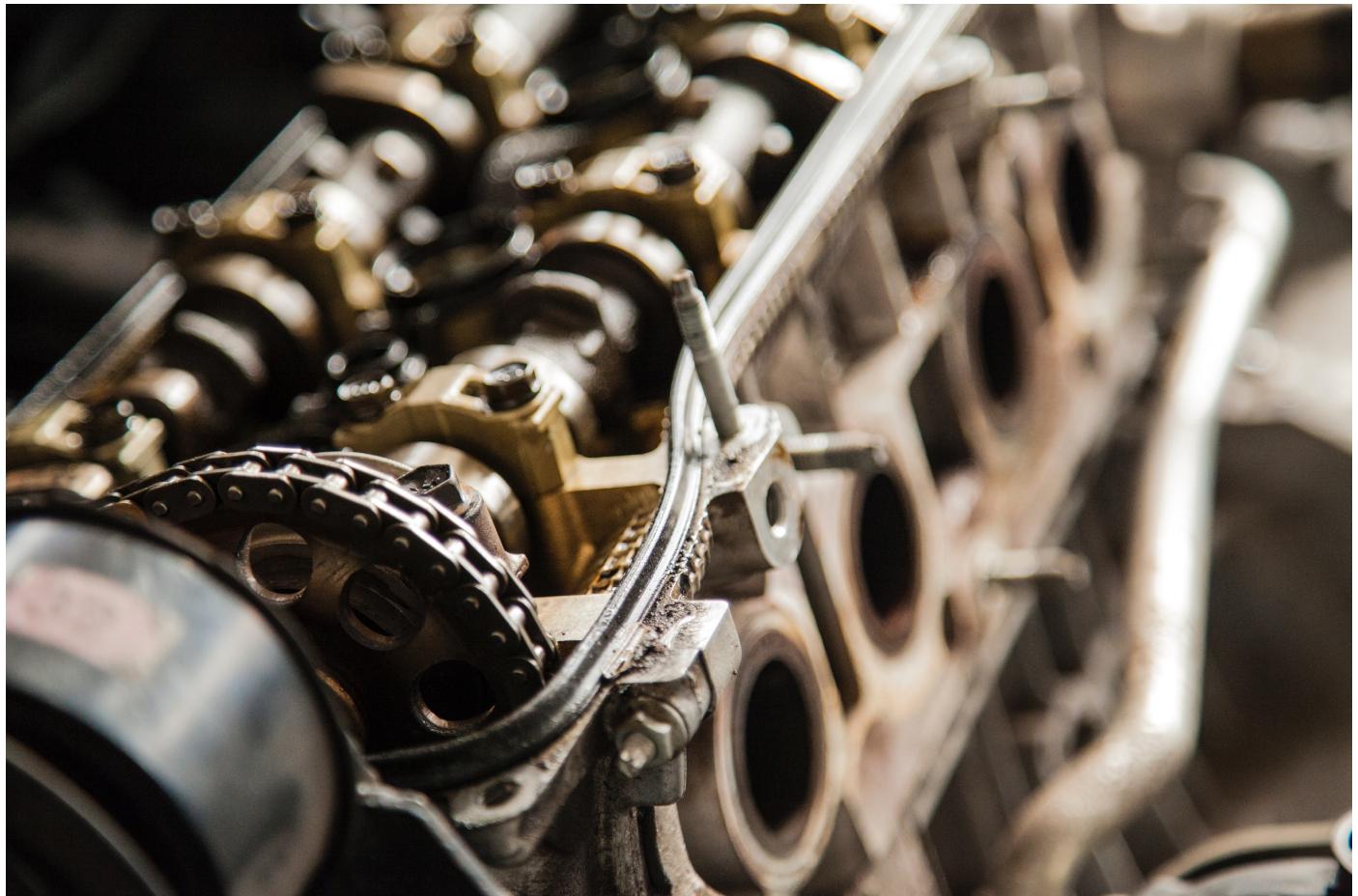


# MY\_BLOG

Ein Projekt von Fabian Bächli  
27.11.2017 – 28.01.2018



GitHub: [https://github.com/fabianbaechli/my\\_blog/tree/master](https://github.com/fabianbaechli/my_blog/tree/master)

