FH JOANNEUM Gesellschaft mbH

MEAN Web Application Development

Bachelorarbeit 2

eingereicht am 15.11.2015

Fachhochschul-Studiengang Software Design

Betreuer: DI Johannes Feiner

eingereicht von: Daniel Frech

Personenkennzahl: 1210418058

November 2015

Zusammenfassung

Für die Entwicklung moderner und interaktiver Web Applikationen ist die Auswahl der richtigen Technologien und Frameworks von großer Bedeutung. Diese Entscheidung ist keine leichte, bieten doch unzählige Frameworks verschiedenartige Möglichkeiten. MEAN ist dabei ein weiteres Framework, das verspricht diesen Anforderungen gerecht zu werden. Es vereint die dokumentenorientierte NoSQL Datenbank MongoDB, Node.js und Express.js für die Entwicklung des Applikationsservers, sowie AngularJS für die clientseitige Darstellung der Applikation in einem Framework.

Diese Arbeit untersucht die vier Technologien des MEAN Frameworks im Detail. Neben der Entwicklung einer Muster-Applikation wird das MEAN Framework mit zwei weiteren Wettbewerbern, Django und Oracle Application Express, in fünf verschiedenen Kategorien verglichen. Dabei zeigt sich, dass das MEAN Framework den Anforderungen der Entwicklung moderner Web-Applikationen gerecht wird. Bestnoten in den Kategorien „User experience“, „Change- and testability“ und „Deployment“ unterstreichen das gute Ergebnis. Der vergleichbar hohe Einarbeitungsaufwand und die nicht in allen Punkten umfangreiche Dokumentation führen allerdings zu Abzügen in den Kategorien „Rapid application development“ und „Development environment“.

Abstract

The choice of the right tools, technologies and frameworks is an essential one, when building modern and interactive web applications. With the number of available frameworks that offer various possibilities, this is not an easy choice to make. The MEAN stack is an additional framework in that respect, that promises to fulfil the requirements for building modern web applications. It combines the document-orientated NoSQL database MongoDB with Node.js and Express.js for building the application server and AngularJS for the client-side user interface into one framework.

This thesis examines the technologies of the MEAN framework in detail. Besides the development of a prototype application, the MEAN framework is compared in five categories with two other frameworks, Django and Oracle Application Express. The results show, that the Mean framework does meet the requirements for building modern web applications. This manifests in top grades in the categories "User experience", "Change- and testability" and "Deployment". The comparably hight initial effort that is needed to get productive with the framework and the in some parts not sufficient documentation results in deductions in the categories "Rapid application development" and "Development environment".

Contents

List of abbreviations

APEX Application Express

API Application Programming Interface

CDN Content Delivery Network

CSS Cascading Style Sheets

CRUD Create-Read-Update-Delete

DRY Don't Repeat Yourself

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

I/O In/Out

JS JavaScript

MEAN MongoDB, Express.js, Angular.js, Node.js

MTV Model-Template-View

MVC Model-View-Controller

NPM Node Package Manager

OS Operating System

PEP Python Enhancement Proposal

PIP Python Package Index

PL/SQL Procedural Language/Structured Query Language

RAD Rapid Application Development

RDBMS Relational Database Management System

SPA Single Page Application

SQL Structured Query Language

UI User Interface

WSGI Web Server Gateway Interface

XE Express Edition

Chapter 1

Introduction

This introductory chapter outlines the motivation, which explains why the subject of this thesis is relevant. Additionally, the content of the research is introduced. The final part illustrates the structure of this thesis.

1.1 Motivation

When developing modern and interactive websites, the choice of the right tools is essential. This choice is not an easy one, as numerous technologies and frameworks offer various possibilities and options. The MEAN stack is an additional framework in that respect, that promises to fulfil the requirements for building modern web applications. It combines the document-orientated NoSQL database MongoDB with Node.js and Express.js for building the application server and AngularJS for the client-side user interface into one full-stack web development framework.

1.2 Research content

This bachelor thesis examines the details that the MEAN framework consists of and introduces a prototype application that has been build with the MEAN stack. The application "kletter.tracker" lets climbers track their climbed routes and gives the management staff of climbing halls the possibility to manage their routes and customers.

In addition, the MEAN framework is compared to two other web application frameworks, Oracle Application Express and Django. The comparison is divided into five different categories, that reflect the needs for developing modern web applications. Further, each category has been given a certain weighting factor, based on the prototype application. Table 1 shows the five different categories with their weighting factor.

Comparison	**Weighting**
Rapid application development	0.5
Development environment	1.5
User experience	2.0
Change- & testability	1.0
Deployment	1.0

Table 1: Weighting for prototype application

The goal is to provide a grounded guidance for developers, to select the right tool for their next project.

1.3 Structure

Chapter 2 presents the four different frameworks used with the MEAN stack. It will focus on selected details and highlight the points, that make each framework special and different to other frameworks currently in use.

Chapter 3 introduces the prototype application that has been built for this thesis. Besides selected code snippets of the application it describes, how the MEAN application has been deployed.

Chapter 4 will take the things learned from building the prototype application and compare the MEAN stack to two other web-application development frameworks.

Finally, chapter 5 wraps up the results from the comparison and reflects on the things learned.

Chapter 2

The MEAN stack

2.1 Overview

MEAN is a word that literally has different meanings. One of them is that of "being mean", which is a synonym of being nasty, evil, unpleasant or malicious. It could however, in a more positive way, also be described as being different, attacking others weaknesses and aggressively providing different ways of doing things. In the respect of web application development, MEAN is an acronym for the combination of four different technologies, that together form a full stack web application framework:

- **MongoDB** is an agile and scalable NoSQL database that stores data in form of documents, rather than traditionally in tables.
- **Express.js** is a Node Package Manager (NPM) package that is used for building web servers. It is the most common package for building web applications with Node.js (cf. Dickey (2014), p. 20).
- **AngularJS** is a framework for building applications inside the browser and is targeted toward highly interactive applications.

- **Node.js** is a development framework, that is based on Google's V8 JavaScript engine. It uses an event-driven, non-blocking I/O model that makes it a good fit for building fast and scalable network applications (cf. Gackenheimer (2013), p. 1).

All technologies have in common, that they use JavaScript as their programming language, hence allowing the programmer to use the same language from the back-end to the client-facing user interface.

This chapter takes a closer look at the four different technologies used in the MEAN stack, as well as the combined framework itself. For every technology alone, several books have been published throughout the last years. Therefore the introduction in this thesis focuses on the special parts and strengths of each framework, but does not include a full introduction of each technology. However, there are some excellent resources on the internet for learning the different technologies, like `https://thinkster.io` for AngularJS. Appendix A on page 56 lists additional sources for each framework.

2.2 MongoDB

The M in MEAN is reserved for the database, MongoDB. With that, the MEAN stack introduces a NoSQL database, instead of the often used Relational Database Management System (RDBMS) in other frameworks.

> *"MongoDB is a database management system designed for web applications and internet infrastructure. The data model and persistence strategies are built for high read and write throughput and the ability to scale easily with automatic failover. Whether an application requires just one database node or dozens of them, MongoDB can provide surprisingly good performance."* (Banker (2015), p. 4)

MongoDB is a document-oriented database, that is built upon a set of basic concepts (cf. Chodorow (2013), p. 29):

- Data is stored in form of *documents*, which is roughly equivalent to a row in a RDBMS. However, data can be saved in a more expressive way in a document. For example, also nested data of any depth can be stored in documents. In addition, documents do not follow a schema, like rows in relational tables do. This means, that every document, even for the same data can take up varying data. Documents in MongoDB are stored in the BSON format, which is MongoDBs extension of the more commonly known JSON format (cf. MongoDBJsonBson (visited on 21.10.2015)).
- A *collection* is a group of documents, similar to a table in a RDBMS.
- Each document has a special field called "_id" that uniquely identifies the document within a collection.
- Collections are grouped into *databases*. A single instance of MongoDB can host several databases, each containing an arbitrary amount of collections. Each database has its own permissions and is stored in separate files on disk.

MongoDB also supports replication and sharding. *Replication* is a way of keeping identical copies of data on different servers. MongoDB accomplishes this with *replica sets*, which is a group of servers with one primary and multiple secondary servers, that store copies of the data stored on the primary server (cf. Chodorow (2013), p. 169). This has the advantage, that even if one server fails, the data is still stored on the secondary servers. The downside of replication is the possibility of data inconsistency. With different clients reading from different servers, there is the danger that a change has not yet propagated to all secondary servers for example (cf. Sadalage & Fowler (2012), p. 42). MongoDB diminishes this danger, by only allowing requests to the primary server. Still, in case of a failure there is a

possibility that not all changes have already been saved on all secondary servers. Figure 1 illustrates the concept of replica sets in MongoDB.

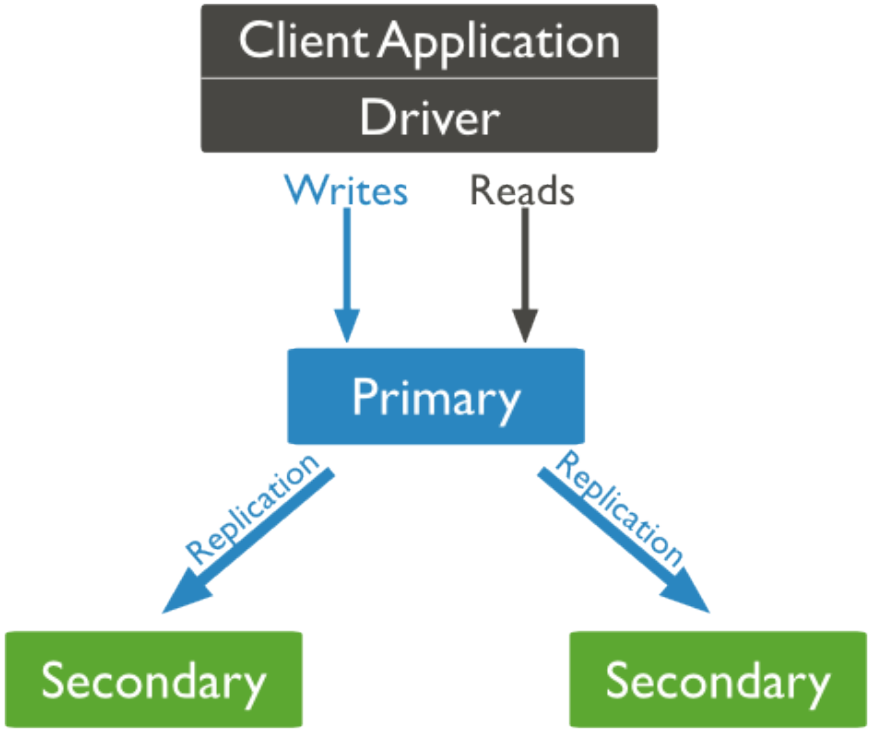

Figure 1: Replication (MongoDBReplication (visited on 21.10.2015))

Sharding in contrast, distributes the data across multiple servers. Not all data is stored on every server, but only part of the data. The goal is, to scale horizontally rather than vertically, meaning that more demand can be handled by adding more servers, rather than by empowering the one server that contains the data. Essentially, sharding is not an easy process, but MongoDB has been built with the very concept in mind from the beginning. Figure 2 shows the process of sharding. Mongo routers have the responsibility to fetch the requested data from the appropriate shards. Each shard, quite interestingly, is its own replica set, hence can again be replicated if needed (cf. Banker (2015), p. 186).

