
Hochschule für Technik und Wirtschaft Berlin

Fachbereich 4

# Bachelorarbeit

im Studiengang Internationale Medieninformatik

| | |
|---|---|
| **Thema:** | Entwicklung einer prototypischen Freizeitaktivitäts-Plattform als responsive Webanwendung |
| **Eingereicht von:** | Markus Arendt (s0538148@htw-berlin.de) |
| **Eingereicht am:** | 04. April 2016 |
| **Betreuer:** | 1. Prof. Dr.-Ing. David Strippgen<br>2. Prof. Dr. Tobias Lenz |

**Markus Arendt**

*Entwicklung einer prototypischen Freizeitaktivitäts-Plattform als responsive Webanwendung*

*Hochschule für Technik und Wirtschaft Berlin*

*Wilhelminenhofstraße 75A*

*12459 Berlin*

*Telefon: (030) 5019 0*

*Telefax: (030) 5090 134*

*www.htw-berlin.de*

# Eidesstattliche Erklärung

Ich erkläre, dass ich meine Bachelorarbeit *Entwicklung einer prototypischen Freizeitaktivitäts-Plattform als responsive Webanwendung* selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Ich versichere, dass die eingereichte schriftliche Fassung der auf dem beigefügten Medium gespeicherten Fassung entspricht.

Berlin, den 04. April 2016

Markus Arendt

# Zusammenfassung

Die folgende Arbeit dokumentiert die Entwicklung einer prototypischen Freizeitaktivitäts-Plattform als responsive Webanwendung. Nach abgeschlossener Entwicklung soll es Benutzern dieser Webanwendung möglich sein, nach Unternehmungen anderer Benutzer im eigenen Umfeld zu suchen und daran teilzunehmen. Ebenfalls sollen eigene Unternehmungen, welche genaue Standortangaben beinhalten, angeboten werden können.

Nach vorheriger Analyse und der Erarbeitung von Grundlagen, wird die Implementierung der prototypischen Webanwendung als Schwerpunkt dieser Arbeit behandelt. Dabei kommen überwiegend folgende Technologien zum Einsatz: Node.js, MongoDB, Express, Passport, AngularJS, Google Maps und Bootstrap.

Anschließend wird das Ergebnis der Implementierung betrachtet und ein Resümee daraus gezogen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis



# Tabellenverzeichnis

# 1 Einleitung

Soziale Netzwerke sind zu einem festen Bestandteil des alltäglichen Lebens vieler Menschen geworden. Facebook, das weltweit größte soziale Netzwerk, wird seit dem 31. Januar 2015 aktiv von monatlich 1,59 Mrd. Menschen genutzt. [1] Bei einer für Januar 2016 geschätzten Weltbevölkerung von rund 7,4 Mrd. Menschen [2] sind also ca. 21,49 % aller Menschen bereits Facebook-User. Und Facebook will mehr: In einem Gespräch mit USA Today sagte Mark Zuckerberg, Gründer und Vorstandsvorsitzender von Facebook Inc., dass er bis zum Jahr 2030 bereits 5 Mrd. User erreichen will. [3]

Durch soziale Netzwerke bleiben wir in Kontakt mit Familie und Freunden, lassen diese an unserem Leben teilhaben und verabreden uns für öffentliche oder private Veranstaltungen. Beispiel: Für die anstehende Geburtstagsparty ist schnell ein Event auf Facebook erstellt und mit nur wenigen Mausklicks werden alle Freunde über ihre Einladung benachrichtigt.

Doch wie lernt man neue, bisher unbekannte Menschen kennen? Ein soziales Netzwerk wie Facebook verknüpft den User mit bereits gewonnenen Freunden und darüber hinaus mit deren Freunden. Außerhalb dieses Kreises gibt es nur die Möglichkeit in Gruppen, öffentlichen Profilen und öffentlichen Events mit bisher unbekannten Menschen Kontakt aufzunehmen.

In der folgenden Arbeit wird der Aufbau eines Prototyps einer responsiven Webanwendung dokumentiert, die das Finden potenzieller neuer Freunde durch gemeinsame Freizeitaktivitäten erleichtern soll. Diese Freizeitaktivitäts-Plattform soll speziell Menschen in einem neuen Umfeld dabei unterstützen, andere Menschen mit gleichen Interessen zu finden und sich mit diesen zu treffen. Dabei wird großen Wert daraufgelegt, dass es sich nicht um eine weitere Event- oder Dating-Plattform handelt. Die Webanwendung soll Menschen im gleichen Umkreis dazu anregen, gemeinsam etwas zu unternehmen, um sich so kennenzulernen.

## 1.1 Zielsetzung

Zum Ziel wird die Erarbeitung, Umsetzung und Dokumentation einer prototypischen Freizeitaktivitäts-Plattform als Webanwendung gesetzt. Technisch soll dies mit möglichst modernen Mitteln umgesetzt werden. Zur Benutzung auf mobilen Endgeräten soll der Inhalt der Webanwendung responsiv gestaltet sein. Es sollen Daten von und an eine Datenbank geschickt werden, welche auf diversen Endgeräten möglichst benutzerfreundlich dargestellt werden. Diese Daten sollen, mit Zuhilfenahme von Geodaten, als Standort auf einem Kartendienst gesetzt und später dargestellt werden.

So sollen Benutzer der Webanwendung in der Lage sein, sich für Unternehmungen an einem gewählten Standort zu verabreden, um so neue Menschen aus ihrem Umfeld kennenzulernen.

# 2 Analyse

## 2.1 Anforderungsanalyse

In der folgenden Anforderungsanalyse wird die benötigte Funktionalität des Prototyps der Freizeitaktivitäts-Plattform erarbeitet. Das Resultat der Analyse wird als Ziel der Entwicklung gesetzt.

### 2.1.1 Zielgruppe

Um den Funktionsumfang der Freizeitaktivitäts-Plattform im Rahmen der Bachelorarbeit einzuschränken, werden zwei Personas vorgestellt, die als Benutzer in Frage kommen und spezielle Anforderungen an die Webanwendung stellen.

1. **Kali**, 21 Jahre alt aus Indien, die für ihr Studium nach Berlin zog.
2. **Jan**, 27 Jahre alt aus Berlin, der innerhalb seines Freundeskreises gerne Brettspieleabende veranstaltet.

### 2.1.2 Use Cases

1. **Kali**

   Kali zog vor einem Monat für das Medieninformatik-Studium an der HTW nach Berlin. Sie wohnt in einem Studentenwohnheim nahe der Hochschule. Nach den ersten Wochen im Studium hat Kali noch wenig Kontakt zu ihren Kommilitonen. Ihre Deutschkenntnisse sind schlecht, das neue Umfeld fremd für sie. Sie vermisst ihre Familie und Freunde in Indien und fühlt sich isoliert.

   Kali meldet sich über ihren Laptop auf der Freizeitaktivitäts-Plattform mit ihrem Facebook-Profil an und sucht an ihrem aktuellen Standort nach Unternehmungen im Umkreis von 20 km. Anschließend filtert

sie die Ergebnisse der Umkreissuche nach Unternehmungen mit dem Wort „India“ im Titel. Ihr wird eine passende Unternehmung in einem indischen Restaurant in Kreuzberg angezeigt. Der Gastgeber sucht nach jungen Menschen, die bei einem gemütlichen Abendessen Unterhaltungen in Hindi führen möchten.

Über die Plattform tritt Kali der Unternehmung bei und meldet sich bei dem Gastgeber und einer bereits beigetretenen Benutzerin über einen Kommentar in der Unternehmung, auf das der Gastgeber nach kurzer Zeit reagiert. Er begrüßt Kali herzlich und freut sich über eine weitere Zusage zu seiner Unternehmung.

Bei dem Abendessen erfährt Kali, dass der Gastgeber vor zwei Jahren ebenfalls für ein Studium nach Berlin gezogen ist. Nach dem Abend im Restaurant bleibt sie mit ihm über Facebook in Kontakt. Die Beiden schließen schnell Freundschaft und planen gemeinsam ähnliche Unternehmung zukünftig regelmäßig anzubieten.

2. **Jan**

   Jan ist in Berlin geboren, arbeitet als Erzieher in einem Kindergarten und lebt seit drei Jahren in einer WG mit zwei weiteren Mitbewohnern. Er veranstaltet in seinem Freundeskreis regelmäßig gemütliche Brettspielabende in seiner WG. Jedoch sind in der Regel nur zwei bis drei seiner Freunde daran interessiert.

   Für sein kürzlich erworbenes Brettspiel fehlen ihm mindestens zwei weitere Mitspieler. Bisher hatte er sich nur aus reiner Neugier auf der Freizeitaktivitäts-Plattform angemeldet. Er fasst den Entschluss, seinen Brettspielabend an einem Samstag als eigene Unternehmung anzubieten. Über sein Smartphone erledigt er dies in seiner Mittagspause. Als Standort wählt er die genauen Koordinaten seiner WG, gibt der Unternehmung einen passenden Titel, eine kurze Beschreibung um was genau es sich handelt und ein Datum mit Uhrzeit. Für ihn unerwartet treten seiner Unternehmung drei potenzielle Mitspieler

bei. Über die Kommentarfunktion werden weitere Informationen ausgetauscht. Der Brettspieleabend findet statt und die drei neugewonnenen Mitspieler werden in die regelmäßige Runde mit aufgenommen.

### 2.1.3 Evaluation

Die Personas stellen in den zwei unterschiedlichen Use Cases folgende Anforderungen an den Prototyp der Freizeitaktivitäts-Plattform:

**Kali**:

- Verknüpfung zu Facebook-Profilen von Benutzern
- Umkreissuche für Unternehmungen
- Filter für Inhalte der Unternehmungen
- Unternehmungen beitreten
- Kommentarfunktion in Unternehmungen

**Jan**:

- Unternehmung mit genauem Standort erstellen
- Unternehmung mit Titel, Beschreibung, Datum und Uhrzeit erstellen
- Beigetretene Benutzer sollen in der Unternehmung angezeigt werden
- Unterhaltungen über die Kommentarfunktion der Unternehmung

## 2.2 Wettbewerbsanalyse

Nach vorheriger Ideenfindung in Abschnitt 1 und der Anforderungsanalyse in Unterabschnitt 2.1 wird nach möglichen Wettbewerbern gesucht. Dabei wird die Suche im Rahmen der Bachelorarbeit auf den deutschen Markt beschränkt. Gefunden wurden die zwei Wettbewerber *Spontacts* und *BuddyMe*, die eine ähnliche bis gleiche Idee verfolgen und im Folgenden kurz vorgestellt werden.