# INHALTSANGABE

EINLEITUNG.....	3
DAS ENDPRODUKT.....	3
DER HEADER.....	3
DIE BLOG SEITE.....	4
DIE ADMIN SEITE .....	6
TECHNISCHES .....	7
FÜNF TIER ARCHITEKTUR .....	7
DREI-TIER.....	7
VIER-TIER.....	7
FÜNF-TIER .....	7
TRANSACTIONS.....	7
SUB-HEADER-SLIDER PROBLEMS.....	8
REFS.....	8
WRONG POSITIONING ON LOAD .....	8
DEBOUNCE .....	8
DIFFE-HELLMAN KEY-EXCHANGE.....	9
HOW IT WORKS.....	9
DESIGNEN DER PAGE .....	11
HOW TO RUN IT.....	12
SCHRITT 1 – DATENBANK AUFSETZEN .....	12
SCHRITT 2 – DOCKER CONTAINER ERSTELLEN.....	12
REFLEXION .....	13

## EINLEITUNG

Im Rahmen dieses Projekts habe ich einen persönlichen Blog realisiert. Für mich war hierbei wichtig, dass ich mein Können im Frontend-Bereich aufbessere. Dazu gehörte, ein neues Framework zu lernen; das Entwickeln von Web Applikationen ohne ein Framework wird ja zusehends weniger gewöhnlich.

Ich fokussierte mich im Zuge der Implementierung sowohl auf Faktoren, die für mich persönlich wichtig sind – Dinge wie Design, Kryptographie und React – wie auch auf die Modulanforderungen. Diese vielen und unterschiedlichen Aspekte machten dieses Projekt ein umfang – und lehrreiches, welches für mich als eins der besten, wenn nicht als das Beste eingeht.

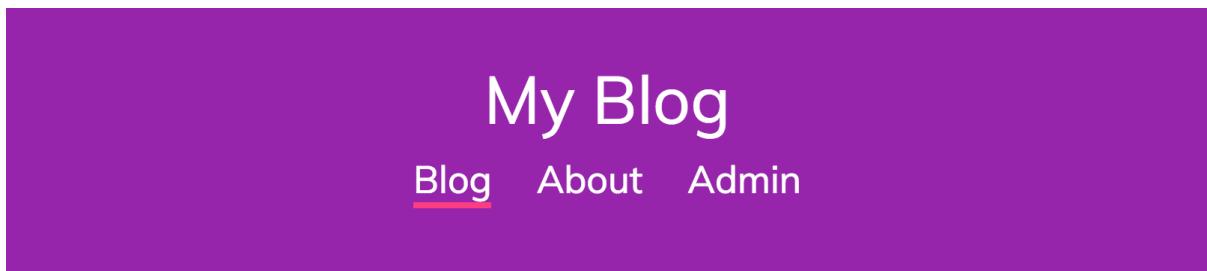
Von der Reihenfolge dieser Dokumentation her, werde ich Ihnen zuerst das Endprodukt näherbringen, mich danach auf die technischen Aspekte fokussieren und zu guter Letzt wird ein Setup-Guide kommen.

## DAS ENDPRODUKT

In diesem Abschnitt möchte ich Ihnen die Funktionalitäten der Applikation aus der User-Perspektive näherbringen. Das heisst, es wird nur das Erklärt, was der Benutzer sieht.

### DER HEADER

Die Kopfzeile ist simpel gehalten. Sie zeigt einen Titel, drei Untertitel und einen Slider, welcher jeweils den Untertitel unterstreicht, der momentan aktiv ist. Sie sieht so aus:



Da es sich hier um eine Single Page Applikation handelt, findet kein Refresh statt, wenn man von der Blog-Seite auf z.B. die Admin-Seite wechselt. Dieses nicht-neu-Laden, ermöglicht zum einen, dass die Seiten Blitzschnell da sind, ab dem Punkt, ab dem sie das erste Mal geladen sind, zum andern, ermöglicht es, dass der Slider von einem Untertitel zum nächsten «gleitet», sobald man daraufklickt. Es findet also eine Animation statt und nicht einfach ein plumpes Unterstreichen des aktiven Kontextes.

## DIE BLOG SEITE

Die Blog Seite ist das Erste, das man sieht, wenn man die Website aufruft. Sie zeigt die einzelnen Posts. Ein Post kann so aussehen:

### About

2018-01-12

#### Why

My goal with this project was, to get better in writing front-end web applications. Although I've previously created projects, which involved creating a web page, the focus never solely was the frontend.

