



Technische  
Universität  
Braunschweig



Institut für Nachrichtentechnik

Bachelor Thesis

# Development of an Administrative Web Frontend for Deep Learning Research

Lukas Güldenhaupt

Matrikelnummer: 4571429

01.03.2017

Technische Universität Braunschweig  
Institute for Communications Technology  
Schleinitzstraße 22 – 38106 Braunschweig

Prüfer: Prof. Dr.-Ing. Tim Fingscheidt  
Betreuer: Samy Elshamy, M.Sc.

## Erklärung

Hiermit versichere ich die vorgelegte Bachelor Thesis zum Thema

**„Development of an Administrative Web Frontend for Deep Learning Research“**

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

---

Hiermit versichere ich an Eides statt, dass ich die vorliegende Bachelorarbeit zum Thema „Development of an Administrative Web Frontend for Deep Learning Research“ selbstständig und nur mit den angegebenen Quellen und Hilfsmitteln erstellt habe.

---

Braunschweig, den 01.03.2017

---

Lukas Güldenhaupt

## **Preface (optional)**

A preface is optional and can come here.

## **Abstract**

**Keywords:**

# Contents

<b><u>Content</u></b>	<b><u>Page</u></b>
Erklärung.....	2
Preface (optional) .....	3
Abstract .....	4
Contents.....	5
1 Architecture .....	7
1.1 Server-side.....	7
1.2 Client-side .....	7
1.3 Database .....	8
2 Software Design .....	9
2.1 Typescript.....	9
2.2 Components.....	9
2.3 Data-handling .....	10
2.3.1 Data model.....	10
2.3.2 Data services.....	10
2.3.3 Observables .....	10
2.4 Authentication .....	10
3 Users perspective.....	11
3.1 Profile/Login .....	11
3.2 Projects .....	11
3.3 Configurations.....	11
3.3.1 Mappings .....	11
3.3.2 Filtering .....	11
4 Developers Perspective .....	12
4.1 Developer Tools .....	12
4.2 Setting up a development environment .....	12
4.3 Coding.....	12
4.3.1 Adding Components .....	12
4.3.2 Adding Collections .....	12
4.3.3 Working with Observables .....	12
4.3.4 Extending Functionality .....	12
4.4 Documentation .....	12
4.5 Deployment .....	12
A Appendix .....	13

Contents	6
B Bibliography .....	14
C List of Figures .....	15
D List of Tables.....	16
E Abbreviations (optional).....	17

# 1 Architecture

Since the basic idea of this tool is to give a lasting web frontend for the institute a good choice of what software to use is essential. Therefore, for client and server-side code we chose well maintained and well-known frameworks as there are Meteor as a mostly server-side JavaScript framework and Angular as a frontend JavaScript framework. With this an all in all forward-looking webpage is ensured. In this chapter we evaluate why the chosen software fits our purpose and how they work together. We could build the web server and client completely from scratch but Meteor and Angular provide an overall good structure and a solid base for further development. Furthermore, we chose MongoDB as a database.

## 1.1 Server-side

As mentioned before Meteor is our chosen framework for the server side. It is an opensource full-stack JavaScript platform for web, mobile and desktop development. The power of this platform is its fast learning curve, its usability for any device and its technology integration. What that means is that without knowing much about web servers you can easily create your own application. Meteor also is known for its compatibility since you can use it independently from the platform, no matter if its web, iOS, Android or desktop. In our case we use it as a webserver but with further development of the website it could be optimized for mobile devices or become an app itself if desired.

A big point for Meteor is that you can share code between server, client and the database, what accelerates the development process enormously. This is what makes our application very reactive. Meteor uses data on the wire, sending not HTML to the client but data which gets rendered directly on client-side. With that provided reactivity the client reflects the true state of the data in real time. In combination with our frontend framework Angular no page reloading is necessary to have the latest data.

Behind the easy-to-use platform lies a NodeJS-server. When deploying Meteor-code it generates a stand-alone Node application. And this is the only dependency it has which means everywhere you have NodeJS installed you can run your meteor application.