As already mentioned, MongoDB documents, in the MEAN stack also called models, do not have any schema that they adhere to. That means, that data in a document can be stored in any kind of format. As a result, documents within one collection must not necessarily have the same structure. What sounds tempting

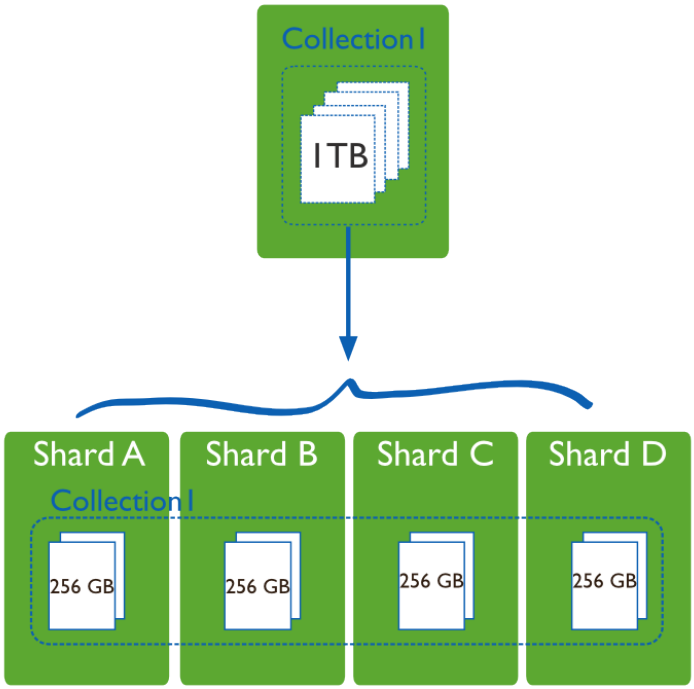

Figure 2: Sharding (MongoDBSharding (visited on 21.10.2015))

at the beginning, can also lead to severe problems when trying to process data from documents with different structures. To organize models the MEAN stack therefore also includes a helping tool called *Mongoose*. The main benefits of Mongoose are (cf. Dayley (2014), p. 297):

- It allows the creation of a schema structure for documents
- Documents can be validated
- Application data can be typecasted into the object model
- Business logic hooks with middleware can be applied

Combined with Mongoose, MongoDB offers a robust and flexible alternative to storing data, that nicely fits the needs in modern web development.

2.3 Express.js

Express.js (Express) is a web application framework for Node, modelled after the Ruby project Sinatra (cf. Wilson (2013), p. 88). It is minimal and out of the box only supports the basic features of a web framework. Yet it is very powerful and flexible, as middle-ware and Node modules can extend the features in a modular way. The following code snippet would be sufficient, to create a "Hello world" web server with Express.js:

```
1 var express = require('express');
2 var app = express();
3
4 app.get('/', function(req, res) {
5    res.send(200, 'Hello world!');
6 });
```

Express consists of three core components (cf. Yaapa (2013), p. 9):

- The *application object* is an instance of Express, in the above example represented by the variable "app". It is the main object of an Express application and all other functionality is built upon it.
- The *request object* is created, whenever a client makes a request to the Express app. It contains properties and methods related to the current request. In above example, the variable "req" represents the request object.
- The *response object* is created together with the request object, in order for the middleware to work on the request and response object at the same time. In above snippet, the response object is represented by the variable "res".

When working with the MEAN stack, Express is the least noticeable of the four technologies. In fact, as Express is a Node.js package, there is nothing that could not be done without it and pure Node instead. But Express adds a layer of

abstraction that makes it easier to create the web server. The following example shows this difference between pure Node.js and Express for the task of routing incoming requests-

```
// node.js with http module
var http = require("http");

http.createServer(function(req, res) {

    // Index page
    if (req.url == "/") {
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end("Welcome! This is the main page!");
    }

    // Reservations page
    else if (req.url == "/reservations") {
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end("This is the reservations page!");
    }

    // Not found
    else {
        res.writeHead(404, { "Content-Type": "text/plain" });
        res.end("Error 404: Page not found.");
    }

}).listen(80, "localhost");
```

With Node.js, the routes are defined in an if block together with the creation of the server. When more routes are added, this can quickly become confusing and difficult to maintain.

In Express in contrast, routes are added to the Express app. The creation of the server is separated from the routings.

```
// express.js
var express = require("express");
```

```
var http  = require("http");
var app   = express();

app.all("*", function(request, response, next) {
    response.writeHead(200, { "Content-Type": "text/plain" });
    next();
});

// Index page
app.get("/", function(request, response) {
    response.end("Welcome! This is the main page!");
});

// Reservations page
app.get("/reservations", function(request, response) {
    response.end("This is the reservations page!");
});

// Not found
app.get("*", function(request, response) {
    response.end("Error 404: Page not found.");
});

http.createServer(app).listen(80);
```

While Express is the most popular and often used package for building web servers with Node.js, there are other, recently developed packages like koa.js or hapi.js for the same tasks. Hapi.js for example offers a lot of functionality out of the box, that needs to be added with middle-ware in Express. The same example from above would require a few more lines in hapi.js, but would further increase the readability of the routing configuration.

```
// hapi.js
const Hapi    = require('hapi');
const server  = new Hapi.Server();

server.connection({
```

```
    host: 'localhost',
    port: 80
});

// Index page
server.route({
    method: 'GET',
    path:'/',
    handler: function (request, reply) {
        return reply("Welcome! This is the main page!");
    }
});

// Reservations page
server.route({
    method: 'GET',
    path:'/reservations',
    handler: function (request, reply) {
        return reply("This is the reservations page!");
    }
});

// Not found
const handler = function (request, reply) {
    return reply("Error 404: Page not found.").code(404);
};

server.start((err) => {
    if (err) {
        throw err;
    }
});
```

While Hapi.js can be a valuable alternative for Express, for the scope of this thesis and the prototype application, the MEAN stack has been taken as it is with the combination of the Express framework.

Within the MEAN stack, Express is responsible for the RESTful API that is provided at the server side. Besides that, it helps with tasks such as session management via cookies, parsing of incoming requests or rejection of malformed requests.

2.4 AngularJS

AngularJS is a framework for building applications inside the browser, that is built on top of JavaScript and a lightweight version of jQuery. It is targeted toward highly interactive applications. Out of the box, it comes with many components useful for building applications. AngularJS is an open source project that is sponsored and maintained by Google.

At the heart of AngularJS is a concept called *two-way data binding*. This allows the binding of Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) to the state of a JavaScript variable. If the variable changes, AngularJS updates all HTML and CSS that hold a reference to the variable (cf. Karpov & Netto (2015), p. xxvi).

The following example illustrates the data-binding in AngularJS and the difference to other JavaScript libraries, such as JQuery. Both with JQuery and AngularJS a message should immediately show the text entered in an input field. With JQuery, the way to change the content of the web page is to manipulate the DOM. To achieve this, an event listener is added to the input field, after the page is loaded. The JQuery code would therefore contain the a section like the following to display the input field and message section:

```
1 <div>
2     <input type="text" id="jqueryInput" placeholder="Enter a message
       here">
3     <h2>Message: <span id="jqueryMessage"></span></h2>
4 </div>
```

In addition, a script would add the JQuery event listener after the document has loaded:

```
<script>
    function updateJQueryMsg(){
        $( "#jqueryInput" ).keyup(function() {
            $("#jqueryMessage").text($( "#jqueryInput").val());
        });
    }
    $( document ).ready(function() {
        updateJQueryMsg();
    });
</script>
```

AngularJS instead makes the HTML document the definite source of how the data is displayed. And it binds an HTML or CSS property to the value of a JavaScript variable. The HTML mark-up could in this case look as follows:

```
<div ng-app>
    <input type="text" ng-model="angularjsMessage" placeholder="
    Enter a message here">
    <h2>Message: {{angularjsMessage}}</h2>
</div>
```

Both JQuery and AngularJS achieve the same result in this example. However, with AngularJS the same message could be easily repeated by only multiplying the message line. And, more importantly, this binding can be factored out into separate files as controllers, that handle the calculation of variables as values change.

In summary, two-way data binding eliminates the need for web programmers to explicitly set up DOM interactions. As these DOM interactions are one of the leading causes of web application faults, the concept helps to make the application more reliable and maintainable (cf. Ocariza et al. (2015), p. 1).

Appendix C on page 58 shows the full HTML for the example given here. It has to be noted, that JQuery plug-ins like "jquerymy.js" can enhance the functionality

of JQuery to also make use of two-way data binding without AngularJS.

Besides data binding, AngularJS offers various other important components:

- *Modules* represent components in an application. They provide a namespace and make it easier to package and reuse parts of an application (cf. Dayley (2014), p.398).

- A *scope* is a JavaScript representation of data that is used to populate a view presented on the web page. The source of a scope can either be from the database, but can also come from a remote web service or a client-side AngularJS code. The scope is important with AngularJS, as it can be manipulated within the AngularJS code (cf. Dayley (2014), p.399).

- *Directives* in AngularJS extend and enhance HTML in order to create rich web applications. Besides many built-in directives which can directly be used, AngularJS also allows to build custom directives. Depending on the type of directive, they can then be applied to HTML elements (cf. Freeman (2014), p. 220).