#### How

Of course, if you want to be a good web dev today, plain JS won't cut it anymore. So I created this page with the React framework. My experience with it has been mixed, but over all very positive. I say mixed because many things in the beginning were confusing and frustrating to me.

The backend was written in Node.js. This wasn't a challenge since I've created previous backends using Node.

The database is a MySQL database. The backend communicates with it over stored procedures.

#### Features

##### Highlighting code-snippets

```
const greet = () => console.log("Hello world")
let foo = (bar) => {
  bar()
}
```

##### Parsing markdown

The text you're reading right now, is stored as a [markdown](#) string in the database. The client parses this markdown to HTML.

##### Admin controls

If you have an admin account, you can log in with your credentials in the "Admin" tab. After you're successfully logged in, you can create and change posts in the frontend



Ein Post kann verschiedenste Elemente beinhalten. Sie sehen in dem oberen Ausschnitt einen Übertitel und mehrere verschachtelte Untertitel, einen Link, der blau gekennzeichnet ist, ein Bild und ein Codeauszug (Der Codeauszug ist nicht einfach ein Bild, sondern wirklicher Text, den man auch kopieren kann).

Diese weiteren Features sind möglich, welche im obigen Bild nicht gezeigt werden:

- Geordnete und ungeordnete Listen
  - o mit Unterelementen
- Tabellen
- Diverse Textstylings, Hervorhebungen

Zu Demonstration dieser Features, habe ich einen Eintrag erstellt:

# Various examples

2018-01-25

## List

### Unordered List

- This is an element
  - This is a sub-element
  - Another sub-element
- This is another element

### Ordered List

1. This is the first element
2. This is the second element
3. This is the first element of the second element

## Emphasis possibilities

This is bold text

*This is italic text*

~~Strikethrough~~

## Tables

Option	Description
data	Data
engineeeeesssse	More DATA
ext	SOME more Data

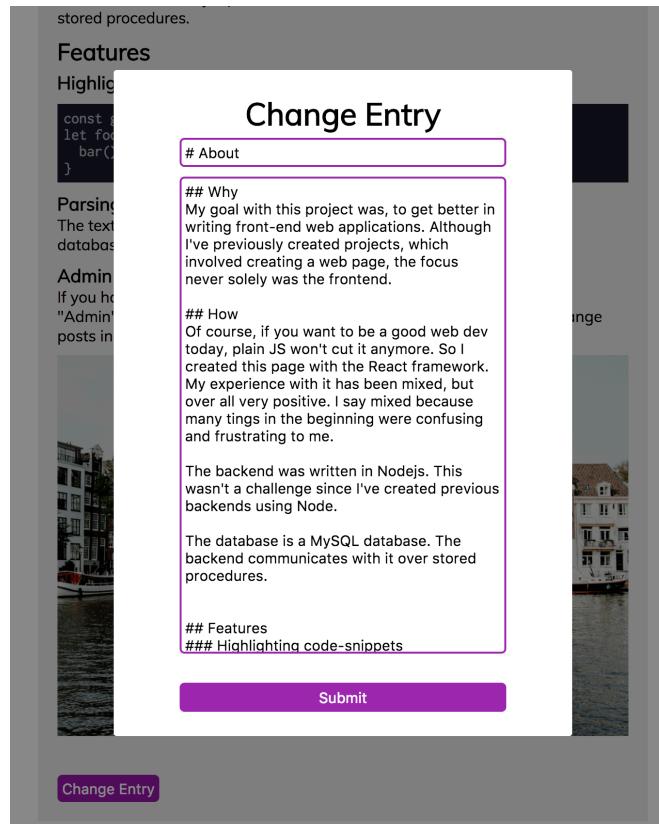
## DIE ADMIN SEITE

Die Admin Seite wird benutzt, um sich als Administrator einzuloggen. Sie sieht so aus:

The screenshot shows a light gray login interface. At the top, the text "not authenticated" is displayed. Below it are two input fields: one for the username containing "fabian" and another for the password containing "\*\*\*\*\*". At the bottom is a purple "Submit" button.

Wenn man Submit drückt, wird ein Diffie Hellman Key-Exchange gemacht, der einen Key generiert, mit dem der Benutzername und das Passwort verschlüsselt wird. Das heisst, dass die Verbindung sicher ist, obwohl kein SSH gebraucht wird. Zu dem Schlüssel-Austausch habe ich im späteren Abschnitt mehr geschrieben. Es muss hier angemerkt werden, dass es zwei bis drei Sekunden dauert, bis man angemeldet ist. Dies ist der Fall, wegen den gigantischen Key Sizes.