**Spontacts** (https://www.spontacts.com) wird durch die Spontacts GmbH entwickelt und kommt der Idee der Freizeitaktivitäts-Plattform sehr nahe. Es existiert als Webanwendung, iOS-App und Android-App. Über Spontacts können Benutzer als „Mitmacher“ Freizeitaktivitäten in ihrer Stadt suchen und daran teilnehmen. Nach passenden Freizeitaktivitäten kann durch die Filter „Was“, „Wo“, „Wann“ und „Wer“ gesucht werden. Gefundene Freizeitaktivitäten sind als Kacheln dargestellt. Benutzer können ebenfalls neue Freizeitaktivitäten erstellen andere Benutzer daran teilnehmen lassen. Dabei sind Titel, Beschreibung, Ort als Adresse, Datum, Uhrzeit und eine Kategorie anzugeben. Um Spontacts benutzen zu können, muss sich der Benutzer entweder direkt auf Spontacts oder über ein Facebook-Profil anmelden. Spontacts ist eine sehr ausgereifte Plattform mit vielen nützlichen Funktionen. Leider findet man in der Webanwendung und den Apps sehr viel Werbung unter den angebotenen Freizeitaktivitäten. Dazu wird ein monatliches Bezahlmodell angeboten, welches die Werbung entfernt und weitere Funktionen freischaltet. Außerdem kann der Standort einer Unternehmung nicht genau bestimmt werden: Als Standort werden nur Adressen und Orte benutzt. Einer neuen Unternehmung muss zwingend eine Kategorie zugewiesen werden. Die angebotenen Kategorien sind nicht durch den Benutzer erweiterbar. Eine Unternehmung, auf die keine dieser Kategorien zutrifft, kann somit nicht genau beschrieben werden.

**BuddyMe** (https://buddyme.me) existiert ausschließlich als Webanwendung, welche für mobile Endgeräte optimiert ist. Auch hier muss sich der Benutzer

direkt auf BuddyMe oder über sein Facebook-Profil anmelden. Unternehmungen sind ebenfalls als Kacheln dargestellt. Hier kann ausschließlich nach einer Stadt, in der die Unternehmung stattfindet, gefiltert werden. Eine Suche nach Unternehmungen im direkten Umfeld ist damit nicht möglich. Darüber hinaus können Inhalte in Unternehmungen über ein Suchfeld gesucht werden. Weitere Filtermöglichkeiten stehen dem Benutzer nicht zur Verfügung. Tritt der Benutzer einer Unternehmung bei, muss dieser eine Nachricht in der Unternehmung hinterlassen. In der Unternehmung können jetzt die Nachrichten aller Teilnehmer gelesen werden. Um eine neue Unternehmung auf BuddyMe anzulegen, muss der Benutzer einen Titel, eine Stadt und eine Beschreibung angeben. Außerdem besteht die Möglichkeit ein Bild hochzuladen, welches in der fertig angelegten Unternehmung angezeigt wird. Der Benutzer kann einen Zeitrahmen, in dem die Unternehmung stattfinden soll, durch Terminvorschläge festlegen. Dieser Zeitrahmen wird durch die Auswahl bestimmter Tage bestimmt. Auf BuddyMe wird, anders als auf Spontacts, nicht mit Standorten, festen Zeitpunkten oder Kategorien gearbeitet. Benutzer können ausschließlich durch Nachrichten in Unternehmungen miteinander kommunizieren.

# 3 Grundlagen

Im folgenden Abschnitt werden zum Verständnis die Grundlagen von Webanwendungen erläutert und moderne Methoden zur Umsetzung dieser genauer betrachtet. Dies führt zu dem Entschluss, welcher sogenannte Software Stack als Grundlage zur Entwicklung der Freizeitaktivitäts-Plattform benutzt wird.

## 3.1 Webanwendungen

Als Webanwendung bezeichnet man eine Anwendung im Browser, welche nach dem Client-Server-Modell arbeitet und i.d.R. zwischen Server und Client über das HTTP-Protokoll kommuniziert. Ein moderner Webbrowser fungiert dabei als Client, der mit dem Server kommuniziert und mit dem der Benutzer arbeitet. So ist für die Benutzung einer Webanwendung weder die Installation einer Software oder App, noch ein spezielles Betriebssystem nötig. Die Rechenlast wird in der Praxis gewöhnlich so verteilt, dass der Server den größten Teil der Arbeit übernimmt und der Client, als sogenannter Thin Client, oft nur zur Navigation dient und die vom Server empfangenen Daten darstellt. [4]

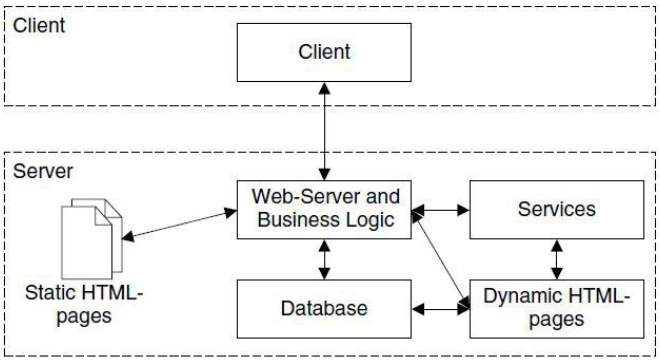


Abbildung 1: 2-Layer Client / Server Architektur [5]

Die Unabhängigkeit von bestimmten Betriebssystemen führt dazu, dass Webanwendungen eine zeit- und kostensparende Alternative zu herkömmlichen Programmen oder Apps geworden sind. Als Entwickler konzentriert man sich auf eine einzige Anwendung, die auf allen Endgeräten funktionieren soll, statt mehrere maßgeschneiderte Lösungen für einzelne Betriebssysteme und Endgeräte, z.B. als native App, bereitzustellen.

## 3.2 Software Stack

> *„Every website you visit is the product of a unique mixture of libraries, languages, and web frameworks.“ [6]*

Ein Software Stack, oder auch Solution Stack genannt, ist eine Sammlung von Software, auf die eine Applikation aufgebaut wird. Für die Programmierung einer Webanwendung ist die Wahl eines Software Stacks eine essenzielle Entscheidung, denn hier wird festgelegt mit welcher Datenbank, Programmiersprache und Serverarchitektur gearbeitet wird. [7]

Als ein Teilziel dieser Bachelorarbeit wurde in Unterabschnitt 1.1 definiert, dass bei der Entwicklung der Freizeitaktivitäts-Plattform mit möglichst modernen Mitteln gearbeitet werden soll. Nach ausführlicher Recherche kamen dafür zwei Software Stacks besonders in Frage, die im Rahmen der Bachelorarbeit auf Tauglichkeit getestet wurden.

Um durch ausführliche Dokumentationen und Tutorials einen schnellen Einstieg in die Programmierung zu gewährleisten, wurde ebenfalls die jeweilige Größe der Entwickler-Community und die aktuelle Beliebtheit einzelner Software berücksichtig.

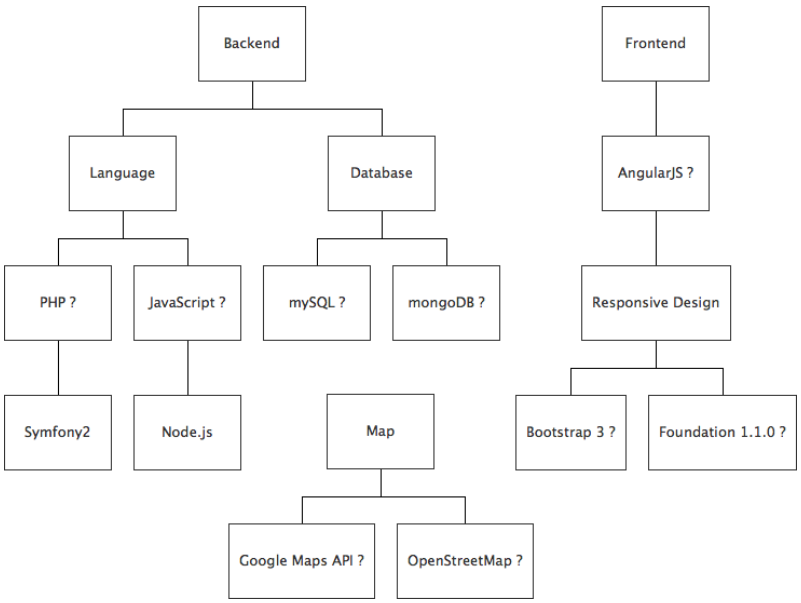


Abbildung 2: Planung des Software Stacks der Freizeitaktivitäts-Plattform

### 3.2.1 LAMP

Die Abkürzung LAMP steht für Linux, Apache, MySQL und PHP. Dieser Software Stack besteht in seiner Urform serverseitig aus dem Betriebssystem Linux, auf dem der Webserver Apache läuft. Als Datenbank wird MySQL, als Programmiersprache PHP genutzt. Dieser Software Stack ist schon seit etwa 1996 in der Entwicklerszene bekannt und komplett kostenlos und quelloffen. [8]

Durch stetige Weiterentwicklung der einzelnen Komponenten ist der LAMP Stack auch heute noch eines der bekanntesten und beliebtesten Software Stacks für Webanwendungen.

Der entscheidende Punkt, warum LAMP im Rahmen der Bachelorarbeit für die Entwicklung der Freizeitaktivitäts-Plattform in Frage kommt, ist Symfony 2: Das in PHP5 geschriebene *Web Application Framework* bietet zur schnellen und zeitgemäßen Entwicklung moderner Webanwendungen

viele nützliche, bereits fertige Komponenten an und ist aktuell bei Entwicklern äußerst beliebt. Weitere Komponenten können mit dem Paketmanager *Composer* hinzugefügt werden.