- *Controllers* expose JavaScript data and functions to the HTML. They are instantiated from the HTML by using the "ng-controller" directive. In contrast to the services, a new instance of a controller is created with every instance of the "ng-controller" directive. In essence, controllers are responsible for exposing an Application Programming Interface (API) to the HTML. The directives then interact with this API to produce the web page that is visible (cf. Karpov & Netto (2015), p.96).

- *Services* are singleton objects, that provide certain functionality for the application. They are independent of context and state and can be consumed from the components of an application. The instance of a service is shared between all controllers, services and directives that depend on that service. Hypertext Transfer Protocol (HTTP) requests, logging, parsing or anima-

> tions are examples of already built-in services of AngularJS (cf. Dayley (2014), p.400).

All components together make up a framework, that empowers developers to build modern and highly interactive web applications.

2.5 Node.js

> Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world (nodejs.org (visited on 21.10.2015)).

The statement mentions the central aspect of Node.js, its non-blocking In/Out (I/O) model. This is in contrast to existing web-servers like Apache for example, where a blocking I/O operation causes the connection to wait until the operation has completed. Blocking I/O are for example database queries or file reads. The time the connection needs to wait is thereby dependent on the latency of the I/O operation and can last from milliseconds up to minutes. In order to handle a higher amount of requests, those web-servers usually use a multi-threaded approach, where each thread handles one or more server connections. Unused threads are handed back to a thread-pool, waiting for the next connection (cf. Cantelon et al. (2013), p. 7). To illustrate this, imagine a restaurant, where the service personnel is not only taking the orders, but also preparing the drinks and food. After each order, the service personnel would be busy for quite some time. The only way to serve more guests would be to increase the number of service personnel.

Node.js non-blocking I/O model works the other way round instead. Just as in a real restaurant, the service personnel would pass the food and beverages order to

the kitchen and the bartender, after haven taken the order. That way, the service personnel is free to serve other guests in the meantime. This concept can handle more requests at the same time and has recently also been introduced in fast-food chains. Where previously the personnel at the counter took the order of the guest and then picked-up all the ordered items, these tasks have been split. As a result, more orders can be handled at the same time and with many customers wanting to place orders at the same time, no long queues develop at the order counter.

In Node.js there is, following this concept, only a single thread for all connections. For slow I/O operations the program never waits, but instead immediately continues with the next line of code. Whenever the I/O operation returns , a callback function is triggered and the result can be processed (cf. Wandschneider (2013), p. 4). Going back to the example of the fast-food counter, the callback-function would be the order number that the customer receives upon completing the order and that is displayed on a monitor, once the order is ready for pick-up. Figure 3 illustrates the processing model of Node.js.

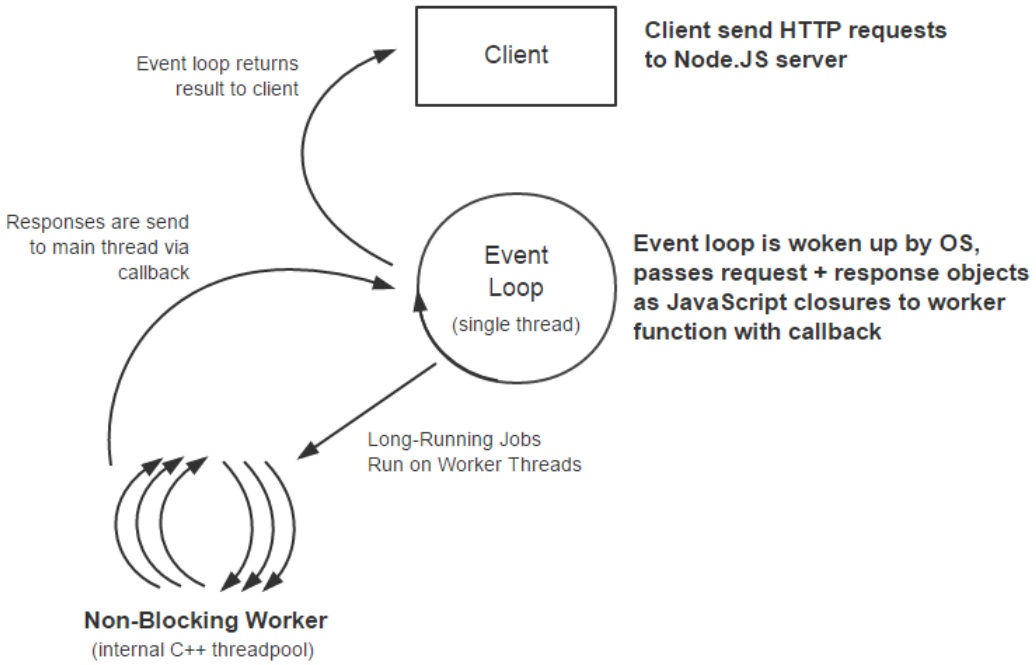

Figure 3: Node.js processing model (Liu (visited on 23.10.2015))

The problem that can arise with this approach is also quite obvious. In the exam-

ple of the fast-food restaurant, because of the non-blocking model, the number of personnel taking orders can be reduced. If now anything however does block the remaining personnel, the impact on the order process is much higher than before. For Node.js this means, that the developer has to take care meticulously to not write any blocking code. Nevertheless, losing control of asynchronous code in JavaScript can be easy (cf. Casciaro (2014), p. 58). It is therefore especially important to not sacrifice the qualities of modularity, reusability, and maintainability. Luckily, with the MEAN stack all technologies work together well and each of them provide many features, like testing or code analysis, to emphasize these qualities.

2.6 The combined framework

Currently there are two separate projects called MEAN: MEAN.JS and MEAN.IO. Both are open source projects hosted on Github, that combine the four technologies into one framework. The goal is, to quickly being able to get up and running with the full development stack. And of course, that applications with the MEAN stack can easily be deployed whenever production ready. To achieve this, both frameworks have included a few other useful frameworks such as Bower for handling web libraries, Grunt and Gulp for handling repetitive tasks or, in case of the MEAN.JS framework, Yeoman for bootstraping the project.

Both projects have been initiated by the same person, Valeri Karpov. MEAN.js is however a fork of MEAN.io, that happened after the open source project of MEAN.io has been taken over by a company called Linnovate. The exact details of why Valeri Karpov forked out the project to the new MEAN.js project can be found on the MEAN.js blog at `http://blog.meanjs.org/post/76726660228/forking-out-of-an-open-source-conflict`.

For this thesis the MEAN.js project has been chosen for two reasons:

1. MEAN.js is the currently more active project on Github. While the total number of forks and watches is higher with MEAN.io, MEAN.js has more commits and forks recently (see Github projects "meanjs/mean" and "linnovate/mean").

2. Simple deployment options like Digital Ocean offer a pre-build MEAN ecosystem upon server creation, that is based on the MEAN.js framework (cf. Brežnjak (visited on 23.10.2015))

To sum up the introduction it has to be noted, that there are also other alternatives, that make use of the technologies in the MEAN stack, like Meteor, Sails.js or Cleverstack to only name a few. Nevertheless, the MEAN stack offers a good combination and predefined set-up for starting application development with these technologies. The following chapter will present the prototype application, that has been built for this thesis with the MEAN technologies.

Chapter 3

Prototype application

3.1 Overview

The prototype application built with MEAN.js, "kletter.tracker", is a tracking app for climbers and a management app for operators of climbing halls. German has been chosen as the language of the application.

The application is a Single Page Application (SPA) that provides the following functionality for climbers:

- *Route tracking*: Tracking of climbed routes for the logged in user.
- *Statistics*: Display of all climbed routes of the logged in user.
- *The wall*: Display of currently climbed routes of all users.
- *Social-media integration*: Ability to log in with social-media account from Facebook or Google and linking social-media accounts to the user profile.

In addition, the application provides the following functionality for the climbing hall management staff:

- *Route management*: Management of climbing routes in the climbing hall. Besides the name, a route should represent a difficulty grade, a color that identifies the handles, the category of route and the section where the route is located.

- *Configurable*: Codes like sections, difficulty grades and categories are freely configurable.

- *User management*: Users can be created and managed from within the app. Besides the possibility to change profile data, the user management displays climbed routes for each user.

For this thesis, the application has been built completely by using the MEAN stack. Another scope of the prototype application has been to provide an API, that can be consumed by other applications, such as a native mobile app. An iOs app has been developed together with Daniel Hösele, that reuses the REST API built with the MEAN application. The mobile app itself is however not part of this thesis and therefore not included in the GitHub project. Figure 4 shows an architectural overview of the prototype application.

The following figures show selected screen shots of the application. Figure 5 displays the login page, where users can log in with a local or social media account, register a new account and reset their password.

Figure 6 shows the page to track climbed routes. Besides the tracking, users can rate the route, change the difficulty grading and leave a comment.

"The Wall" displays routes that climbers have tracked recently. This page can be seen in figure 7.

And finally, figure 8 shows the overview page of user accounts for the management staff of the climbing hall.