Nachdem man sich erfolgreich angemeldet hat, verändert sich die Blog-Seite (ohne Neu laden der Seite natürlich). Es erscheint in jedem Post ein «Change Entry» Button und am unteren Ende der Seite findet sich ein «Create Entry» Button. Wenn man einen drückt, erscheint ein Pop-up, das so aussieht:



Nachdem man entweder einen neuen Post schreibt oder einen bestehenden ändert, werden die Posts in der Veränderten Form angezeigt (ohne Neu laden der Seite natürlich)

# TECHNISCHES

Nachfolgend, werde ich über die technischen Aspekte schreiben. Wie einzelne Dinge funktionieren, was ich mir überlegt habe, was für technische Probleme ich während der Entwicklung hatte und wie ich diese in den meisten Fällen lösen konnte.

Viel davon Stammt aus dem File `learning_progress.md`, welches ich während der Implementierung geführt habe und welches deshalb auf englisch ist. Ich werde jedoch damit beginnen, die Aspekte, welche das Modul abverlangten zu beschreiben.

## FÜNF TIER ARCHITEKTUR

### DREI-TIER

Dies war kein Problem, da ich schon mehrere drei-tier Webapplikationen gemacht habe. Die Datenbank ist eine MySQL Datenbank, mit welcher ich per Sequel Pro interagiere. Die Businesslogik ist eine Node.js Server. Das Frontend ist eine Single-Page App, realisiert mit React. Bevor ich das Frontend entwickelte, erarbeitete ich einige Beispiele, um in React hereinzukommen. Diese sind im Branch «`react_prototype_and_testing`» abgelegt.

### VIER-TIER

Um diesen Stand zu erreichen, hatte ich dafür sehr viele Schwierigkeiten. Ich versuchte gut und gerne etwa 6 Stunden, ODBC zum Laufen zu bekommen. Das Problem ist, dass der Treiber, welcher ich für das vorgängige mini-Projekt gebraucht habe, abgelaufen ist und nun einen Lizenzschlüssel fordert. Wenn ich einen der Treiber nehme, welche auf der offiziellen MySQL Seite angegeben werden, für mein Betriebssystem und Bit Architektur, bekomme ich eine Fehlermeldung mit dem Inhalt «`h`». Das andere Problem ist, dass UnixODBC das hinterletzte Programm ist.

Zum Glück, wurde es uns ja erlaubt, die Datenbankverbindung einfach in eine andere Klasse auszulagern und somit auch eine Abstrahierung zu schaffen. Dies tat ich dann auch. Die Datenbankklasse befindet sich im File ‘`my_blog/implementation/database.js`’. Die Hauptklasse (‘`my_blog/implementation/index.js`’) kann auf die Funktionen der Datenbankklasse so zugreifen:

```
connection.authenticate(username, password, (rows) => {
```

### FÜNF-TIER

Dies war wiederum kein Problem, denn Stored Procedures sind mir recht gut bekannt. Die Create Statements für die Procedures finden sich im File:  
`my_blog/implementation/database/stored_procedures.sql`

## TRANSACTIONS

Eine andere Anforderung war ja, Transactions zu nutzen. Ich habe diese bei mir bei der `change_entry` Funktionalität eingebaut, weil diese Stored Procedure die einzige ist, die ein Update-Call ausführt. Dies kann man im oben erwähnten `stored_procedures` File auf der Zeile 54-57 sehen.

Es folgen nun drei persönliche Themen, mit welchen ich mich während des Projekts befasst habe.

## SUB-HEADER-SLIDER PROBLEMS

The sub-header-slider caused me a big headache. It is a [html hr](#) element. I wanted to set the width and margin from left based on the currently viewed page. This means, that it should underline the Admin sub-header, if you're on the admin-page. Also, it should 'slide' from one position to another and not just 'jump'. Sounds not too bad right? No, wrong! Sounds terrible!

### REFS

The first problem was, that when you pass the active header element as a prop, React ignores the css transition and just **jumps** the slider. THIS IS NOT SUPPOSED TO HAPPEN. I've nearly lost my mind over this, no joke. I sat at this exact problem for god knows how many hours. It would've also worked with using document.getElementById but this is an even bigger no-no in React than refs. So, I ended up using them instead.