### 3.2.2 MEAN

Wie LAMP ist auch der Software Stack MEAN komplett kostenlos und quelloffen. MEAN steht für die Komponenten MongoDB, Express, AngularJS und Node.js. Diese Zusammensetzung erlaub es Entwicklern komplett in einer Programmiersprache zu arbeiten: JavaScript. [9] Darüber hinaus arbeiten alle Komponenten des MEAN Stacks, im Gegensatz zum LAMP Stack, welcher standartmäßig Linux voraussetzt, unabhängig von einem bestimmten Betriebssystem. Einzige Voraussetzung ist, dass JavaScript ausgeführt werden kann.

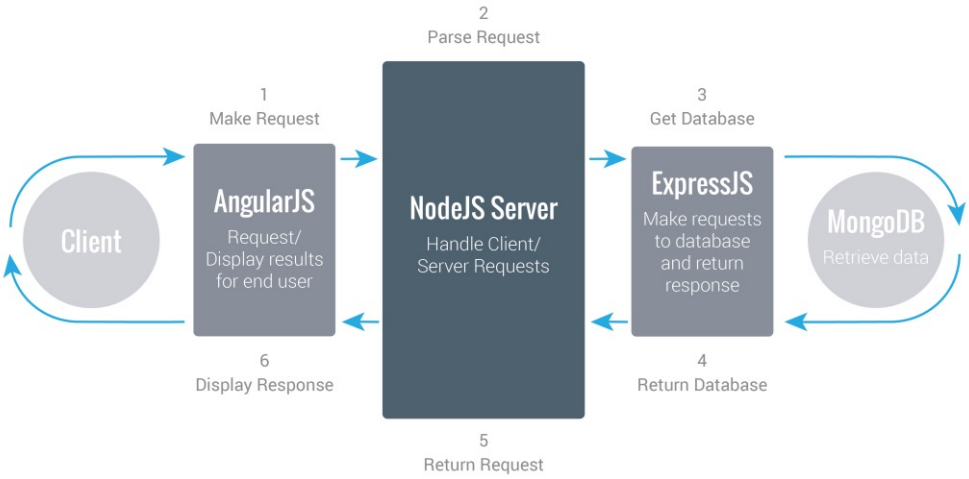


Abbildung 3: Funktionen der einzelnen Komponenten im MEAN Stack

**MongoDB**, eine sogenannt NoSQL-Datenbank, arbeitet im Gegensatz zu MySQL dokumentenbasiert. Daten werden im BSON-Format (Binary JSON) in Dokumente (zu vergleichen mit Datensätzen in MySQL) gespeichert, die wiederrum in Kollektionen (zu vergleichen mit Tabellen in MySQL) zusammengefasst werden. Dies bringt einige Vorteile mit sich: Daten werden nicht

in festen Tabellenstrukturen gespeichert, was Änderungen an der Datenbankstruktur und eine einfache Skalierbarkeit ermöglicht. Auch müssen Tabellen nicht mehr umständlich durch z.B. Joins zusammengeführt werden, denn Relationen werden mit in die Dokumente gespeichert oder bei Abfrage verknüpft. [10] Die gewünschten Datensätze werden ohne weitere Konvertierung im JSON-Format geliefert, was die Verarbeitung mit JavaScript sehr vereinfacht.

**Express** ist im MEAN Stack das Gegenstück zu Symfony 2, welches jedoch im LAMP Stack kein fester Bestandteil ist. Dieses *Node.js Web Application Framework* ist minimalistisch und flexibel aufgebaut, was für den Benutzer bedeutet, sich an wenige Vorgaben halten zu müssen. Für eine Webanwendung besonders wichtige Funktionalitäten wie z.B. Routing, Error Handling und Template Engines sind in Express bereits implementiert und erleichtern den Einstieg in die Programmierung. Durch das zusätzliche NPM Package *Express-Generator* lässt sich mit wenigen Befehlen eine neue Express Applikation mit vorgefertigtem Grundaufbau als Einstiegspunkt für die eigene Webanwendung erstellen.

**AngularJS** wird im MEAN Stack zur Realisierung des Clients verwendet. In der Webanwendung dient es primär zur Aufbereitung und Darstellung von Daten. Die Idee hinter dem von Google entwickelten AngularJS ist, statisches HTML durch die Darstellung dynamischer Inhalte so zu erweitern, dass es sich zur Umsetzung moderner Applikationen anbietet. Dies war zuvor nur durch zusätzlicher Nutzung von Bibliotheken wie jQuery möglich, indem man den Browser mittels DOM-Manipulationen dazu brachte, dynamische Inhalte darzustellen. AngularJS vereinfacht dies als komplettes Web Framework durch erweiterte Syntax und der Benutzung von Direktiven. So ist es z.B. mit wenig Aufwand möglich, Daten zwischen HTML und JavaScript bidirektional zu verbinden oder DOM-Elemente situationsbedingt darzustellen oder zu verstecken. [11] Ähnlich dem Webserver wird in AngularJS mit Views und Controllern gearbeitet.

**Node.js** ist das Kernstück des MEAN Stacks und dient als ereignisgesteuerter Webserver. Auf Basis der für Google Chrome entwickelten JavaScript-Laufzeitumgebung V8, ist Node.js in der Lage serverseitig JavaScript auszuführen. Dies machte Entwickler auf Node.js aufmerksam, denn so ist es möglich sowie client- als auch serverseitig mit JavaScript zu arbeiten. Die Trennung der Teilgebiete Backend und Frontend, die durch unterschiedliche Programmiersprachen und Fähigkeiten eines Entwicklers hervorgebracht wurde, kann so bei der Programmierung einer Webanwendung umgangen werden. Entwickler, die sowohl am Backend als auch am Frontend arbeiten, bezeichnen sich gebräuchlicher Weise als Full-Stack-Developer.

Neben Schlankheit und Effizienz der JavaScript-Laufzeitumgebung bringt Node.js, ähnlich wie Composer bei Symfony 2, den *Node Package Manager* (kurz NPM genannt) als einen weiteren großen Vorteil mit sich. NPM bietet dem Benutzer eine große Auswahl von quelloffenen Packages zur Wiederverwendung in der eigenen Anwendung. Installiert man ein Package, so wird es als Dependency mit Namen und Versionsnummer in der Datei *package.json* vermerkt. Dies hält die eigentliche Applikation schlank: Von NPM installierte Packages müssen, z.B. zur Versionskontrolle über GitHub, nicht mit hochgeladen werden. Bei einer Installation der Applikation auf einem anderen System muss lediglich der Befehl *npm install* ausgeführt werden um alle Dependencies in der gewählten Version zu installieren.

```
{
  "name": "test_project_1",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.13.2",
    "cookie-parser": "~1.3.5",
    "debug": "~2.2.0",
    "ejs": "~2.3.3",
    "express": "~4.13.1",
    "morgan": "~1.6.1",
    "serve-favicon": "~2.3.0"
  }
}
```

Abbildung 4: *package.json* eines mit *Express-Generator* erstellten Projekts

## 3.3 Auswahl des Software Stacks

Der folgende Unterabschnitt behandelt die Wahl eines der in Unteranschnitt 3.2 vorgestellten Software Stacks und die Zuhilfenahme weiterer Komponenten, die zur Implementierung der Freizeitaktivitäts-Plattform benötigt werden.

### 3.3.1 LAMP vs. MEAN

Erste Tests zur Umsetzung der Webanwendung wurden mit dem LAMP Stack und Symfony 2 durchgeführt. Für lokale Tests wurde unter OS X die Software MAMP in der Version 3.5 benutzt. Durch diese Software werden alle Komponenten des LAMP-Stacks bereitgestellt, wobei das Betriebssystem Linux durch OS X ersetzt wird. Diese Tests brachten jedoch schnell die Erkenntnis, dass das Framework Symfony 2 für die Entwicklung der Freizeitaktivitäts-Plattform im Rahmen der Bachelorarbeit ungeeignet ist. Symfony 2 ist zu undurchsichtig und komplex strukturiert. Es werden zu viele Vorgaben seitens des Frameworks gemacht und zusätzliche Packages, wie z.B. das *FOSUserBundle* zur Implementierung von Benutzerkonten, sind oft vom Funktionsumfang her zu groß. Während der Testphase konnten nur wenige Teilergebnisse erzielt werden und die Gewöhnung an das Arbeiten mit PHP fiel schwer.

**LAMP Stack**

| Pro | Contra |
|---|---|
| • Zur Entwicklung von Webanwendungen altbewährt<br>• Tutorials für jeden Anwendungszweck vorhanden<br>• Komponenten arbeiten zuverlässig zusammen | • Festgefahrener Aufbau<br>• Einstieg fällt schwer<br>• Tutorials setzen oft Vorwissen voraus<br>• Tutorials oft nicht aktuell<br>• Funktionsumfang von Packages zu groß<br>• Datenbankstruktur nicht flexibel<br>• Das Arbeiten mit PHP fällt schwer |

Tabelle 1: LAMP Test Pro/Contra

Das genaue Gegenteil zeigten erste Tests mit dem MEAN Stack. Getestet wurde Node.js in der Version 4.2.2 mit dem durch NPM heruntergeladen Package *Express-Generator*. Mit diesem Package kann in wenigen Schritten ein neues Node.js Projekt generiert werden, welches bereits Express als Framework enthält und als Basis einer neuen Webanwendung sofort lauffähig ist. Der Server wird mit dem NPM Package *Nodemon* gestartet. Nodemon überwacht Dateiänderungen innerhalb des Node.js Projekts und automatisiert den Neustart des Node.js Servers. So ist es möglich, neue Implementierungen direkt und ohne manuellen Neustart im Browser zu testen. Als Datenbank wird MongoDB Version 3.0.7 lokal installiert und ausgeführt. Mit dem NPM Package *Mongoose* lässt sich die lokale Datenbank mit wenig Aufwand in das Node.js Projekt implementieren. Mongoose ist eine Object Data Modeling (ODM) Library und dient als Bindeglied zwischen MongoDB und Node.js. Mit diesen Voraussetzungen konnten während der Testphase schnell erste Ergebnisse erzielt werden. AngularJS wurde dabei noch nicht genutzt. Stattdessen wurden Websites mit Hilfe der Templating Engine *EJS* (Embedded JavaScript) serverseitig generiert.