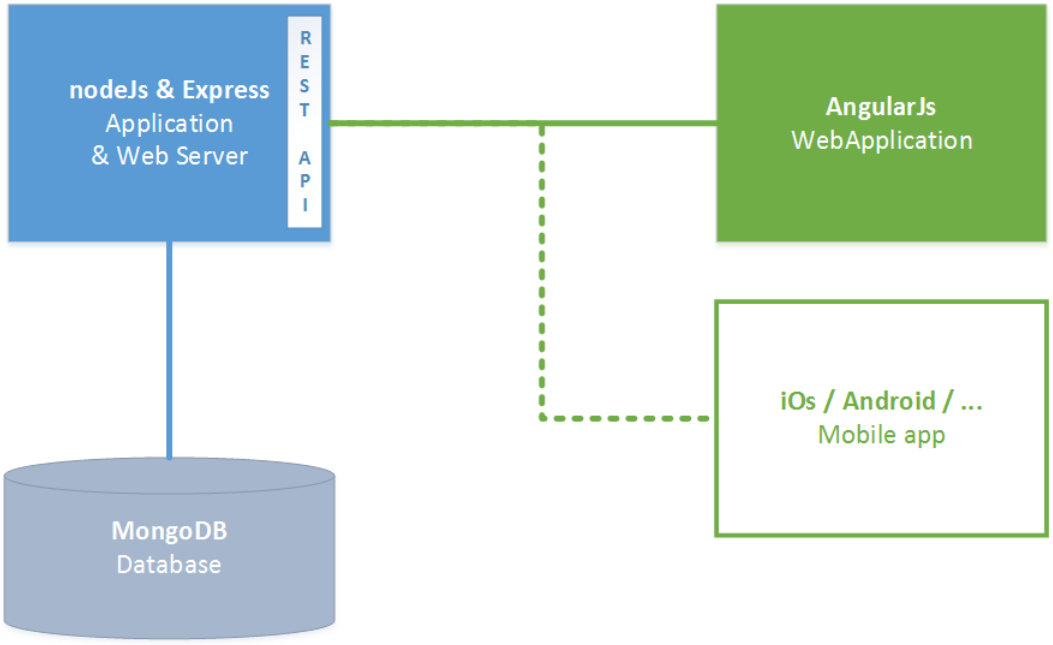

Figure 4: Prototype application - overview

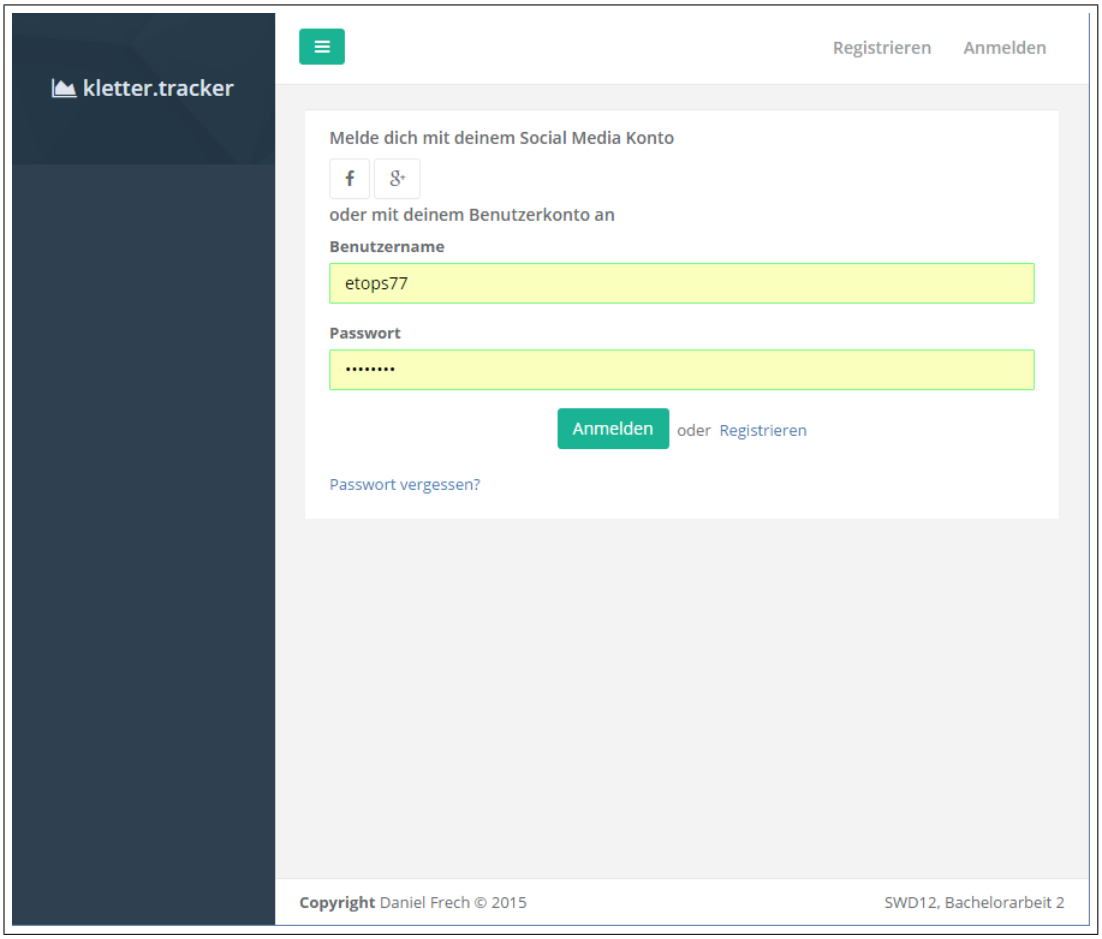

Figure 5: Prototype application - Login page

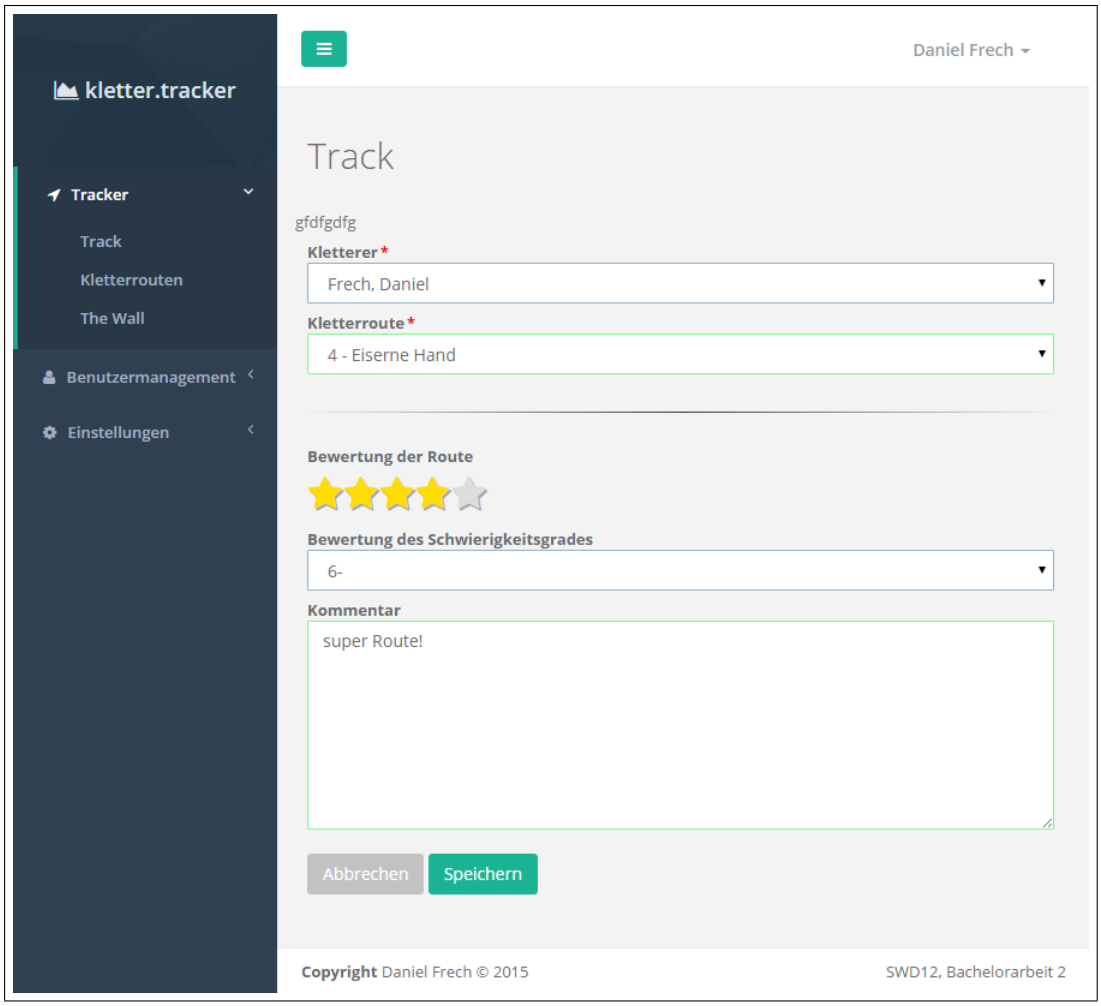

Figure 6: Prototype application - Track climbed route

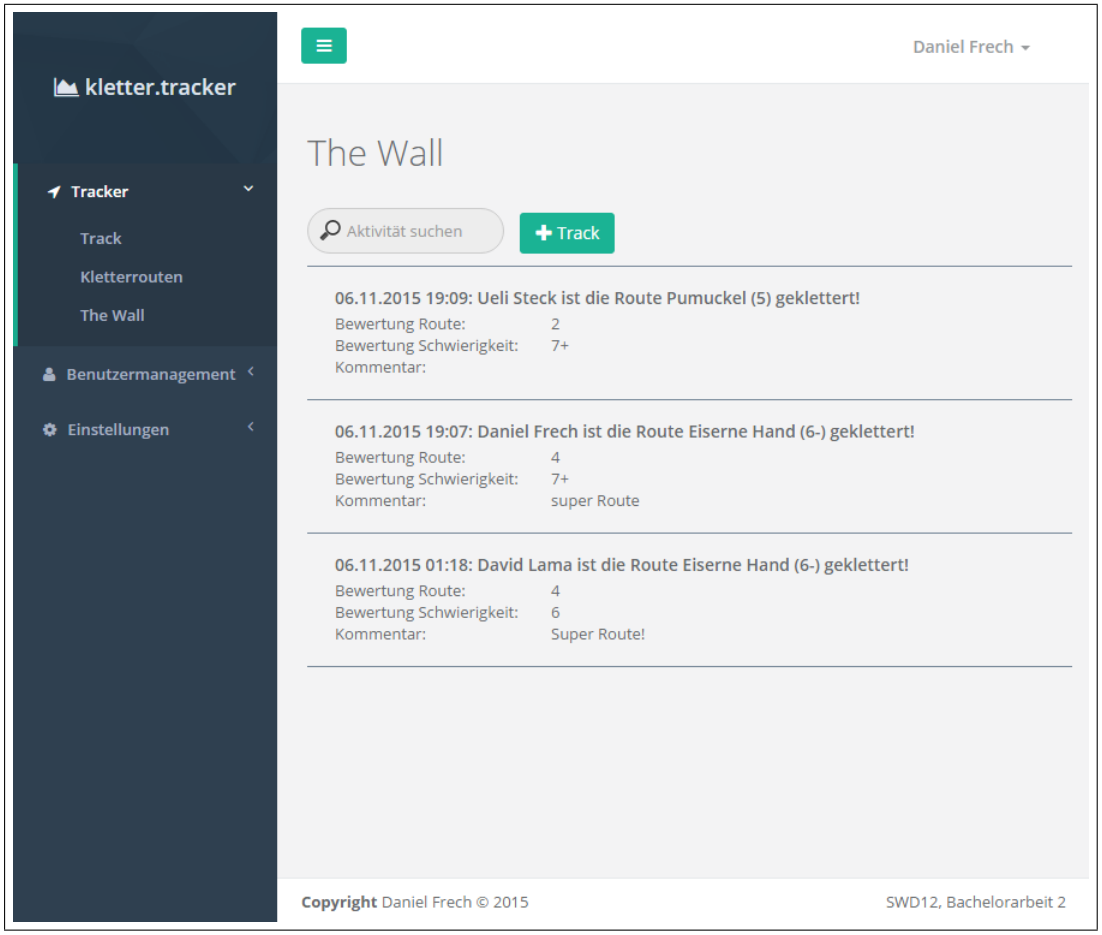

Figure 7: Prototype application - The Wall

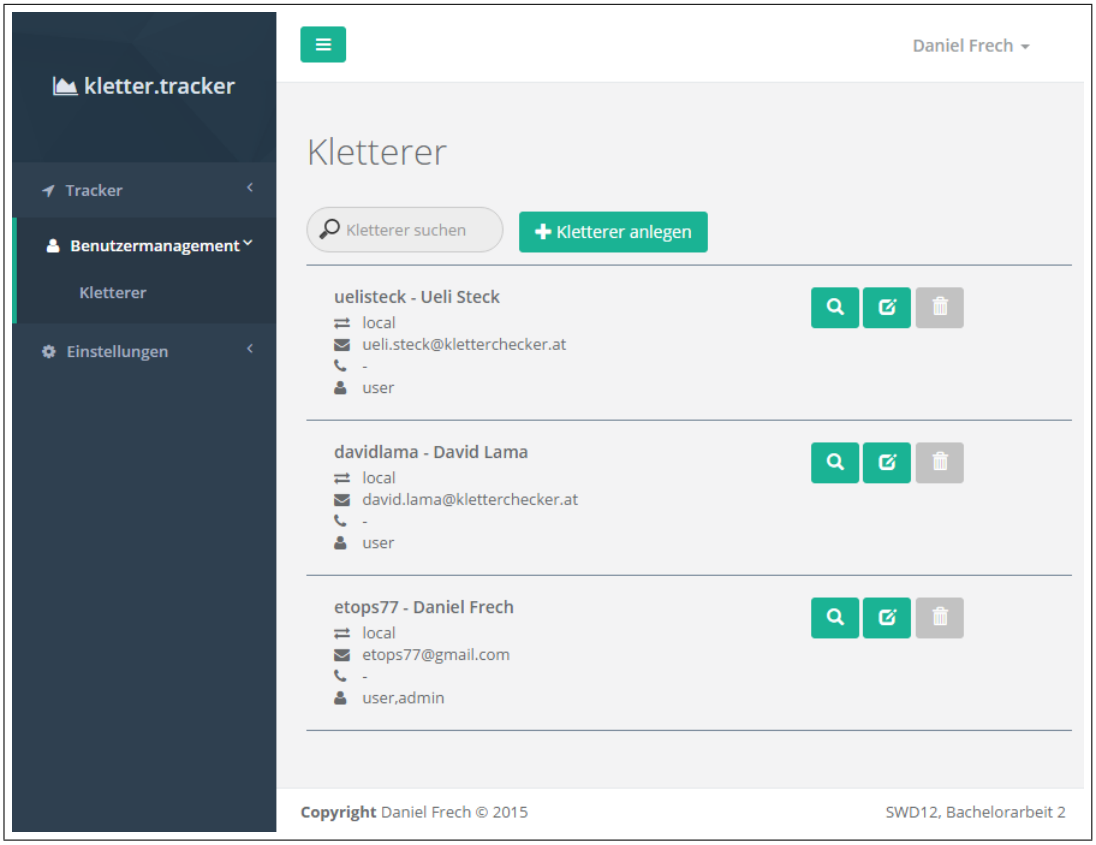

Figure 8: Prototype application - Customer overview

3.2 Code snippets

The following code snippets give an impression of the developed application and present some of the notable parts during development.

The first snippet shows a part of the AngularJS controller that fetches all users. The result is used in the select box for users, as can be seen in figure 6. The query is only executed, if the logged in user has the role "admin" granted. If that is the case, a promise function selects all users, with exception of the logged in user, into an array.

```
$scope.climbers = [];
if ($scope.authentication.user.roles.indexOf('admin') > 0 ) {
  Users.query().$promise.then(function (climbers) {
    climbers.forEach(function (climber) {
      if (climber._id !== $scope.authentication.user._id) {
        $scope.climbers.push(climber);
      }
    });
  });
}
```

The next snippet shows the Mongoose model of the activity model, which is used when a climbing route is tracked. Here, the activity document should hold a reference of the object id of the climbing route that is tracked.

```
climbingroute: {
  type: Schema.ObjectId,
  ref: 'Climbingroute'
}
```

Following up on the previous declaration of the activity model, the next snippet shows the selection of activities through the "list" function in the Node.js controller. To not only return the object id of the climbing route, the object is populated with the full object instead with the "populate" command.

```
exports.list = function(req, res) {
```

```
  Activity.find().sort('-created').populate('climbingroute').exec(
    function(err, activities) {
    if (err) {
      return res.status(400).send({
        message: errorHandler.getErrorMessage(err)
      });
    } else {
      res.jsonp(activities);
    }
  });
};
```

The last snippet shows the markup for a search field in the AngularJS view. The input field is in this case passed to the filter of the section that displays the climbing routes.

```
<div>
  <input ng-model="search.$" type="search" placeholder="Route suchen
    ">
</div>
...
<table>
  <tr data-ng-repeat="climbingroute in climbingroutes | filter:
    search:strict">
    <td>
      <h4 {{climbingroute.number}} - {{climbingroute.name}}</h4>
    </td>
  </tr>
</table>
```

3.3 Project deployment

The prototype application has been deployed to a Digital Ocean server. Digital Ocean offers the creation of servers (or dropplets, as they are called at Digital Ocean), that have the prerequisites for MEAN applications already installed. The

following list describes the steps necessary for the development of the MEAN application:

- First, a new release has to be created within the GitHub repository. In this case, the release "0.0.1" has been created from the web interface of GitHub. The same could be created from the Git shell or an IDE like WebStorm as well.
- After that, the Digital Ocean droplet with the MEAN stack needs to be created.
- With the following Git command, the release can be downloaded into a folder on the droplet:

```
$ git clone --branch 0.0.1 https://github.com/etops77/
    klettertracker.git --depth 1
```

- After the download, the NPM dependencies need to be installed with the following command:

```
$ npm install
```

- Finally, the production environment and the port for the application to run need to be defined. Then the application can be started:

```
$ export NODE_ENV=production
$ export PORT=80
$ grunt
```

For future releases, the same process can be followed. Figure 9 illustrates this process.

3.4 Project code

The source code of the prototype application can be obtained from the CD that is enclosed to this thesis and the GitHub repository located at `https://`

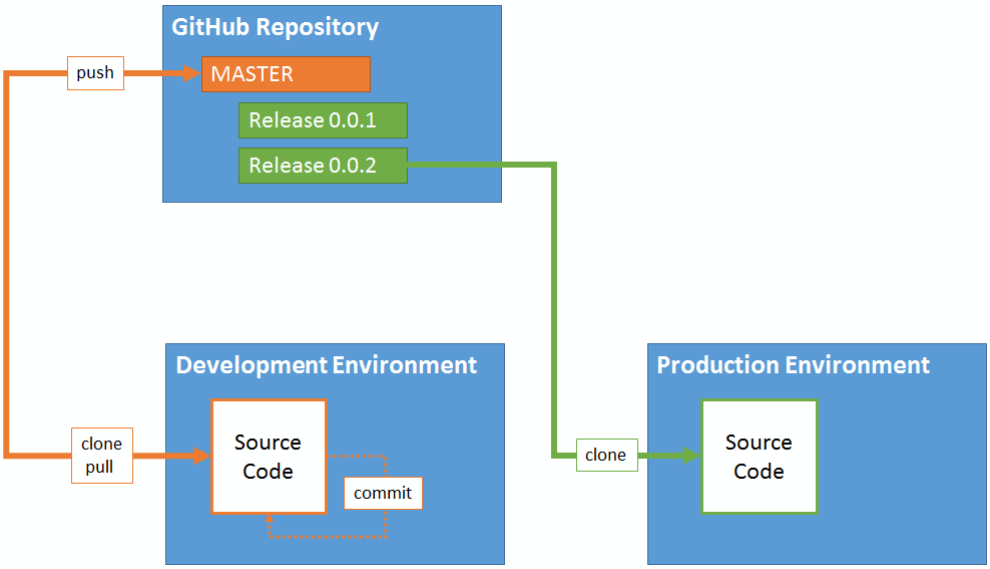

Figure 9: Deployment overview

`github.com/etops77/klettertracker`. As the repository is not publicly available, please contact Daniel.Frech@edu.fh-joanneum.at or etops77@gmail.com if you want access.

The following chapter will take everything learned from developing the prototype application with the MEAN stack and compares the development process to that with two other frameworks.

Chapter 4

Comparison

4.1 Overview

Comparing web development frameworks is, due to the complexity and diversity of the tools, certainly not an easy task. When adding the numerous different requirements of projects and the varying team member development skills, this almost becomes daunting. Nevertheless, this thesis has taken up the challenge and compares two other frameworks to the MEAN framework in five different categories. The goal is not only to find a winner of the comparison, but to also give some grounded insight into the differences of the frameworks. As a result, this thesis should be a guidance for developers, to select the right tool for their next projects.

4.2 Comparison criteria

The comparison focuses on five different categories. In every comparison, each framework can achieve from one to a maximum of 5 points.

Comparison 1: Rapid application development

What is needed to get going with the framework? The first comparison will take a close look at the ability of each framework to quickly develop an application from scratch.

Comparison 2: Development environment

The second comparison focuses on the development process. What support do Integrated Development Environment (IDE)s or user interfaces offer for features such as code completion, configuration or deployment? How much support can be found from the framework documentations, books and other sources on the internet? And how active is the community around these frameworks?