## 1.2 Client-side

On client-side we chose the JavaScript framework Angular in version 4, developed and maintained by Google. Angular makes client development across all platforms possible. It grants fast speed and good performance and allows us to extend the template language HTML with

our own written components. Nearly every web IDE supports Angular to give the user syntax-highlighting, code-completion and Angular-specific help. In our case it replaces the Meteor-standard blaze-templates. Meteor and Angular work perfectly together on various platforms, when displaying real time data and keeping the reactivity of our application on a very high level.

### 1.3 Database

We chose MongoDB as our database. It is a strong and popular no-SQL, document driven database. Even with large data sets it scales very well and provides a high performance. Unlike in SQL an entity is represented by a collection which contains documents as its entries. A document is very similar to a JSON-Object and can be easily read and modified. Thanks to the flexibility of MongoDB we can design our collections freely and edit them with small effort, without losing our existing data. We can define the basic structure of a document and shape the rest of it as we want to.

This feature comes in handy when we have very variable data entries. In our case it does not matter how a given configuration file generated by one of many neural network programs looks like. With MongoDB we could insert the data without adjusting it to match a pattern.

Another big point of using MongoDB is that you have a decreased learning curve. Making queries is easy to understand and use. The necessary concept of using foreign keys to connect documents is also featured as every entry has its own unique id.



## 2 Software Design

In this chapter we explain how our web-application is generally structured, how and where the different tasks are handled. Certain constructs are set by the technology we are using however there are a lot of conceptual thoughts to be made. For example, how the code should look like to be intuitive on the one hand and compact at the other. Very important is the fact, that the progression does not have to be finished with the work of this bachelor thesis. The application is build and meant for further development. Therefore, a good documentation and clear project structure is helpful.

At first, we declare the design choices made on the client-side of our application. Later, we get to the data-handling and the server-side structures.

### 2.1 Typescript

Both our client and our server almost fully consist of JavaScript code. In fact, we use ECMAScript 5, which is a standardized version of it. The syntax of [JS](#) is similar to C or Java, which makes it easy to understand for everyone who has coded a bit. There is one big downside to it, being type insecure. Pure [JS](#) has no variable types. In ES5 to declare a variable, you can choose between *var*, *let* and *const* as a keyword. Each keyword has a different function or scope. To declare a variable globally you chose *var*, to declare a variable scoped between to curly brackets *let* should be used and to declare constant variables, that will or must not change *const* is the keyword to go. These keywords are helpful in some way but when it comes to huge applications with data-handling and complex functions a better approach is needed. Fortunately, Angular uses an extension called Typescript to ensure better coding qualities. Typescript compiles to JS what then can be seen in the browser. It supports definitions of classes, interfaces, generics, enumerations, inheritance, types of course and more useful features. With a package for Meteor we can write even our server code in typescript bringing this java like structure to the whole project. With Typescript the code is much more readable, clearer and java like.

### 2.2 Components

Angular offers a system to encapsulate logically independent code. These “blocks” of functionality are called “NgModules”. The root of our web application is the AppModule which contains all of our classes, services and helpers.

With Angular we can create UI blocks called components. In our code components are marked with the annotation `@Component()`. A component has a visual part and a logical part. You could say, that an Angular application is a tree of components. This could be a whole page, a table, even a text label or anything you want. Thanks to the independence of a component you can as many instances as you want anywhere in your application.

## **2.3 Data-handling**

### **2.3.1 Data model**

- uml diagram

- explain foreign ids

### **2.3.2 Data services**

- MongoDB queries

### **2.3.3 Observables**

- RXJS

## **2.4 Authentication**

## **3 Users perspective**

### **3.1 Profile/Login**

### **3.2 Projects**

### **3.3 Configurations**

#### **3.3.1 Mappings**

#### **3.3.2 Filtering**

## **4 Developers Perspective**

### **4.1 Developer Tools**

### **4.2 Setting up a development environment**

### **4.3 Coding**

#### **4.3.1 Adding Components**

#### **4.3.2 Adding Collections**

#### **4.3.3 Working with Observables**

#### **4.3.4 Extending Functionality**

-Npm packages

### **4.4 Documentation**

### **4.5 Deployment**

## **A Appendix**

## **B Bibliography**

**Im aktuellen Dokument sind keine Quellen vorhanden.**

## **C List of Figures**

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

## D List of Tables

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.



---

## E Abbreviations

---

### ***E***

ES5 · *ECMAScript 5*

ES6 · *ECMAScript 6*

---

### ***J***

JS · *JavaScript*