**MEAN Stack**

| Pro | Contra |
|---|---|
| • Leichter Start durch Express-Generator<br>• JavaScript, JSON<br>• Aktuell für Webanwendungen beliebt<br>• Gute Dokumentationen mit Anwendungsbeispielen<br>• Sehr aktuelle Tutorials<br>• Unabhängig von Betriebssystemen<br>• Datenbankstruktur kann einfach geändert werden<br>• Node.js sehr performant<br>• Aufbau kann frei gewählt werden | • Tutorials schwerer zu finden<br>• Fehler oft nicht eindeutig<br>• Oft zu viele Freiheiten |

Tabelle 2: MEAN Test Pro/Contra

Nach Beendigung der Testphase wurde MEAN als Software Stacks für die Freizeitaktivitäts-Plattform gewählt. Bei der Einarbeitung durch diverse Tests wurde sofort festgestellt, dass die Benutzung des MEAN Stacks die wesentlich modernere Methode zur Umsetzung einer Webanwendung ist. Die Arbeit mit dem MEAN Stack hinterließ sofort, im Gegensatz zur Benutzung des LAMP Stacks, einen positiven Ersteindruck. Der Einstieg wurde durch gute Dokumentationen und einem frei wählbaren Aufbau deutlich erleichtert. Dazu, durch die aktuelle Beliebtheit von Node.js, waren Tutorials zu einzelnen Anwendungsgebieten stets aktuell. Im Gegensatz zu PHP fiel auch das Arbeiten mit JavaScript wesentlich leichter. Funktionalitäten konnten so schneller implementiert werden.

Im späteren Verlauf der Implementierung kam zur Umsetzung eines Clients ebenfalls AngularJS zum Einsatz.

### 3.3.2 Bootstrap

Bootstrap gilt als heutzutage populärstes CSS Framework und wird von Twitter entwickelt. Es stellt verschiedene vorgefertigte Komponenten zur Gestaltung einer Website bereit, welche HTML Elementen als Klasse zugewiesen werden. Auf diese Weise kann schnell ein erstes Design erstellt werden. Das Besondere an Bootstrap ist, dass alle Komponenten mit dem Ansatz „mobile first“ entwickelt wurden: Sie passen sich automatisch der Größe des Displays verschiedener Endgeräte an. Wie viel Platz eine Komponente dabei einnimmt und ab welcher Displaygröße eine Umordnung aller Komponenten stattfinden soll, wird durch Bootstraps Grid-System festgelegt. Dabei wird die Breite einer Website in zwölf Spalten unterteilt. Eine Komponente kann beliebig viele dieser Spalten belegen. So nimmt z.B. eine Komponente, der sechs Spalten zugewiesen werden, fünfzig Prozent der Breite einer Website ein.

Bootstrap stellt damit eine geeignete Ergänzung zu AngularJS im Client dar und wurde für das Design des Prototyps der Freizeitaktivitäts-Plattform verwendet.

### 3.3.3 Google Maps API

Google Maps, der wohl bekannteste Online-Kartendienst, bietet für Entwickler verschiedene, jeweils gut dokumentierte APIs an. Diese unterscheiden sich in Programmiersprache und Funktionsumfang. Von einer einzelnen eingebetteten Karte auf einer Website bis hin zu komplexen ortsbasierten Anwendungen, kann so Google Maps für verschiedenste Zwecke im eigenen Projekt benutzt werden.

Zur Implementierung von Karten und der Visualisierung von ortsbasierten Daten wurde die Google Maps JavaScript API und die Google Places API verwendet. Zusätzlich wurde das für AngularJS entwickelte Directive *Ng-Map* verwendet. Dieses quelloffene Directive dient zur erleichterten Benutzung der Google Maps APIs unter AngularJS und bietet verschiedene Anwendungsbeispiele an.

# 4 Implementierung

Im folgenden Abschnitt wird ein Prototyp einer Webanwendung realisiert, welcher der Idee einer Freizeitaktivitäts-Plattform aus Abschnitt 1 entspricht und möglichst viele aus Unterabschnitt 2.1 resultierende Anforderungen erfüllt. Der Prototyp der Webanwendung wird auf den MEAN-Stack aufgebaut, für das sich in Unterabschnitt 3.3.1 entschieden wurde. Zusätzlich werden Bootstrap und Google Maps APIs verwendet. Wie in Unterabschnitt 3.1 beschrieben, teilt sich die Webanwendung in einen Server und einen Client. Für die Strukturierung der Webanwendung wurde nach dem MVC-Modell (Model-View-Controller) gearbeitet. Dabei wird versucht, Models, Views und Controller getrennt voneinander zu behandeln. Dies bietet den Vorteil, dass Änderungen oder spätere Ergänzungen leichter umzusetzen sind.

- Models kümmern sich um den Aufbau von Daten
- Views legen die Darstellung von Daten fest
- Controller verknüpfen Models und Views durch Programmlogik

## 4.1 Server

Die Aufgabe des Servers ist es den Benutzer zu authentifizieren, seine Anfragen zu bearbeiten und schließlich die angeforderten Daten an den Client zu senden. Dies wird nach dem Prinzip einer RESTful API realisiert: Der Client kann den Server durch HTTP-Requests dazu auffordern Daten bereitzustellen (GET), zu speichern (POST), zu editieren (PUT) oder zu löschen (DELETE).

**Models**

Damit die Webanwendung später in der Lage ist Daten zu senden, müssen diese zuerst in der lokalen MongoDB-Datenbank gespeichert werden. Dafür wird ein User-Model und ein Event-Model mit Hilfe des NPM-Package *Mongoose* implementiert. Mongoose setzt voraus ein Schema zu definieren, in dem die spätere Form der Dokumente in Collections festgesetzt wird. Diese

Form wird im Schema durch Key-Value-Paaren bestimmt. Ein Key speichert dabei immer eine Value eines Datentyps. Neben den im Schema definierten Key-Value-Paaren besitzt jedes Dokument in der MongoDB den Key *_id*, in dem automatisch eine eindeutige ID vergeben wird. Diese von MongoDB generierten IDs werden in der Webanwendung zur Identifizierung von User und Events benutzt. In einem Schema können ebenfalls Datensätze miteinander verknüpft werden. Der Key *participants* des Event-Models fasst z.B. als Value ein Array aus User-IDs mit Referenz auf das User-Model. So ist es später möglich, Events nur mit User-IDs, mit allen Daten oder mit nur ausgewählten Daten der teilnehmenden User als Embedded Document ausgeben zu lassen.

Das Event-Model wurde mit zwei weiteren Besonderheiten implementiert: Es enthält ein zweites Schema, welches die Form von Kommentaren in einem Event als Embedded Document festsetzt. Dadurch kann auf ein eigenständiges Kommentar-Model und eine Referenz im Event-Model verzichtet werden. Kommentare sind damit Teil des Event-Models und existieren nur dort. Als zweite Besonderheit wird der Key *loc*, welcher als Value einen Array vom Typ Number speichert und für Latitude und Longitude eines Standorts vorgesehen ist, mit einem 2dsphere Index versehen. Dadurch können raumbezogene Queries auf Events angewendet werden. Durch den 2dsphere Index werden Standorte von Events auf eine Kugel übertragen und mittels Geospatial Query Operators miteinander verglichen. Für die Freizeitaktivitäts-Plattform wird dies später mit dem Operator *near* zur Implementierung der Umkreissuche verwendet.

Diese Schemas werden anschließend als Model exportiert und können so an anderen Stellen, wie z.B. in Controllern, zur Benutzung eingebunden werden.

**Authentifizierung**

Die Authentifizierung von Benutzern ist ein existenzieller Teil der Webanwendung. Die Benutzung der Freizeitaktivitäts-Plattform sieht voraus, dass

sich Benutzer zuvor registrieren oder einloggen müssen, um von anderen Benutzern erstellte Inhalte zu sehen. Dies bietet als eine erste Hürde einen Schutz vor Missbrauch von Benutzerprofilen und Events. Um diesen Schutz zu verstärken, wurde im Prototyp der Freizeitaktivitäts-Plattform bewusst nur eine Registrierung mit einem existierenden Facebook-Profil über die Facebook API implementiert. Dies bringt folgende Vorteile:

- Der Registrierungsprozess wird erleichtert. Alle wichtigen Benutzerdaten werden bei Zustimmung von Facebook bereitgestellt.
- Benutzer der Freizeitaktivitäts-Plattform handeln nicht komplett anonym, da Benutzerprofil und Facebook-Profil miteinander verknüpft sind.
- Das erstellen von Fake-Profilen und eine mehrfache Registrierung wird erschwert.
- Benutzer können sich so vor einem Treffen mit anderen Benutzern durch deren öffentliches Facebook-Profil vergewissern, dass es sich um eine reale Person handelt.

Zur Authentifizierung müssen Benutzer auf einer Startseite einen Button anklicken. Falls der Benutzer noch nicht registriert ist, leitet die Webanwendung den Benutzer zu Facebook weiter. Er wird dort um Einverständnis für den Zugriff auf sein öffentliches Facebook-Profil und wahlweise seine E-Mail-Adresse gebeten. Stimmt der Benutzer zu, wird er als neuer User in der Datenbank erfasst und als authentifizierter User sofort auf die Webanwendung mit vollem Zugriff weitergeleitet. Lehnt er die Freigabe seiner Daten wiederum ab, wird er auf die Startseite zurückgeführt und erhält keinen Zugriff. Bei der Registrierung werden Facebook-Profil ID, Vorname, Nachname, Geschlecht und E-Mail von der Facebook API in den Datensatz des Benutzers gespeichert. Darüber hinaus wird ein erster Username als String aus Vorname und Initiale des Nachnamens generiert und gespeichert. Außerdem werden, mit Hilfe der Facebook-Profil ID, Links zum Facebook-Profil und Profilbild als Strings generiert und ebenfalls gespeichert. Falls der Benutzer schon in der Datenbank existiert, werden nochmals Vorname, Nachname, Geschlecht,

E-Mail und der Link zum aktuellen Profilbild erneut von der Facebook API abgefragt und aktualisiert. So kann sichergestellt werden, dass alle relevanten Benutzerdaten, die von der Facebook API kommen und auf Facebook durch den Benutzer änderbar sind, immer in aktueller Form vorliegen. Ob sich ein Benutzer neu registriert oder als bereits registrierter Benutzer erneut anmeldet, wird durch eine Query an die Datenbank abgefragt, welche die in der Datenbank gespeicherte Facebook ID von bereits registrierten Benutzern mit der von der Facebook API gelieferten Profil ID vergleicht.