Comparison 3: User experience

The design and features of applications are dependant of the features the frameworks offer. This comparison takes a look at how applications that are built with the contesting frameworks look and feel, how well they support modern web development standards and to what extend the design can be changed.

Comparison 4: Change- and testability

When designing web applications that should last, the change- and testability of these applications is an important factor. This comparison identifies the tools that the frameworks offer developers for these tasks and how tightly the aspect of testing is integrated within the frameworks. In addition, the tools for debugging errors are investigated.

Comparison 5: Deployment

The fifth comparison focuses on the deployment process of the different frameworks. Besides the ease of deploying a web application, the separation of the different environments like development, testing or production is in focus of this comparison.

4.3 Weighting

Every project is different and has varying requirements. Therefore each part of the comparison can be given a certain weighting factor. This factor will at the end be multiplied with the result of the comparison in order to achieve a final result. For the scope of this thesis, a sample weighting has been chosen, based on the prototype application.

Comparison	**Weighting**
Rapid application development	0.5
Development environment	1.5
User experience	2.0
Change- & testability	1.0
Deployment	1.0

Table 2: Weighting for prototype application

4.4 The contestants

For the scope of this comparison, the Django framework and Oracle Application Express have been chosen as the contestants. Oracle Application Express is quite in contrast to the MEAN framework. It fully lives in the relational world of the Oracle database and is often selected in for enterprise applications. Django on the other hand has become very popular a few years back already, in combination with

the rise of another framework, Ruby on Rails. More recently built frameworks like Play or in this case MEAN are build on top of many things that have first been introduced with frameworks like Django. Oracle Express and Django therefore make up two quite different contestants for the MEAN stack.

The MEAN stack is a full stack web application development framework. This means that all major parts, from the database to the web server and to the user interface are included in the framework. The two contestants in contrast, do not have all those components included. However, they are usually used with a certain set of components or are even bound to certain components. This comparison will look at the contestants, combined with this set, as described in the following overview.

4.4.1 Django

Django calls itself on its website

> ***The web framework for perfectionists with deadlines.***
> (DjangoSoftwareFoundation (visited on 21.10.2015))

This nicely reflects the reason, why Django has been developed in the beginning. To save time, in 2003 Adrian Holovaty and Simon Willison, who worked for the Lawrence Journal-World newspaper in Kansas, factored out common modules and tools that they needed for their various news sites, into something called "The CMS". What began as an internal framework in order to save time and meet deadlines, eventually was open-sourced in 2005 under the name "Django" (cf. Ravindran (2015), p. 3). Some years and lots of changes and additions later, Django still reflects the spirit of its creation.

Django is built with Python and with that, lots of things that make Python the exiting language that it is, can be found in Django as well. Pythons philosophy is best documented in its gurus, Tim Peters, principles, that guided the development

process of Python itself. They have been so influential to programmers around the world, that they are immortalized as Python Enhancement Proposal (PEP) 20, called "The Zen of Python" and in the Python distribution itself as the easter egg module called "this" (cf. Alchin (2013), p. 1):

```
>>> import this
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

(PythonPEP20 (visited on 25.10.2015))

One, a bit confusing detail of the Django framework, is its leaner interpretation of the MVC pattern, called Model-Template-View (MTV). The separation of concerns in Django is between the database interfacing model ("Model"), the request processing classes ("View") and a templating language for the final presentation ("Template") . Compared with the MVC pattern, the "Model" is comparable to the model in Django. The "View" is found in Djangos templates and the "Controller" is the framework itself that processes incoming HTTP requests and

routes it to the correct view (cf. Ravindran (2015), p. 8).

A central aspect of Django's slightly modified form of the MVC architecture is the notion, that sections of code that perform different functions should not rely on how the others operate. This loose coupling is in contrast with tight coupling, where modules rely on the internal details of other modules' implementations (cf. Alchin (2013), p. 4).

In summary, today Django is a modern web application framework designed to build complex web applications quickly and without any hassle (cf. Dickey (2014), p. vii).

4.4.2 Oracle Application Express

Oracle Application Express (APEX) is quite in contrast to the other two contestants of the comparison. With the MEAN stack and Django, all components of the framework are open-source. This is not the case with APEX. The web application development framework itself does however not incur any license fees. But costs may accumulate for the database component, as only Oracles Express Edition (XE) with limited features and storage space is without charge. Further, APEX is the only framework that is bound to one specific database, Oracle.

Also, Oracle APEX is very close to the database. In fact it is built with its core language SQL and PL/SQL and the program itself resides within the database. With that, developers with knowledge of these languages can very quickly become productive with the framework (cf. Gault et al. (2013), p. 1), even if they are rather inexperienced in web development with HTML, CSS, JavaScript and web centric frameworks like JQuery.

In addition APEX provides a clear and powerful web interface, that is used for the development process. For building sites, APEX also provides several wizards, that guide developers through the process and present pluggable components for

reports, charts, fields or buttons.

For the web server part, Oracle offers different possibilities that are thoroughly described on the APEX installation documentation (see ApplicationExpressInstallationGuide (visited on 25.10.2015)). The exact details of the different options are out of scope of this comparison. However, APEX is bound to the options described there, which are all part of the Oracle database ecosystem. Again, it is not possible to replace that part of the framework with some other web server.

In conclusion, Oracles Application Express is a framework that is fully integrated with the Oracle database and offers a tool to rapidly build web applications.

4.5 Comparison 1: Rapid application development

What is needed to get going with the framework? How quickly can a new application be developed from scratch, independent from existing knowledge? How many different technologies has someone who is new to the framework have to learn? The first comparison answers these questions for each of the three contesting frameworks.

Django

Django is written in Python, so anybody new to Django has to learn the programming language up front. Especially the code formatting style in Python, where line breaks and intentions control the program flow instead of brackets and delimiters like semi-colons, can feel a bit strange at the beginning. Using an IDE like Exclipse or PyCharm helps to alleviate the programming flow.

Python offers numerous packages that can enhance or modularize certain functionality, either for Python or Django. The packages can best be handled with

Pythons package manager called Python Package Index (PIP). In addition, the installation of the Python package "virtualenv" helps to handle dependencies and versions of the installed packages. As working with this package can be a bit difficult, another add-on package called "virtualenvwrapper" makes working with the virtual environments much more pleasant.

Django itself does not offer comprehensive functionality for scaffolding a new application, like Ruby on Rails does for example. The basic structure of an application that includes the major parts can however be scaffolded. Alternatively, a more sophisticated project template that can be found at the GitHub project "django-twoscoops-project", can be used for the project initialization and gets a new project started quickly (cf. Greenfeld & Roy (2013), p. 21 and django-twoscoops project (visited on 25.10.2015)).

Django and Python also include a text-based database called SQLite that can be used for new projects out of the box. While not a preferred option for production, it still offers the possibility to quickly get started with a new application. All created models can also be quickly included in the Django admin site. This allows the creation of data through the web interface directly. While the Django admin has a bit of an outdated User Interface (UI), this can easily be bootstrapped with Python packages like "django_admin_bootstrapped".

Result: **4 points**

Once familiar with the Python language, everything is quickly installed and with Python / Django a lot can be achieved in short time. The included admin interface also helps to quickly create data for the application. However, missing scaffolding options, as well as the a bit clumsy handling of the virtual environments, make up a total of 4 points for Django in this category.

APEX

Oracles Application Express is based on SQL and PL/SQL and fully integrated with the database. That means, that at least some knowledge of SQL and PL/SQL is essential when starting application development with APEX.

If this is the case, APEX is a breeze for rapidly getting results. For example, the Oracle XE database already includes a fully functional APEX environment upon installation. Alternatively, the Oracle cloud offers a free workspace for developers that can be used for development at the start.

The rich web user interface of APEX offers wizard like procedures for creating components of a page in an amazingly short time. For example, an interactive report based on a table or SQL statement is fully functional within seconds, including filtering, row highlighting or exporting of data. Predefined themes apply a modern look and feel. The new version 5 of APEX takes this even one step further, by providing a very modern looking and flexible theme called "Universal Theme".

Result: **5 points**

The possibility to create results within minutes, even with limited knowledge of web development, are unmatched. This does require knowledge of SQL and PL/SQL, but knowledge of some kind of programming language is needed in any of the frameworks of course. Therefore the result is the maximum amount of points.

MEAN

The MEAN framework template includes a lot, almost everything needed. Node.js, MongoDB, Bower and Grunt need to be installed up front, but after that the following steps are sufficient to build and start a new project with the MEAN.JS template:

```
# Cloning The GitHub Repository
$ git clone https://github.com/meanjs/mean.git meanjs
$ git checkout v0.4.0