### WRONG POSITIONING ON LOAD

I determined the position which the slider should have, with the getBoundingClientRect().left property of the currently active sub-header. I then pass this value into the render method of the element as an inline style. The problem was, that the .getBoundingClientRect().left property of the currently active sub-header returned the wrong value when the page was firstly loaded. This means, that the slider was off by a cm or so on the first render of the page. I was pretty lost and didn't know how to fix this bug. So, I let it slide and strangely I solved the problem unintentionally later on, when I learned about debounce.

### DEBOUNCE

I wanted to re-set the position of the slider, when the page was resized. So, I added an event-listener and when it was called, I re-set the position of the slider. This worked great but there was a problem: Because the resize listener gets called about a trillion times when you resize the browser by a hair, the performance of your website is **heavily** compromised. Luckily this is a common problem people face and there is a simple solution: Debounce. Basically, you say that the function, which does the repositioning only gets called every 50ms although the listener is fired. This results in a **slight** delay between your resizing and the repositioning but this difference is marginal and the performance gain is huge. This also resolved my previous wrong position on load problem because the function was called 50ms after the first render function call and the DOM has fully loaded at that point.

## DIFFE-HELLMAN KEY-EXCHANGE

Because of reasons, I've implemented a diffie hellman key exchange, to encrypt/decrypt the username and password posted to the "/authenticate" route. It would be fatal, if the username and the password were not encrypted because having the admin username and password enables you to alter and create entries and I don't have a SSH certificate at the moment. I quite like cryptography and to do this by hand and not by using a library was quite nice. Here's how it works:

### HOW IT WORKS

The goal of the exchange is, that party a and party b end up with the same key, which they can use to encrypt and decrypt messages. The problem is, that this key cannot be sent directly since the connection between the two is not secure; we have to assume, that every message sent to the other party can and will be read.

The numbers involved in this key exchange are gigantic and you have to consider, that you have to execute arithmetic operations on them. This leads to a number of problems. For one, JS cannot store these huge numbers natively. I used a library called [big-integer](#) for that purpose. This library allows you to store virtually infinite numbers and execute arithmetic operations on them. So that's the first problem you of the way.

The other problem is directly linked to the first problem in the sense that the operations which you apply on these numbers will take a tediously long time, if you don't optimize them. Luckily, the only kind of operation, which you have to do on these numbers is something called a modular exponentiation ( $A^B \bmod C$  where B is typically huge). I say luckily, because you can optimize the heck out of that operation. You'll find a ton of results if you search for "fast modular exponentiation".

### PUBLIC VARIABLES

There are two public variables, which are visible to anyone. They are g and n. They both are generated when the server is started and will not change for as long as the server is up.

- **public\_g:** The public variable g is a small prime number. We're talking something in the two to three digit region here.
- **public\_n:** The public variable n is a **huge** random number. The standard is 4000bit nowadays. I've used a 2000 digit decimal number. I haven't calculated how many bits the binary representation of such a number has but it's quite huge. The number is generated by reading out 250 bytes from the /dev/urandom file

### PRIVATE KEYS

These keys are **not** sent over the network. Both parties generate them on every key exchange

- **private\_key:** A random number between  $g/4$  and  $g$ . They could in theory range between 0 and the size of g. I didn't want that because small private keys could lead to security vulnerabilities. You also have to consider that you can't make the minimum random number too small because you end up with a too small random pool. I chose the number  $g/4$  as the smallest possible number.

## PUBLIC KEYS

These keys are sent over the network.