Die Authentifizierung von Benutzern über deren Facebook-Profile wird mit dem NPM Package *Passport* realisiert. Passport ist eine für Node.js entwickelte Middleware zur Authentifizierung von Benutzern und bietet dafür über 300 Strategien an. Eine dieser Strategien ist die Authentifizierung von Benutzer durch Facebook, welche zusätzlich als NPM Package *Passport-Facebook* installiert wird. Passport-Facebook authentifiziert Benutzer über das Facebook-Profil und OAuth 2.0 Tokens. Dazu muss die Webanwendung zuvor bei Facebook als App registriert werden. Nach der Registrierung erhält man eine der Webanwendung zugewiesene App ID und ein App Secret, welche für die Passport-Authentifizierungsstrategie benötigt werden und in der Webanwendung implementiert sein müssen. Außerdem muss eine Callback URL definiert werden, an die Facebook nach erfolgreicher Authentifizierung einen Access Token, einen Refresh Token und die Daten des Facebook-Profils an den Server der Webanwendung zurückschickt. Dies wird serverseitig in einer Session gespeichert. Der Benutzer gilt als authentifiziert und erhält Zugriff zur Webanwendung, bis dieser sich wieder abmeldet oder die Session verfällt.

**Views**

Serverseitig werden für den Prototyp der Freizeitaktivitäts-Plattform nur drei Views benutzt: Die erste View *error.ejs* dient der Darstellung von Fehlermeldungen im Browser und wird beim Erstellen eines neuen Projekts durch Express-Generator automatisch angelegt. Hier werden dem Benutzer z.B.

404-Fehler angezeigt. Die zweite View *index.ejs* dient als Startseite. Hier wird der Benutzer begrüßt und die Features der Webanwendung in kurzen Texten beschrieben, um die Idee hinter der Freizeitaktivitäts-Plattform zu verdeutlichen. Außerdem befindet sich hier der Button zur Authentifizierung über Facebook, welcher als Kernelement platziert ist. Die dritte View *home.ejs* dient als Startpunkt des Clients. In dieser View befindet sich die Navigationsleiste der Webanwendung. Unter der Navigationsleiste wird in einem Container das AngularJS Directive *ngView* eingebunden. Dieses Directive bestimmt, an welcher Stelle die AngularJS-Views geladen werden. Ab diesem Punkt übernimmt AngularJS die clientseitige Darstellung der Daten. Bei den drei Views handelt es sich um EJS-Dateien. Durch die Templating Engine Embedded JavaScript kann dynamischer Inhalt in HTML direkt eingebettet werden. Da die Webanwendung clientseitig AngularJS zur Darstellung von dynamischen Inhalten benutzt, wird EJS nur wenig verwendet. In den Views *index.ejs* und *home.ejs* wird der Titel der Website im Header mit EJS generiert. Außerdem wird der API-Key zur Einbindung von Google Maps als Umgebungsvariable an die View *home.ejs* weitergegeben.

**Controller**

Die Controller der Webanwendung verbinden Models und Views miteinander und legen die Endpunkte des Servers durch ein Routing fest. In Routen werden primär Queries an die Datenbank erstellt, ausgeführt und deren Ergebnisse als Response gesendet. Es werden vier Controller implementiert, welche die Funktionalität unterschiedlicher Teile der Webanwendung bestimmen: Index, Authentifizierung, User und Event.

Der Controller *index.js* dient in erster Linie zur Organisation weiterer Controller. Nur der Index-Controller ist direkt in die Node.js-Applikation eingebunden. Alle anderen Controller sind wiederum im Index-Controller eingebunden. Dies erleichtert das nachträgliche Hinzufügen neuer Controller und ermöglicht eine saubere Strukturierung aller Endpunkte. Des Weiteren wird hier das Rendering der Views *index.ejs* und *home.ejs* als Einstiegspunkt

in die Webanwendung veranlasst. Besucht ein Benutzer die Startseite der Web-anwendung, wird hier überprüft, ob dieser schon authentifiziert ist oder nicht. Ist er noch nicht authentifiziert, so wird die View *index.ejs* als Startseite gerendert. Hat sich der Benutzer jedoch schon zuvor eingeloggt und es existiert bereits eine serverseitige Session für diesen, wird die View *home.ejs* gerendert, in der als Startpunkt eine Liste der Events dargestellt wird.

Der Controller *auth.js* stellt alle für die Middleware Passport benötigten Endpunkte bereit. Klickt der Benutzer auf den Login-Button der Startseite, wird durch diesen Controller die Funktion *passport.authenticate* ausgeführt. Danach übernimmt eine Callback-Route, welche die Daten von Facebook empfängt und entweder zur Success-Route oder zur Failure-Route weiterleitet. Hat sich der Benutzer erfolgreich über Facebook eingeloggt, wird er auf die zuvor beschriebene Route des Index-Controllers geleitet, welche bei erfolgreicher Authentifizierung die View *home.ejs* rendert. Ist die Authentifizierung nicht erfolgreich abgeschlossen, wird über die Failure-Route ein Fehler an den Benutzer ausgegeben. Außerdem enthält der Auth-Controller eine Logout-Route, die den Benutzer über die Passport-Funktion *logout* serverseitig abmeldet und als nicht authentifizierten Benutzer zurück auf die Startseite leitet.

Der Controller *user.js* liefert und verarbeitet alle Daten in Bezug auf Benutzer der Freizeitaktivitäts-Plattform. Die Endpunkte dieses Controllers sind durch die Middleware *isLoggedIn* geschützt: Vor allen Routen in diesem Controller wird die Passoprt-Funktion *isAuthenticated* aufgerufen, welche überprüft, ob ein Benutzer bereits auf dem Server authentifiziert ist. Ist dies der Fall, so wird der nächste Arbeitsschritt, z.B. das Abrufen von Profildaten, für den Benutzer zugelassen. Ist der Benutzer jedoch nicht authentifiziert und versucht trotzdem einen Endpunkt des Controllers zu nutzen, wird er auf die Startseite umgeleitet. Der User-Controller stellt eine Route zum abrufen und bearbeiten des eigenen Benutzerprofils bereit. Dies wird mit den HTTP-Requests GET und PUT realisiert. Der GET-Request liefert dem authentifizierten Benutzer seine gesamten Benutzerdaten als JSON. Durch den PUT-Request kann der

Benutzer die Value seines Keys *username* und *about* ändern. Bei Änderungen der Benutzerdaten wird automatisch auch die Value vom Key *updatedAt* auf das aktuelle Datum in Form eines Zeitstempels geändert. So kann nachvollzogen werden, wann genau der Benutzer sein Profil aktualisiert hat. Treten keine Fehler beim Speichern auf, werden ebenfalls die Benutzerdaten in aktualisierter Form als JSON zurückgegeben. Des Weiteren beinhaltet der User-Controller eine Route zur Ausgabe von Daten anderer Benutzer. Durch den Parameter *id* und der Mongoose-Funktion *findById* wird nach einem Benutzer anhand der Benutzer-ID gesucht. Wird ein Benutzer mit passender ID gefunden, wird eine JSON mit Benutzer-ID, Username, About-Text, Geschlecht, Link zum Facebook-Profilbild und Link zum Facebook-Profil zurückgegeben. Falls ein Fehler auftritt oder kein Benutzer mit passender ID gefunden wird, wird ein Fehler ausgegeben.

Der Controller *event.js* übernimmt das serverseitige Routing für Events und ist der umfangreichste Controller der Webanwendung. Alle Routen des Controllers werden, wie beim User-Controller, durch die Middleware *isLoggedIn* vor unbefugtem Zugriff geschützt. Der GET-Request zum Abrufen der Event-Liste liefert dem Benutzer alle noch bevorstehenden Events. Wird dieser Request durch den Client mit zusätzlichen Parametern für Distanz, Latitude und Longitude ausgeführt, so werden nur Events innerhalb eines gewählten Umkreises an einem gewählten Standort berücksichtigt. Dabei wird die Distanz als Radius eines Umkreises in Meter übergeben und innerhalb der Query in Kilometer umgerechnet. Der Standort wird als Punkt durch Latitude und Longitude übergeben. Durch die Mongoose-Funktion *near* kann so eine Umkreissuche auf Event-Standorte, die in Events als Key *loc* gespeichert und mittels Geospatial Index im Event-Model indexiert sind, durchgeführt werden. Ein POST-Request dieser Route dient zum Hinzufügen eines neuen Events. Hierfür werden Titel, Beschreibung, User-ID des Erstellers, Datum mit Uhrzeit und die Koordinaten eines Standorts benötigt. Diese Daten werden über ein HTML-Formular vom Client geliefert. Sind alle Daten in passender Form vorhanden, wird das neue Event in der Datenbank angelegt

und danach als JSON zurückgegeben. Zwei weitere Routen mit GET-Request sind für die Ausgabe von Event-Listen speziell für das Benutzerprofil vorgesehen:

- Alle Events, die der Benutzer erstellt hat.
- Alle Events, an denen der Benutzer teilnimmt oder teilgenommen hat.