# Dependency install
$ npm install

# Running the application
$ grunt
```

The template also includes a fully functional login process and a boilerplate welcome page. After the initial installation, Create-Read-Update-Delete (CRUD) models, AngularJS models or other parts of the application can be easily scaffolded by using the Yeoman generator.

However, the extensive functionality results in an extensive amount of components. Getting an overview of what part is responsible for what and how things are connected is not easy. Adding custom functionality and changing existing, generated components is therefore at the beginning not an easy task. Also changing scaffolded CRUD models can be challenging and time consuming at the start.

Result: **3 points**

While providing tons of functionality out of the box, due to its complexity it takes longer to get productive and produce results with the MEAN framework, than with the other two contestants. Therefore only three points for the MEAN framework in this category.

4.6 Comparison 2: Development environment

The second comparison focuses on the development process. What support do IDEs or user interfaces offer for features such as code completion, configuration or deployment? How much support can be found from the framework documenta-

tions, books and other sources on the internet? And how active is the community around the frameworks?

Django

Development for a Django project can be conducted solely with a text editor like Sublime for example. However, using an IDE greatly affiliates the process. Some of the populare IDEs for Python and Django development are PyCharm, WingIDE and the Eclipse plug-in PyDev. Especially PyCharm does work like charm when developing a Django application, as it includes lot of functionality especially targeted on the Django framework like starting the development server or collecting static files.

The documentation of Django is exceptionally good for an open source project. Only add-on packages sometimes lack a proper documentation, but these should probably best be avoided anyway.

The numerous Python and Django add-on packages, as well as tons of answered questions on Stack Overflow indicate a vibrant and active community.

Result: **5 points**

In regards to the documentation and development environment, nothing is really missing when developing a Django application. Therefore 5 points for Django in this category.

APEX

As a result of being integrated in the Oracle environment, the documentation of APEX is thorough and detailed in every facet. Also the community is very active, as can be seen in the Oracle user groups around the world.

Besides that, with APEX there is no need for an IDE. The web interface that is

used for the development process is almost flawless and makes it a breeze to get started.

Result: **5 points**

APEX is the only fully company backed framework in this comparison. As a result, no negative points could be discovered in this comparison, leading to the full amount of points.

MEAN

Because of the strong modularization of code with MEAN apps, using an IDE is highly recommended. WebStorm from JetBrains for example offers great support of the included technologies. Grunt tasks and Bower or NPM installations can for example directly be initiated from within the IDE.

In regards of documentation, both Node.js and AngularJS have extensive documentation available. Only the documentation of the combined MEAN framework is as substantial as it could be. Several books have been published regarding this topic recently, some of them can be found in the bibliography of this thesis.

The community around the technologies of the MEAN stack is vibrant and active. Especially AngularJS, that is maintained by Google, received a lot of attention recently. A minus is however the somewhat vague situation with the different MEAN providers, MEAN.io and MEAN.js, as described in section 2.6 on page 20. A fork of the Node.js framwork into IO.js in 2014 that received a lot of attention, has shortly afterwards been re-merged into the Node.js project.

Result: **4 points**

The only contestant that misses one point in this comparison is the MEAN stack. This is due to the somewhat brief documentation of the combined MEAN framework, as well as the slightly unclear situation of the different MEAN providers.

4.7 Comparison 3: User experience

The design and features of applications are dependant on the features the frameworks offer. This comparison takes a look at how well the different frameworks support modern web development standards, to what extend the design can be changed and how the end-user experience is for applications that are built with each framework.

Django

Django makes it easy to add third party libraries for an application. Also the popular package manager Bower can be used by installing the Python package "django-bower". That way, almost anything that is new and shiny on the internet can be used within a Django project.

In addition, numerous other Python packages provide tons of functionality that can enhance Django projects. To only name two useful examples, the package "Sphinx" adds lots of functionality for documenting the application and the package "South" greatly helps with database migrations when models needs to be changed.

Asynchronous code can easily be included by using JQuery in the project for example. However, Django relies more on the concept of separate routings and server requests for each part of the application. SPAs that should behave like a native application with no full page refreshes, are therefore not a natural fit for Django applications. However, there is a lot of functionality to build REST endpoints with Django. With the addition of another framework like BackboneJS or AngularJS, also Django could be used for such an application.

Result: **4 points**

In conclusion, there is not much that is missing to create a great user experi-

ence with Django. Only the tendency to create sites that do lots of full page reloads costs the framework one point, even though this could be compensated by bringing in an additional framework for the front-end.

APEX

Like applications that fully rely on Twitters bootstrap standard components, APEX applications can often quickly be identified. First of all because of the standard themes that come with APEX and which are often used. Changing these themes is possible, even new themes can be created. However, themes are stored in form of PL/SQL procedures and contain different regions and components like report templates. For a new theme to replace an existing one, it needs to have the same regions and components included. If developers have created custom components, replacing a theme can therefore easily become a rather time consuming process.

Web frameworks such as JQuery or plug-ins like sliders or calendars can be included by adding the reference in the page template. When using frameworks like JQuery, there is an important distinction in the state of variables in APEX, that has to be kept in mind. Changing a variable with JQuery does not change the application state of the variable in APEX. This is quite in contrast to the concept of two-way data binding in AngularJS, that has been described in section 2.4 on 15. As a result, a consequent action like a submit might still have an outdated value of the variable, although it is displayed differently on the page. Therefore dynamic actions, an APEX specific feature for handling dynamic and asynchronous tasks, need to be used most of the time in conjunction with JQuery or JavaScript calls. Instead of one JQuery action, these dynamic actions then contain multiple actions to also update the application state.

Another way that an APEX application can be recognized is the URL. A default APEX URL looks like this:

http://www.sampleapp.com/apex/f?p=509:1:

In this case, the number 509 would reflect the application number and 1 the application page. The look of the URL cannot be changed in APEX itself, only with a URL rewrite in a web server like Apache that is placed in front of the APEX application.

On more thing to note with APEX is, that third-party web libraries always need to be manually included in a page or a template. There is no built-in tool like Bower that can be used for the import.

Result: **3 points**

While libraries and plug-ins can be included in APEX to create a rich and modern look and feel, changing themes and templates can become a challenging task. In addition, the difference between the application state of variables and the state of the variables in the DOM, makes asynchronous programming more complex. This can lead to applications with more frequent complete page refreshes than needed.

MEAN

With Bower included as a web package manager, Grunt for automatic tasks like source file compression and of course the AngularJS framework, there is nothing missing to create a fantastic, feature rich user experience. With the technologies included in the MEAN stack, especially SPA applications are a natural fit for the framework. The responsiveness of such applications are indeed unmatched. Especially the concept of two-way data binding offers lots of possibilities for developing applications that dynamically load a lot of its content.

And if anything should be missing on the server side, NPM has numerous packages that can enhance the functionality even further. With one of these packages that is already included, "passport", adding support for logging in with social-media

accounts is a breeze.

Result: **5 points**

The maximum number of points go to the MEAN framework in this comparison. Almost everything needed to create responsive and great applications is on board, the rest can easily be added by utilizing NPM packages or linking libraries with Bower.

4.8 Comparison 4: Change- and testability

When designing web applications that should last, the change- and testability of the application is an important factor. This comparison identifies the tools that the frameworks offer developers for these tasks and how tightly the aspect of testing is integrated within the frameworks. In addition, the tools for debugging errors are investigated.

Django

From Django version 1.6 on, the Python package "django-discover-runner" is included in the framework. Without this package, tests needed to reside in the same folder as the tested module. With this package, tests can now be separated. As a result, it is easily possible to create thorough tests of the Django applications. In addition packages for monitoring code-coverage, like "coverage.py" greatly helps to manage tests. The testing however only concerns the Python part of the Django applications. Add-ons to create a rich user-experience with JQuery for example are not affected by these tests and need to be separately included, like with QUint for JQuery.

Another add-on package called "django-debug-toolbar" greatly helps with the otherwise a bit difficult task of debugging. The debugging tool-bar can however

not be activated on the fly for a production environment. In this case, the web console and server logs are the only means of debugging.

Result: **4 points**

A lot is included to do proper testing, maintain a functioning application and to debug errors. In order to achieve a full test set, the frameworks functionality needs to be enhanced however, by adding additional Python packages and frameworks for testing dynamic functionality.

APEX

Quite surprisingly, APEX does not offer any kind of tools for testing. With that, the only possibilities for testing are either tests in the database or user interface tests, like Selenium. Database tests do in consequence require the APEX application to heavily make use of packages, procedures and functions, instead of directly referencing tables or views, as only packages, procedures and functions can be tested within the database. No testing tools make it needless to speak about test coverage tools of course.

The lack of testing capabilities also affects the changeability of APEX applications. Especially dynamic actions with multiple included actions can become difficult to maintain. Changes can then easily break the programmed logic. In addition, the possibility to place CSS or JavaScript functionality in lots of different places, are a strain on maintenance. For example, the styling of an element in APEX can be put in a separate CSS file, can be added as an included script in the page definition, can be included with the theme or template configuration, or can be placed directly in the attribute field of the element.

With no testing tools available, debugging will likely become important at a certain step. Here, APEX does include several integrated debugging functionalities, that can help to detect errors.

Result: **2 points**

The lack of testing capabilities and the possibility to spread logic in different places does cost APEX points in this comparison. Only the debugging functionality offers sufficient support for detecting errors.

MEAN

The aspect of testing is deeply integrated within the MEAN stack. The Node.js side of the application can be tested by using the NPM package "Mocha". The in the MEAN stack included Grunt tasks can automatically run JSHint code checks, that verify the consistent syntax of the JavaScript Code.

On the AngularJS side of the application, the testing tool "Karma" allows the thorough testing of the AngularJS modules. The needed packages are already included with the set-up of a MEAN project. Code coverage can be added for Mocha and Karma with separate packages (eg. "istanbul" for Mocha) or plug ins (eg. "karma-coverage" for Karma). And last but not least, a tool called "Protractor" can be used for combined end-to-end testing.

Debugging of the AngularJS module takes place directly in the developer tools of the browser. In addition, there are tools like "AngularJS Batarang" or "ng-inspector" for Google Chrome and "AngScope" for Firefox, that help with the debugging process and give the opportunity to inspect the AngluarJS scope. For Node.js there are also several debugging tools available, like the popular NPM package "node-inspector" that includes a web interface for debugging.

Result: **5 points**

There are so many tools available for testing and debugging, that it can even be a bit overwhelming at the beginning. That is not a reason of course to subtract a point, therefore again the maximum number of points go to the MEAN stack.

4.9 Comparison 5: Deployment

The last comparison focuses on the deployment process of the different frameworks and examines the possibility to extend web applications that are built with one of the frameworks. Besides the ease of deploying a web application, the separation of the different environments like development, testing or production is in the focus of this comparison.

Django

Django's primary deployment platform is the Web Server Gateway Interface (WSGI), which is the Python standard for web servers and applications. Possible deployments with WSGI would be with Apache mod_wsgi, with Gunicorn or with uWSGI. With all deployments no cost will incur, therefore only an appropriate server, preferably with a Linux distribution needs to be selected. Alternatively to WSGI, also deployments with FastCGI, SCGI or AJP are possible.

For cloud deployment, Heroku offers the possibility to run Django applications. With so called "Dynos", the needed performance can easily be adjusted. Another option would be using a service like Digital Ocean. Here, servers that are called droplets, offer completely pre-configured virtual environments for Django applications, where basically only the project code needs to be imported.

Django also offers configuration files for different environments by default. That makes it easy to separate the needed components and settings. The only thing to keep an eye on is the integration of Python packages that have dependencies on components of the Operating System (OS). An example would be the image library PIL, that needs several components of the OS in order to be installed successfully. Including such a package can cause problems when deploying to a cloud environment.

Result: **5 points**

Django offers several deployment options starting at small costs, combined with a clear separation of configuration settings for different environments. Combined, this results in the full amount of points in this category.

APEX

When deploying an APEX application, the whole application needs to be exported. This creates a SQL file that can afterwards be imported in a different environment. Unfortunately, application components such as CSS or JavaScript files that are stored in APEX shared components, are not included with the export. Such components should therefore rather be included by file reference to a location on the web server or a Content Delivery Network (CDN). That way, all static files can be taken from central location and no separate exports are required.

For deployment itself, an APEX application does require a fully configured Oracle database. Besides the option to use a private server, Oracle also offers the possibility to host the application in the Oracle cloud. Setting up and maintaining an Oracle database server certainly requires advanced skills, the hosting option alleviates this process.

The process of separating the environments is not integrated within the application. This needs to be done with the server set-up.

Result: **3 points**

There are a few options for deployment, however APEX is always bound to the Oracle database environment. Additionally, the handling of applications requires special considerations and environment settings are not clearly separated. As a result, only three points for APEX in this category.

MEAN

The deployment process of MEAN applications is quite similar to the one of Django. There is just no need to select a web server, as Node.js together with Express.js are the web server. Besides the possibility of running the MEAN app on a server, also hosting options like Heroku or Digital Ocean apply for MEAN applications.

The configuration files also differentiate between the different environments and allow a clear separation.

Result: **5 points**

Similar as with Django, the MEAN stack offers a lot of possibilities for deploying applications and makes a clear distinction between the different environments. In conclusion, no reason to subtract a point.

4.10 Combined result

Below tables show the results of the three frameworks for the five categories, in combination with the chosen weighting for the prototype application. First of all **Django**, who showed no real weaknesses and scored quite evenly in all categories.

Comparison	**Result**	**Weighting**	**Final result**
Rapid application development	4.0	0.5	2.0
Development environment	5.0	1.5	7.5
User experience	4.0	2.0	8.0
Change- and testability	4.0	1.0	4.0
Deployment	5.0	1.0	5.0
Total	**22.5**		**26.5**

Table 3: Final result Django

The second contestant, **APEX** did perform very well in the first two categories. But especially the lack of testing capabilities did cost the top spot at the end.

Comparison	**Result**	**Weighting**	**Final result**
Rapid application development	5.0	0.5	2.5
Development environment	5.0	1.5	7.5
User experience	3.0	2.0	6.0
Change- and testability	2.0	1.0	3.0
Deployment	3.0	1.0	3.0
Total	**18.0**		**21.0**

Table 4: Final result APEX

Last but not least, the **MEAN** framework and center of this thesis showed its strengths especially in the user experience, change- and testability and deployment categories.

Comparison	**Result**	**Weighting**	**Final result**
Rapid application development	3.0	0.5	1.5
Development environment	4.0	1.5	6.0
User experience	5.0	2.0	10.0
Change- and testability	5.0	1.0	5.0
Deployment	5.0	1.0	5.0
Total	**22.0**		**27.5**

Table 5: Final result MEAN

While Django and the MEAN stack are heading for a head-to-head race, the weighting for the prototype application does indicate the MEAN stack as the winner of this comparison, if only by a small margin.

Chapter 5

Conclusion

During the development of the prototype application, the MEAN stack has proven to be an interesting, feature-rich and exiting framework for building expressive and modern web applications. Indeed, as chapter 2 on page 6 showed, in MEAN things are done differently than with other frameworks. Combined, the four technologies work together well and the MEAN stack is a valuable alternative when developing web applications.

For the scope of this comparison, the MEAN stack also performed exceptionally well. With focus on the requirements of the prototype application, it even won the comparison. Developing modern web applications however has so many different facets, that several additional categories could be investigated that have not been in the scope of this comparison. Looking at the security aspects of the different frameworks for example, or the ability to scale out on high demand might then result in a different winner.

And of course there are many more contestants out there, all competing with the ones in this comparison. The Play framework, based on Scala and Java, PHP frameworks like Zend and many more.

Nevertheless, in retrospect, the MEAN stack is certainly worth a closer look,

when a new project requires the development of a modern and user centred web application.

Appendix A - MEAN resources

The documentation for the MEAN.JS web application framework can be found at http://meanjs.org/. To learn more about the technologies used in MEAN:

MongoDB

- MongoDB documentation: https://docs.mongodb.org/manual/
- MongoDB tutorial: http://www.tutorialspoint.com/mongodb/
- Mongoose documentation: http://mongoosejs.com/

Express.js

- Express.js documentation http://expressjs.com/api.html
- Express routing: https://scotch.io/tutorials/learn-to-use-the-new-router-in-expressjs-4

AngularJS

- AngularJS tutorials: https://docs.angularjs.org/tutorial
- Free AngularJS courses: http://campus.codeschool.com/courses/shaping-up-with-angular-js/intro
- Learn Angular: http://www.learn-angular.org/

- A better way to learn AngularJS: https://thinkster.io

Node.js

- Node.js documentation: https://nodejs.org/en/docs/
- Node school: http://nodeschool.io/

Appendix B - JQuery/AngularJS example

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>JQuery - AngularJS comparison</title>
5    <meta http-equiv="Content-type" content="text/html;charset=UTF-8">
6
7      <!--framework loading-->
8      <script src="https://ajax.googleapis.com/ajax/libs/angularjs
       /1.4.7/angular.min.js"></script>
9      <script src="https://ajax.googleapis.com/ajax/libs/jquery
       /1.11.3/jquery.min.js"></script>
10
11 </head>
12 <body>
13
14     <!--JQuery message-->
15     <h1>JQuery</h1>
16
17     <div>
18         <input type="text" id="jqueryInput" placeholder="Enter a
       message here">
19         <h2>Message: <span id="jqueryMessage"></span></h2>
20     </div>
21
22     <hr>
```

```
23
24      <!--AngularJS message-->
25      <h1>AngularJS</h1>
26
27      <div ng-app>
28          <input type="text" ng-model="angularjsMessage" placeholder="
        Enter a message here">
29          <h2>Message: {{angularjsMessage}}</h2>
30      </div>
31
32      <!--JQuery event listener -->
33      <script>
34          function updateJQueryMsg(){
35              $( "#jqueryInput" ).keyup(function() {
36                  $("#jqueryMessage").text($( "#jqueryInput").val());
37              });
38          }
39          $( document ).ready(function() {
40              updateJQueryMsg();
41          });
42      </script>
43  </body>
44  </html>
```

