091 ??]. Die Programmiersprache JavaScript wird aufgrund ihrer Namensgebung oft in falsche Zusammenhänge zu Java gebracht. Das häufigste Missverständnis sei, JavaScript wäre eine vereinfachte Version von Java. [1.091]

JavaScript ist eine interpretierte Programmiersprache mit objektorientierten Umsetzungsmöglichkeiten. Interpretation ist in diesem Zusammenhang so zu verstehen, dass der Quellcode zur Laufzeit eines Programms gelesen, übersetzt und ausgeführt wird. Syntaktisch ähnelt JavaScript kompilierten Programmiersprachen wie C, C++ und Java durch gleiche Umsetzung der Kontrollstrukturen wie den Bedingungen, Schleifen oder den booleschen Operatoren. [1.1] Wesentliche Unterschiede sind dagegen, dass JavaScript zum einen eine schwach-typisierte Sprache ist. Durch die schwache Typisierung haben Variablen keinen festen Datentyp und können diesen

dynamisch zur Laufzeit ändern. Des Weiteren findet bei JavaScript die Objektorientierung prototypenbasiert statt. Diese Form der Programmierung wird auch klassenlose Objektorientierung bezeichnet. Anders als bei der klassenbasierten Programmierung, bei der Objekte aus vordefinierten Klassen instanziiert werden, werden hier Objekte durch Klonen bereits existierender Objekte erzeugt. Die Objekte, die geklont werden, sind dabei als Prototyp-Objekte zu verstehen. Beim Klonen werden alle Attribute und Methoden des Prototyp-Objekts in das neue Objekt übernommen und können dort überschrieben sowie erweitert werden. Objekte in JavaScript sind eher als Zuordnungslisten, ähnlich wie assoziative Arrays oder Hash-Tabellen, anzusehen, da bei der Eigenschaftszuweisung lediglich ein Mapping eines Namens (dem Key) zu seiner zugehörigen Eigenschaft (dem Value) stattfindet.



Abbildung 19: JavaScript Objekt [1.29]

Ein weiterer Unterschied zu den anderen Programmiersprachen ist, dass alle Funktionen und Variablen außer der primären Datentypen Boolean, Zahl und Zeichenfolge, als Objekte verstanden werden können.

3.3.4 Anwendungsgebiete TODO

Ursprünglich fand JavaScript seinen Einsatz hauptsächlich darin, dynamische Webseiten im Webbrowser anzuzeigen. Die Verarbeitung erfolgte dabei meist clientseitig durch den Webbrowser (dem sogenannten Frontend). [1.3]

Heutzutage findet sich die Sprache dagegen in wesentlich größeren Einsatzgebieten wieder. Bis vor einigen Jahren war die Serverseite anderen Programmiersprachen wie Java oder PHP vorbehalten. Die Veröffentlichung von Node.js, einer plattformübergreifenden Laufzeitumgebung, die JavaScript außerhalb eines Webbrowsers ausführen kann, führte zu einer immer größeren Verbreitung von serverseitigen Anwendungen (dem Backend), die auf JavaScript basieren. Auf Node.js wird ausführlicher im nächsten Kapitel eingegangen. Ferner findet JavaScript heutzutage aber auch seinen Einsatz in mobilen Anwendungen, Desktopanwendungen, Spielen oder 3D-Anwendungen. [1.4]

3.4 IDE

Hier steht mein IDE Text.

3.5 MongoDB

Hier steht mein Database Text.

Es gibt verschiedene Datenbankmodelle.

3.6 Representational State Transfer - Application Programming Interface

Hier steht mein Database Text.

Es gibt verschiedene Datenbankmodelle.

3.7 Firebase

Hier steht mein Firebase Text.

3.8 Recommendationssystem

Auf der Webseite Youtube allein werden minütlich mehr als 500 Stunden Videomaterial hochgeladen. (<https://blog.youtube/press/>, 10.02.2021) Um bei einer solch unvorstellbaren Menge an Daten (allein auf einer Webseite) den Überblick als Endnutzer behalten zu können, ist ein personalisiertes Filtersystems unausweichlich.

Solche Filtersysteme, auch Recommendation System genannt, nutzt bisher gesammelte Daten um Nutzern potentiell interessante Objekte jeweils individuell vorzuschlagen. Ein sogenannter *Candidate Generator* ist hierbei ein Recommendation System, welches die Menge M als Eingabe erhält und für jeden Nutzer eine Menge N ausgibt. Hierbei umfasst M alle Objekte und gleichzeitig gilt $N \subset M$.