Des Weiteren kann, wie ein Benutzer im User-Controller, ein einzelnes Event mittels Event-ID als Parameter im GET-Request angefordert werden. Values der Keys *createdBy*, *comments.createdBy* und *participants* eines Events werden dabei mit Benutzer-ID, Username und Link zum Profilbild aus dem User-Model angereichert und mit den Event-Daten als JSON zurückgegeben. Durch ein PUT-Request kann ein Event durch den Ersteller komplett editiert werden. Hinzu kommt ein DELETE-Request zum Löschen eines Events durch den Ersteller. Drei weitere Routen dienen der Implementierung von Funktionen innerhalb eines Events:

- Kommentar in einem Event hinterlassen durch PUT-Request
- Dem Event beitreten durch PUT-Request
- Aus dem Event austreten durch PUT-Request

Zusammen bilden diese vier Controller mit ihrem Routing die RESTful API der Freizeitaktivitäts-Plattform. Die Endpunkte des Servers sind somit bereit, von einem Client durch HTTP-Requests benutzt zu werden. Weitere Funktionen können, durch die Aufteilung in einzelne Controller, durch die Implementierung weiterer Routen leicht hinzugefügt werden.

| Controller | Methode | Route |
|---|---|---|
| index.js | GET | / |
| auth.js | GET | /auth/facebook |
| | GET | /auth/facebook/callback |
| | GET | /auth/success |
| | GET | /auth/failure |
| user.js | GET | /api/user/profile |
| | PUT | /api/user/profile |
| | GET | /api/user/<USER._ID> |
| event.js | GET | /api/event/ |
| | POST | /api/event/ |
| | GET | /api/event/user/created |
| | GET | /api/event/user/participated |
| | GET | /api/event/<EVENT._ID> |
| | PUT | /api/event/<EVENT._ID> |
| | DELETE | /api/event/<EVENT._ID> |
| | PUT | /api/event/<EVENT._ID>/comment |
| | PUT | /api/event/<EVENT._ID>/join |
| | PUT | /api/event/<EVENT._ID>/leave |

Tabelle 3: Liste aller Endpunkte der Webanwendung

## 4.2 Client

Für die Realisierung eines prototypischen Clients der Freizeitaktivitäts-Plattform wurde AngularJS in der Version 1.5.0 benutzt. Nachdem der Benutzer sich erfolgreich über das Facebook-Profil authentifiziert hat, gelangt er zum Startpunkt der Angular App. Ab hier übernimmt AngularJS die Darstellung des Clients als Single-Page Application. Zur Installation und Verwaltung aller Packages, die vom Client benötigt werden, wird der Paketmanager *Bower*

benutzt. Bower funktioniert im Prinzip genau wie der Paketmanager *NPM* des Node.js-Servers. NPM verwaltet ausschließlich alle serverseitig benötigten Packages, während Bower die clientseitig benötigten Packages. Die über Bower installierten Packages werden in der Datei *bower.json,* ähnlich der *package.json* von NPM, als Dependency gespeichert. Folgende Packages werden zur Implementierung des Clients benötigt:

- Angular als Framework des Clients.
- Angular-Route für die Realisierung des clientseitigen Routings als Single-Page Application.
- NgMap als Google Maps Directive für AngularJS.
- Bootstrap zur Gestaltung des Clients. Durch Bootstrap wird ebenfalls jQuery installiert, welches als Dependency für Bootstrap benötigt wird.

In Angular wird ebenfalls mit Routing, Controllern und Views gearbeitet. Jedoch handelt es sich bei den Views nicht um komplette Views, sondern um sogenannte Partials. Ein Partial ist vielmehr ein Teilstück eines Views. Im Falle der Freizeitaktivitäts-Plattform wird einmalig die View *home.ejs*, welche die Navigationsleiste enthält und die Single-Page Application startet, vom Server gerendert und durch AngularJS mit einem Partial ergänzt. Dieses legt fest, welcher Inhalt der Webanwendung angezeigt wird. Durch das Package *Angular-Route* wird bestimmt, wann welches Partial angezeigt wird und welcher Angular-Controller dazu benutzt wird.

Nachdem sich ein Benutzer serverseitig erfolgreich authentifiziert hat, startet die Angular App und führt einen GET-Request auf die Route */api/user/profile* aus, um die Benutzerdaten des authentifizierten Benutzers in der Angular-App zu erhalten. Das erhaltene Benutzerprofil wird in der Variable *$rootScope.currentUser* gespeichert, um den Benutzer clientseitig identifizieren zu können. Außerdem wird die Variable *$rootScope.authenticated* auf true gesetzt. Der Benutzer gilt jetzt im Client als vollständig angemeldet. Ne-

ben den zwei Variablen existiert eine Logout-Funktion im *$rootScope*, welche bei Aufruf die Benutzerdaten in der Variable *currentUser* durch ein leeres Objekt überschreibt und die Variable *authenticated* zurück auf false setzt. Variablen und Funktionen im *$rootScope* können innerhalb der Angular-App überall benutzt werden. Nachdem die Benutzerdaten erfolgreich abgerufen und gespeichert wurden, erscheint in der Navigationsleiste das Profilbild des Benutzers. Ein Klick auf das Profilbild öffnet ein Dropdown-Menü mit den Unterpunkten *Profile* und *Logout*. Der Unterpunkt *Logout* sendet einen GET-Request zur Route *auth/logout* des Servers und führt außerdem clientseitig die Logout-Funktion aus. Damit ist der Benutzer gleichzeitig client- und serverseitig abgemeldet. Der Unterpunkt *Profile* führt den Benutzer zu seinem Benutzerprofil. Außerdem befindet sich in der Navigationsleiste der Punkt *+Add* zum Erstellen eines neuen Events.

Als erstes Partial wird die Datei *eventList.html* geladen.

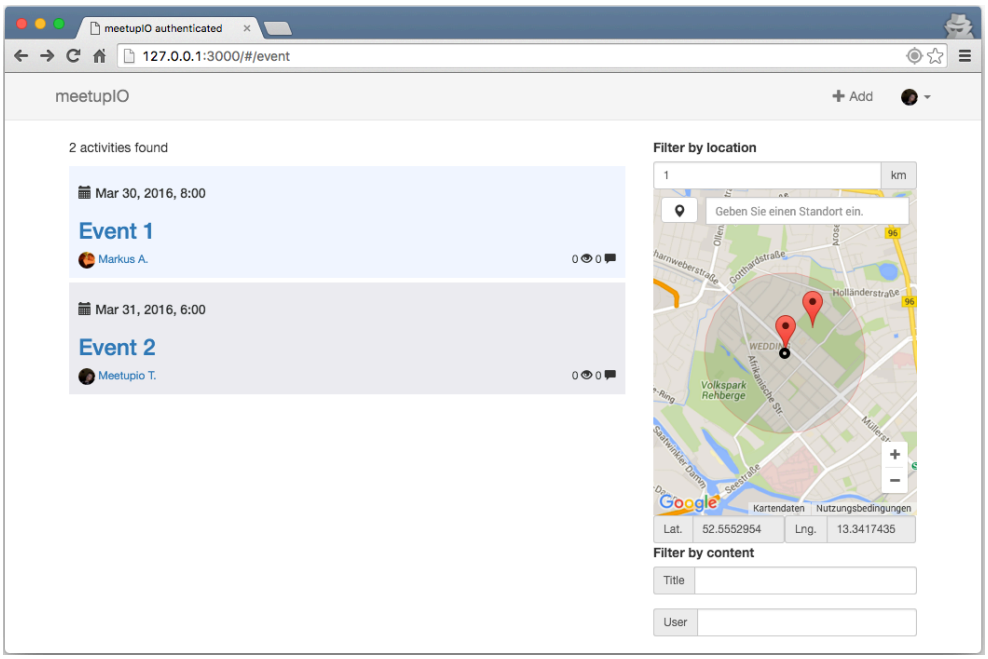


Abbildung 5: Screenshot der Webanwendung mit Partial *eventList.html*

Dieses Partial benutzt den Angular-Controller *eventListController,* welcher einen GET-Request auf die Route */api/event/* ausführt, um eine Liste von

Events vom Server zu erhalten. Mit dem AngularJS-Directive *ngRepeat* wird die Liste der Events im Client dargestellt. Als Standard werden dem Benutzer zuerst alle Events in der Datenbank angezeigt, deren Startdatum größergleich dem aktuellen Datum ist. Diese Liste ist dabei ebenfalls nach dem Startdatum der Events sortiert, damit zeitnahe Events immer am Anfang der Liste stehen. Wird die Umkreissuche unter dem Punkt *Filter by location* mit einem Standort und einer Distanz verwendet, so wird der GET-Request auf die Route */api/event/* erneut mit den Parametern für Latitude, Longitude und Distanz ausgeführt. Die Liste der Events wird nach Erhalt der Daten automatisch neu generiert. Der GET-Request wird automatisch nach einem Delay von 500 Millisekunden ausgeführt, sobald der Radius der Umkreissuche geändert wurde. Die Karte der Umkreissuche ist im Partial durch das Directive *Ng-Map* eingebunden und mit Funktionen des Angular-Controllers *mapController* verknüpft. Aufbau und Funktionalität dieses Controllers werden im späteren Verlauf genauer erläutert. Auf der Karte der Umkreissuche werden folgende Elemente dargestellt:

- Ein Button (Custom-Control) zur Abfrage des aktuellen Standorts des Benutzers.
- Ein Input-Feld (Custom Control) zur Suche von Standorten über die Google Places API.
- Marker zur Darstellung von Standorten aller Events aus der aktuell angezeigten Event-Liste.
- Ein Marker zur Darstellung der aktuell gewählten Position auf der Karte. Dieser kann durch Drag and Drop frei bewegt werden.
- Ein Shape vom Typ Circle, der als Mittelpunkt die Position des Markers der aktuell gewählten Position hat und dessen Radius durch die gewählte Distanz der Umkreissuche bestimmt wird.

Unter dem Punkt *Filter by content* kann die Liste außerdem nach Inhalten im Event-Titel und Event-Erstellern gefiltert werden. Bei Benutzung dieser Filter sind die Ergebnisse sofort in der Liste der Events sichtbar. Über die Liste

der Events kann der Benutzer zu einzelnen Events oder Benutzerprofilen der Event-Ersteller gelangen.

Klickt der Benutzer auf einen Titel eines Events aus der Event-Liste, wird ein GET-Request an die Route */api/event/<EVENT._ID>* gesendet. Die empfangenen Daten werden im Partial *event.html* geladen. Diesem Partial ist der Angular-Controller *eventController* zugewiesen. Der Benutzer sieht hier alle Daten des Events und kann diesem, sofern er nicht der Ersteller des Events ist, beitreten. Tritt der Benutzer dem Event bei, wird ein PUT-Request an die Route */api/event/<EVENT._ID>/join* gesendet. Der Benutzer ist dann als Teilnehmer im Event gespeichert. Mit de AngularJS-Directives *ngShow* und *ngHide* werden die Buttons *Join* und *Leave* im Event ein- und ausgeblendet, je nachdem ob der Benutzer bereits ein Teilnehmer des Events ist. Als beigetretener Benutzer werden bereits vorhandene Kommentare des Events angezeigt. Hinterlässt der Benutzer einen neuen Kommentar im dafür vorgesehenen Input-Feld, wird dies über einen PUT-Request an die Route */api/event/<EVENT._ID>/comment* an den Server gesendet und im Event gespeichert.