List of Figures

Bibliography

Alchin, M. (2013), *Pro Django*, Apress, New York.

ApplicationExpressInstallationGuide (visited on 25.10.2015), 'Application express installation guide', `https://docs.oracle.com/cd/E59726_01/install.50/e39144/toc.htm`.

Banker, K. (2015), *Mongodb in Action*, 2nd edn, Manning Publications Company, Birmingham.

Brežnjak, N. (visited on 23.10.2015), 'Mean.io vs mean.js and deploying the latter on digital ocean', `https://hackhands.com/mean-io-vs-mean-js-deploying-latter-digitalocean/`.

Cantelon, M., Holowaychuk, T. J. & Rajlich, N. (2013), *Node.js in Action*, pap/psc edn, Manning, Birmingham.

Casciaro, M. (2014), *Node.js Design Patterns*, Packt Publishing Ltd, Birmingham.

Chodorow, K. (2013), *MongoDB: The Definitive Guide*, 2nd edn, O'Reilly Media, Inc., Sebastopol.

Dayley, B. (2014), *Node.js, MongoDB, and AngularJS Web Development*, 1st edn, Addison-Wesley Professional, Boston.

Dickey, J. (2014), *Write Modern Web Apps with the MEAN Stack - Mongo, Express, AngularJS, and Node.js*, 1. aufl. edn, Peachpit Press, Berkely.

django-twoscoops project (visited on 25.10.2015), 'django-twoscoops-projectn', `https://github.com/twoscoops/django-twoscoops-project`.

DjangoSoftwareFoundation (visited on 21.10.2015), 'Mongodb manual', `https://www.djangoproject.com/`.

Freeman, A. (2014), *Pro AngularJS*, 1st edn, Apress, New York.

Gackenheimer, C. (2013), *Node.js Recipes - A Problem-Solution Approach*, Apress, New York.

Gault, D., Cannell, K., Cimolini, P., D'Souza, M. & Hilaire, T. S. (2013), *Beginning Oracle Application Express 4.2*, 2nd edn, Apress, New York.

Greenfeld, D. & Roy, A. (2013), *Two Scoops of Django - Best Practices for Django 1.5*, 1st edn, CreateSpace Independent Publishing Platform, Ort.

Karpov, V. & Netto, D. (2015), *Professional AngularJS*, 1st edn, John Wiley and Sons, New York.

Liu, J. (visited on 23.10.2015), 'Node.js processing model', `https://www.processon.com/view/531abf1e0cf2e60fdb725cda`.

MongoDBJsonBson (visited on 21.10.2015), 'Mongodb manual - json and bson', `https://www.mongodb.com/json-and-bson`.

MongoDBReplication (visited on 21.10.2015), 'Mongodb manual - replication introduction', `https://docs.mongodb.org/manual/core/replication-introduction/`.

MongoDBSharding (visited on 21.10.2015), 'Mongodb manual - sharding introduction', `https://docs.mongodb.org/manual/core/sharding-introduction/`.

nodejs.org (visited on 21.10.2015), 'Node.js project page', `https://nodejs.org/en/`.

Ocariza, Jr., F. S., Pattabiraman, K. & Mesbah, A. (2015), Detecting inconsistencies in javascript mvc applications, *in* 'Proceedings of the 37th International Conference on Software Engineering - Volume 1', ICSE '15, IEEE Press, Piscataway, NJ, USA, pp. 325–335.
URL: *http://dl.acm.org/citation.cfm?id=2818754.2818796*

PythonPEP20 (visited on 25.10.2015), 'Pep 20 – the zen of python', `https://www.python.org/dev/peps/pep-0020/`.

Ravindran, A. (2015), *Django Design Patterns and Best Practices*, Packt Publishing Ltd, Birmingham.

Sadalage, P. J. & Fowler, M. (2012), *NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence*, 1st edn, Addison-Wesley, Amsterdam.

Wandschneider, M. (2013), *Learning Node.js - A Hands-On Guide to Building Web Applications in JavaScript*, 1st edn, Addison-Wesley, Amsterdam.

Wilson, J. R. (2013), *Node.js the Right Way - Practical, Server-Side JavaScript That Scales*, 1st edn, Pragmatic Programmers, LLC, Dallas, Texas.

Yaapa, H. (2013), *Express Web Application Development*, Packt Publishing Ltd, Birmingham.