Die Bestimmung einer solchen Menge N beruht grundlegend auf zwei Informationsarten. Erstens die sogenannten Nutzer-Objekt Interaktionen, also beispielsweise Bewertungen oder auch Verhaltensmuster; Und zweitens die Attributwerte von jeweils Nutzer oder Item, also beispielsweise Vorlieben von Nutzern oder Eigenschaften von Items. (Charu C. Aggarwal - Recommender Systems) Systeme, welche zum Bewerten ersteres benutzen, werden *collaborative filtering* Modelle genannt. Andere, welche zweiteres verwenden, werden *content-based filtering* Modelle genannt. Wichtig hierbei ist jedoch, dass *content-based filtering* Modelle ebenfalls Nutzer-Objekt Interaktionen (v.a. Bewertungen) verwenden können, jedoch bezieht sich dieses Modell nur auf einzelne Nutzer - *collaborative filtering* basiert auf Verhaltensmustern von allen Nutzern bzw. allen Objekten.

Ein solches Recommendation System kann im einfachsten Fall wie in 3.8 als Matrix dargestellt werden.

3.8.1 Collaborative Filtering

Unter *collaborative filtering* versteht man das Betrachten von Ähnlichkeiten im Verhalten von Nutzern anhand von Bewertungen und Präferenzen, bzw. anhand der Ähnlichkeiten von Objekten.

Diese Art leidet sehr unter dem *sparsity* Problem, also dass die Nutzer zu wenige Bewertungen von Objekten ausüben. Daher sind Vorhersagen über Ähnlichkeit von Nutzern aufgrund unzureichender Datensätze nicht sinnvoll möglich(siehe 3.8.3).

Generell unterscheidet man in zwei Typen:

		Items					
		1	2	...	i	...	m
Users	1	2		1			3
	2	4			5		
	...			1			4
	u		4		5		1
		2				3	
	n		4		3		

Tabelle 3: Nutzer-Item Matrix mit Bewertungen. Jede Zelle $r_{u,i}$ steht hierbei für die Bewertung des Nutzers u an der Stelle i

1. *Memory-based Methoden*: Es wird, wie oben beschrieben, aus gesammelten Daten Ähnlichkeit herausgearbeitet und Nutzer-Objekt Kombinationen durch eben diese vorhergesagt. Daher wird dieser Typ auch *neighborhood-based collaborative filtering* genannt. Man unterscheidet weiter in:
 - (a) *User-based*: Ausgehend von einem Nutzer A werden Nutzer mit ähnlichen Nutzer-Objekt Kombinationen gesucht, um Vorhersagen für Bewertungen von A zu treffen. Ähnlichkeitsbeziehungen werden also über die Reihen der Bewertungsmatrix berechnet.
 - (b) *Item-based*: Hierbei werden ähnliche Objekte gesucht und diese genutzt um die Bewertung eines Nutzers für ein Objekt vorherzusagen. Es werden somit Spalten für die Berechnung der Ähnlichkeitsbeziehungen verwendet.
2. *Model-based Methoden*: Machine Learning und Data Mining Methoden werden verwendet um Vorhersagen über Nutzer-Objekt Kombinationen zu treffen. Hierbei sind auch gute Vorhersagen bei niedriger Bewertungsichte in der Matrix möglich.

(Charu C. Aggarwal - Recommender Systems)

3.8.2 Content-based filtering

3.8.3 Cold Start Problem

<https://dl.acm.org/doi/pdf/10.1145/3383313.3412488> <http://www.microlinkcolleges.net/elib/files/undergraduate/Photography/504703.pdf>

kann man das als Quelle angeben??

4 Konzept?

Test..

5 Auswahl geeigneter Technologie

Hier steht mein TechnologieAuswahl Text...

5.1 Server

Anforderungen sind ...

5.2 Datenbank

Es gibt verschiedene Datenbankmodelle. Bei grossen Datenbank spielt .. eine wichtige Rolle..

5.3 Kommunikationsschnittstelle

Verschiedene Modelle für die Kommunikation verteilter Systeme...

6 Backend-Implementierung

Unter Backend versteht man... , Sie differenziert durch... von den

6.1 Server

Hier steht mein BackendServer Text.

6.1.1 Einrichtung

Zunächst...

6.1.2 Sicherheit

Den Server gilt es zu schützen.

6.1.3 Webserver

Den Server gilt es zu schützen.

6.2 Datenbank

Hier steht mein BackendDatenbank Text.

6.2.1 Einrichtung

Hier steht mein Imlementierung Text.

6.3 Kommunikationsschnittstelle

Hier steht mein BACKENDKommunikationschnittstelle Text.

6.3.1 Implementierung

7 Funktionen/Komponenten

7.1 Swipe/Aussuchen/Voting

7.2 Matches/Chat

7.3 Film-/Serienvorschläge

7.4 Gruppenorgien

7.5 Gespeicherte Filme/Filmliste

7.6 Zugänglichkeit/Behindertenfreundlichkeit

8 Benutzeroberflächen

8.1 Home-Screen

8.2 Gruppen

8.3 Chat

8.4 Filmliste

9 CodeBeispiele

10 Probleme

11 Fazit