Über den Menüpunkt *+Add* gelangt der Benutzer zum Partial *eventAdd.html.* Hier können alle für ein neues Event benötigten Daten in einem Formular eingegeben und der Standort des Events auf einer Karte gewählt werden. Beim Versenden des Formulars wird die Funktion *$scope.post* des Angular-Controller *eventListController* aufgerufen, welche die Formulardaten durch einen POST-Request an die Route */api/event/* des Servers schickt. Die Karte zum Bestimmen des Event-Standorts wird, wie auch die Karte der Umkreissuche im Partial *eventList.html*, über Funktionen des Angular-Controller *mapController* gesteuert. Beim initialisieren der Karte wird die Funktion *getCurrentPos* ausgeführt. Diese Funktion erfragt den aktuellen Standort des Benutzers über den Browser ab. Beim ersten Erfragen des Standorts wird der Benutzer gefragt, ob er dies zulassen möchte. Nachdem die Daten des aktuellen Standorts erhalten wurden, springt die Karte zu den erhaltenen Koordinaten und ein Marker wird gesetzt. Die aktuellen Koordinaten des Markers

sind im Formular in den nicht-editierbaren Feldern *Lat.* und *Lng.* ersichtlich. Der gesetzte Marker kann vom Benutzer frei per Drag and Drop bewegt werden. Beim Drop-Event werden die aktuellen Koordinaten des Markers durch die Funktion *getMarkerPos* erfragt und in die Formularfelder eingetragen. Über den Custom-Control Button in der Karte kann der aktuelle Standort des Benutzers erneut erfragt werden. Dafür wird, nach Klick auf den Button, die Funktion *getCurrentPos* erneut ausgeführt. Das Custom-Control Input-Feld dient der Suche von Standorten über die Google Places API. Dieses Feld verfügt über eine automatische Vervollständigung durch die Google Places API, was dem Benutzer die Suche nach einem bestimmten Standort erleichtert. Ändert sich der Inhalt des Input-Feldes, wird die Funktion *getPlacePos* ausgeführt. Diese Funktion erfragt die Koordinaten des eingegebenen Standorts über die Google Places API und richtet die Karte und den Marker dorthin aus. Dem Benutzer ist es also möglich, seinen aktuellen Standort erneut zu erfragen, nach einem Standort durch die Google Places API zu suchen und den Marker, der den Standort des Events bestimmt, frei zu verschieben. So kann der Standort eines Events komfortabel und punktgenau ausgewählt werden.

Über das Untermenü der Navigationsleiste gelangt der Benutzer zu seinem Benutzerprofil. Dies wird durch das Partial *userProfile.html* dargestellt und durch den Angular-Controller *userProfileController* gesteuert. Da die Benutzerdaten des Benutzers beim Start der Angular-App in der Variable *$rootScope.currentUser* gespeichert wurden, muss der *userProfileController* beim Aufruf des Partials *userProfile.html* nur zwei GET-Requests auf die Routen */api/event/user/created* und */api/event/user/participated* ausführen. Die erhaltenen Daten werden als eine Liste von Events, die der Benutzer erstellt hat, und eine Liste von Events, an denen der Benutzer teilnimmt oder teilgenommen hat, im Benutzerprofil dargestellt. Damit ist dem Benutzer eine Übersicht geboten, in welchen Events sein Benutzerprofil auftaucht.

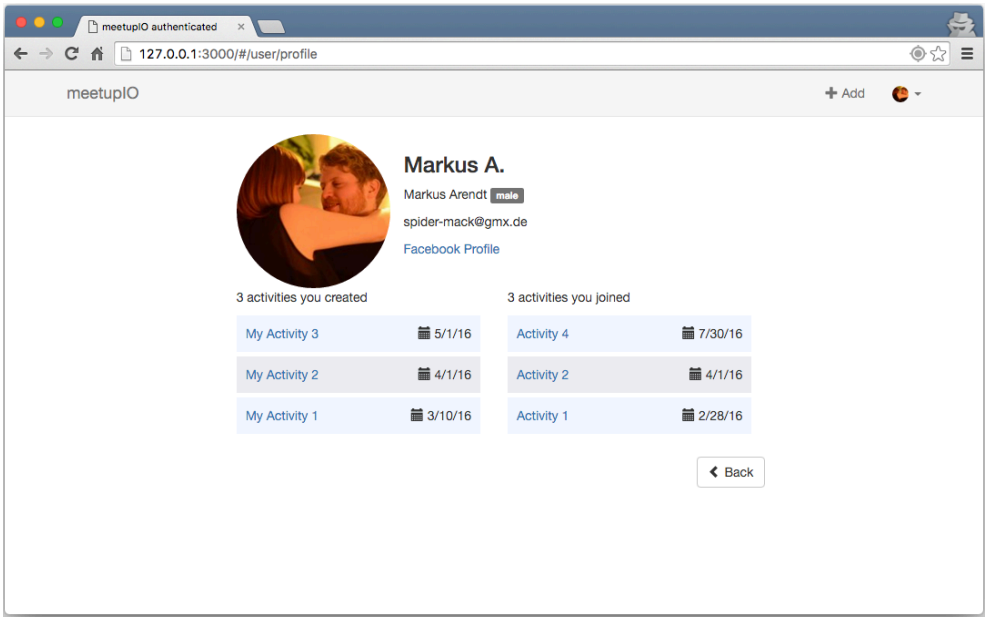


Abbildung 6: Screenshot der Webanwendung mit Partial userProfile.html

Ruft der Benutzer ein fremdes Benutzerprofil auf, wird das Partial *user.html* in die View geladen. Der Angular-Controller *userController* führt zum Erhalten der Benutzerdaten des gewählten Benutzers einen GET-Request an die Route */api/user/<USER._ID>* aus. Durch das Partial werden Profilbild, Username, Geschlecht und Link zum Facebook-Profil des Benutzers dargestellt. Schützenswerte Daten, wie die E-Mail-Adresse oder der vollständige Name eines Benutzers, werden nicht vom Server erhalten.

# 5 Ergebnis

Nach abgeschlossener Implementierung aller in Abschnitt 4 beschriebener Punkte, kann der Prototyp der Freizeitaktivitäts-Plattform lokal benutzt werden. Bevor der Gesamteindruck bewertet wird, werden Server und Client der entstandenen Webanwendung in diesem Abschnitt einzeln genauer betrachtet. Danach folgt als Aussicht eine Liste an Features, die im Rahmen der Bachelorarbeit nicht mehr implementiert werden konnten.

## 5.1 Server

Der lokale Server der Webanwendung arbeitet schnell und effizient. Nach abgeschlossener Implementierung konnten keine Komplikationen festgestellt werden. HTTP-Requests werden ohne Probleme vom Client empfangen und bearbeitet. Die Authentifizierung von Benutzern über Facebook funktioniert zuverlässig und sicher. Außerdem ist der Server, durch den gewählten Aufbau, leicht durch neue Funktionen erweiterbar. Die Anbindung zur lokalen MogoDB-Datenbank funktioniert ebenfalls performant und ausfallsicher. Schon während der Entwicklungsphase konnten keine Verbindungsabbrüche zwischen Node.js und MongoDB festgestellt werden. Daten werden zuverlässig gespeichert und ausgegeben.

Die Implementierung der Benutzerauthentifizierung nahm viel Bearbeitungszeit in Anspruch. Die dazu gewählte Middleware *Passport* setzt viel Einarbeitungszeit voraus, da die Funktionalität sich nicht auf den ersten Blick erschloss. Damit die Authentifizierung fehlerfrei funktionieren konnte, musste an anderen Komponenten des Servers viel Vorarbeit geleistet werden. Den größten Teil der Bearbeitungszeit nahm jedoch das Finden einer Struktur für den Server ein. Der MEAN-Stack lässt dem Entwickler alle Freiheiten, da ein fester Aufbau nicht vorgegeben ist. Auf der Suche einer geeigneten Struktur für die Webanwendung, musste der komplette Aufbau des Servers mehrfach überarbeitet werden. Die schließlich gefundene Struktur, auf Basis des MVC-

Modells, konnte durch leichte Erweiterbarkeit und logischer Arbeitsweise überzeugen.

## 5.2 Client

Die in Unterabschnitt 2.1.3 geforderte Funktionalität konnte vollständig in den Client der Webanwendung implementiert werden. Damit kann die Webanwendung als prototypische Freizeitaktivitäts-Plattform benutzt werden. Der Benutzer kann Unternehmungen innerhalb eines gewählten Umfeldes finden und an diesen teilnehmen. Darin kann er mit anderen Teilnehmern durch die Kommentarfunktion kommunizieren. Auch neue Unternehmungen können mit genauem Standort erstellt werden und sind für andere Benutzer zu finden. Dabei werden alle vom Server empfangenen Daten schnell und fehlerfrei dargestellt. Die Anforderung an den Client, durch ein responsives Design ohne Einschränkungen auf mobilen Endgeräten benutzbar zu sein, konnte durch Bootstrap ebenfalls erfüllt werden.

Trotzdem fallen bei der Benutzung einige Punkte negativ auf: Nutzt der Benutzer nicht die Umkreissuche, wird eine Liste an Unternehmungen mit allen in der Datenbank gespeicherten Unternehmungen vom Server empfangen und dargestellt. Dies kann bei einer großen Menge von in der Datenbank gespeicherten Unternehmungen zu Problemen führen. Einerseits wird dadurch eine unnötig große JSON-Datei empfangen, andererseits ist eine übersichtliche Darstellung im Client so nicht realisierbar. Ein Nachladen von Unternehmungen bei Bedarf konnte, im Rahmen der Bachelorarbeit, nicht mehr implementiert werden. Auch fällt auf, dass bestimmte Karten im Client dargestellt werden, bevor der Standort des Benutzers vollständig durch den Browser ermittelt wurde. Somit springt die Karte erst nach einer kurzen Verzögerung zum eigentlichen Standort des Benutzers und kann erst dann von diesem benutzt werden. Um diesen Fehler zu beheben, muss der Client in der Lage sein, verschiedene Elemente erst darzustellen, wenn die benötigten Daten client-

seitig vollständig bereitstehen. Auch dies war, im Rahmen der Bachelorarbeit, nicht mehr umzusetzen. Das Editieren des eigenen Benutzerprofils, welches im Server bereits durch eine Route implementiert ist, konnte im Client, ebenfalls aus zeitlichen Gründen, nicht mehr implementiert werden. Bei der Implementierung des Clients war die Einarbeitung in AngularJS die größte Herausforderung: AngularJS bietet als Framework sehr viele Komponenten an, deren Nutzen sich nicht sofort erschließen lässt. Die Einarbeitung in diese Technik erfordert viel Geduld und Zeit. Für die Realisierung des prototypischen Clients der Freizeitaktivitäts-Plattform war AngularJS vom Umfang her zu groß und komplex. Außerdem stellte sich die Benutzung des AngularJS-Directive *Ng-Map,* zur Einbindung der Karten, als konterproduktiv heraus: Das Directive soll die Einbindung von Karten in Angular-Apps erleichtern und bietet dafür eine Dokumentation mit vielen Anwendungsbeispielen. Auf den ersten Blick konnte das Directive durch einen logischen Aufbau und den vielen Anwendungsbeispielen überzeugen. Wird jedoch versucht, eine komplexere Darstellung von Daten in einer Karte zu realisieren, fehlt jegliche Hilfestellung oder Erklärung. Dies hat viel Bearbeitungszeit in Anspruch genommen. Die offizielle Dokumentation der Google Maps JavaScript API ist weitaus umfangreicher und hätte nach kurzer Einarbeitungszeit schneller zu einem Ergebnis geführt.

## 5.3 Gesamteindruck

Zusammengefasst wird bei der Benutzung der realisierten Webanwendung ein positiver Gesamteindruck vermittelt. Trotz den in Unterabschnitt 5.1 und 5.2 erwähnten, negativ auffallenden Punkten konnte die Entwicklung einer prototypischen Freizeitaktivitäts-Plattform als responsive Webanwendung abgeschlossen werden. Alle für eine prototypische Benutzung benötigten Features konnten erfolgreich in die Webanwendung implementiert werden. Die einzelnen Komponenten der Webanwendung laufen in der lokalen Test-

umgebung stabil und ausfallsicher. Bei Benutzung ist die Idee hinter der Freizeitaktivitäts-Plattform für den Benutzer deutlich zu erkennen. Auch die exakte Suche nach Unternehmungen im eigenen Umfeld konnte als Alleinstellungsmerkmal der Freizeitaktivitäts-Plattform realisiert werden. Darüber hinaus ist das Umfeld mittels variablem Suchradius und verschiebbarem Standort durch den Benutzer frei bestimmbar.

## 5.4 Aussicht

Um die Webanwendung schließlich für die Öffentlichkeit zugänglich machen zu können, fehlen weitere Features, die im Prototyp der Freizeitaktivitäts-Plattform nicht implementiert sind:

1. **Benutzerprofil editieren oder löschen**

   Benutzern muss es möglich sein, das eigene Benutzerprofil zu editieren oder endgültig aus der Datenbank der Freizeitaktivitäts-Plattform zu entfernen. Hier wurde nur das serverseitige Editieren vom eigenen Benutzerprofil bereits implementiert.

2. **Anderen Benutzern folgen**

   Um die Idee des Freunde-Findens in der Webanwendung zu unterstreichen, sollten Benutzer anderen Benutzern folgen können. So kann ein Benutzer über neue Unternehmungen eines anderen Benutzers informiert werden. Um Sicherheit gewährleisten zu können, muss dieses Feature beinhalten, dass Benutzer selbständig entscheiden können, ob ein anderer Benutzer ihnen auf der Freizeitaktivitäts-Plattform folgen darf oder nicht. Ein Benutzer mit vielen Follower erweckt Vertrauen bei anderen Benutzern.

3. **Erstellte Unternehmungen editieren oder löschen**

   Unternehmungen auf der Freizeitaktivitäts-Plattform müssen zwingend editierbar sein und nachträglich abgesagt oder komplett gelöscht

werden können. Dieses sehr wichtige Feature konnte innerhalb der Bearbeitungszeit der Bachelorarbeit nicht mehr implementiert werden. Wird eine Unternehmung nachträglich bearbeitet, abgesagt oder gelöscht, sollten alle Teilnehmer darüber in Kenntnis gesetzt werden.

4. **Teilnehmerliste einer Unternehmung verwalten**

Benutzer sollten die Teilnehmerliste eigener Unternehmungen verwalten können. Dabei sollten Teilnehmer nachträglich aus der Unternehmung entfernt werden können und eine maximale Anzahl an Teilnehmer festgelegt werden können. Ist die maximale Anzahl an Teilnehmer erreicht, sollte die Unternehmung für Benutzer, die nicht daran teilnehmen, nicht mehr angezeigt werden.

5. **Benachrichtigungen**

Um den ersten, zweiten und dritten Punkt dieser Liste realisieren zu können, ist die Implementierung von Benachrichtigungen notwendig. Benachrichtigungen können als E-Mail oder direkt in der Webanwendung realisiert werden. Ohne dieses Feature würden Benutzer nichts von Anfragen anderer Benutzer oder Änderungen an Unternehmungen, an denen sie teilnehmen, mitbekommen.

6. **Administrative Funktionen**

Bei zunehmender Anzahl an Benutzern und Unternehmungen werden verschiedene administrative Funktionen benötigt. Dazu müssen Benutzerkonten vorher administrative Rechte erteilt werden können. Diese können dann andere Benutzerkonten sperren und auffällige Unternehmungen löschen.

7. **HTTPS**

Bevor der Prototyp der Freizeitaktivitäts-Plattform die lokale Testumgebung verlassen kann, muss der Server vom HyperText Transfer Protocol (HTTP) auf das HyperText Transfer Protocol Secure

(HTTPS) umgestellt werden. Nur so kann, zum Schutz von Benutzerdaten, eine sichere Verbindung zwischen Client und Server gewährleistet werden.

# 6 Resümee

Die in Unterkapitel 1.1 gesetzte Ziele dieser Bachelorarbeit gelten als erfüllt. Es konnte eine prototypische Webanwendung realisiert werden, die der Idee der Freizeitaktivitäts-Plattform entspricht. Ebenfalls konnte dem Client ein erstes responsives Design gegeben werden, welches auf allen Endgeräten dargestellt werden kann. Das Ziel, die Webanwendung mit möglichst modernen Mitteln zu realisieren, wurde durch die Wahl des MEAN-Stacks, als Grundgerüst der Webanwendung, erfüllt. Der entstandene Prototyp der Freizeitaktivitäts-Plattform bietet eine solide Basis für weitere Implementierungen. Dieser kann zukünftig durch die in Unterabschnitt 5.4 genannten Features ergänzt und der Öffentlichkeit als vollständiges Produkt zugänglich gemacht werden.

Während der Bearbeitungszeit konnte ein tiefer Einblick in die verschiedenen Techniken, die der Realisierung moderner Webanwendungen dienen, gewonnen werden. Bereits vorhandene Grundkenntnisse konnten durch eine intensive Auseinandersetzung mit diesem Thema erweitert werden. Die größte Herausforderung dieser Bachelorarbeit war die Einarbeitung in alle benötigten Technologien innerhalb der kurzen Bearbeitungszeit. Das dadurch erarbeitete Wissen und die dazugewonnenen Erfahrungen werden für zukünftige Projekte im Bereich Webentwicklung von großem Wert sein.

# Literaturverzeichnis

[1] Facebook Inc., „Company Info: Statistics," [Online]. Available: http://newsroom.fb.com/company-info. [Zugriff am 7 März 2016].

[2] Stiftung Weltbevölkerung, „Zum Jahreswechsel leben 7.391.068.000 Menschen auf der Erde," 23 Dezember 2015. [Online]. Available: http://www.weltbevoelkerung.de/aktuelles/details/show/details/news/zum-jahreswechsel-leben-7391068000-menschen-auf-der-erde.html. [Zugriff am 7 März 2016].

[3] M. della Cava, „Facebook in 2030? 5 billion users, says Zuck," 4 Februar 2016. [Online]. Available: http://www.usatoday.com/story/tech/news/2016/02/04/facebook-2030-5-billion-users-says-zuck/79786688. [Zugriff am 7 März 2016].

[4] Wikipedia, „Webanwendung," [Online]. Available: https://de.wikipedia.org/wiki/Webanwendung. [Zugriff am 9 März 2016].

[5] Bharat, „Web Application Architechture – Client / Server Architecture," 24 Juli 2010. [Online]. Available: https://webingineer.wordpress.com/2010/07/24/web-application-architechture-client-server-architecture. [Zugriff am 10 März 2016].

[6] S. Davis, „Mastering MEAN: Introducing the MEAN stack," 9 September 2014. [Online]. Available: http://www.ibm.com/developerworks/library/wa-mean1/index.html. [Zugriff am 10 März 2016].

[7] Wikipedia, „Solution stack," [Online]. Available: https://en.wikipedia.org/wiki/Solution_stack. [Zugriff am 10 März 2016].

[8] D. N. Bezroukov, „LAMP Stack as new program development paradigm,“ [Online]. Available: http://www.softpanorama.org/WWW/lamp.shtml. [Zugriff am 2016 März 2016].

[9] Wikipedia, „MEAN (software bundle),“ [Online]. Available: https://en.wikipedia.org/wiki/MEAN_(software_bundle) . [Zugriff am 10 März 2016].

[10] D. Wyllie, „MySQL vs. MongoDB: Datenbanksysteme für Web-Anwendungen im Vergleich, Abschnitt 3,“ 29 Mai 2014. [Online]. Available: http://www.computerwoche.de/a/datenbanksysteme-fuer-web-anwendungen-im-vergleich,2496589,3. [Zugriff am 11 März 2016].

[11] AngularJS, „What Is Angular?,“ [Online]. Available: https://docs.angularjs.org/guide/introduction. [Zugriff am 12 März 2016].