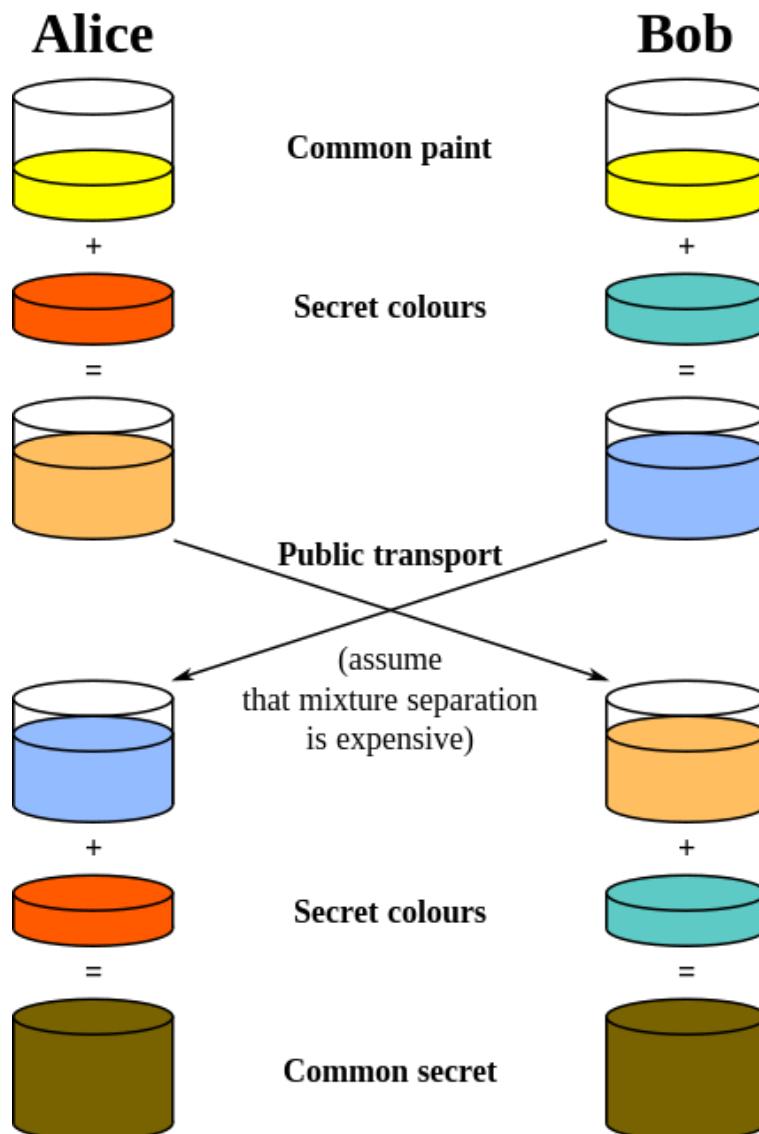
- **public\_key:** They are generated by both parties on every key exchange using their private keys like this:  $((\text{public\_g})^{\wedge}(\text{private\_key})) \bmod \text{public\_n}$

## SHARED KEY

This is the key that both parties end up with in the end to decrypt and encrypt their messages. Of course, they don't send them over the network.

- **shared\_key:** The private shared key is generated like this:  $((\text{public\_key\_of\_other\_party})^{\wedge}(\text{private\_key})) \bmod \text{public\_n}$

I find this a very good visualization of the process:



## DESIGNEN DER PAGE

Das Design der Page kommt nicht von ungefähr. Ich habe mich dabei auf den Material-Design Spezifikationen von Google abgestützt, aber zu einem grossen Teil habe ich Dinge auch auf meine Weise gemacht und mir selber überlegt, was zu dem Projekt und mir passt.

Ich habe keine Frontend Library verwendet. Das gesamte Design der Page ist hand-made. Es hätte mir viel Zeit erspart, eine Library wie Materialize (<http://materializecss.com/>) zu verwenden, jedoch wollte ich das nicht, weil der Blog ja etwas Persönliches sein sollte und wenn ich schon das Design nicht selber mache, kann ich ja gleich eine Template Seite von WordPress nehmen.

Dadurch, dass ich alles selber gemacht habe, haben sich meine Fähigkeiten im Frontend-Designen im Verlaufe dieses Projektes sichtlich verbessert. Ich wurde schneller und auch besser darin. CSS ist für mich jetzt übersichtlicher als zuvor.

# HOW TO RUN IT

In diesem Abschnitt halte ich fest, wie Sie die Applikation auf ihrem System zum Laufen bekommen. Wichtig dabei ist, dass sie weder Node oder NPM installiert haben müssen, denn ich habe einen Docker Container vorbereitet.

## SCHRITT 1 – DATENBANK AUFSETZEN

Da ich es nicht geschafft habe, die Web App und die Datenbank in den gleichen Container zu packen, muss sie der Container mit der Datenbank des Hosts' verbinden. Dies hat zur Folge, dass sie das Datenbankschema zuerst auf ihrem Hostsystem erstellen müssen. Ich habe dazu im File my\_blog/setup.md unter der Sektion «Database» ein Script erstellt, welches alles Nötige aufsetzt.

Damit sich der Docker Container später mit der Datenbank verbinden kann, braucht es einen Benutzer, welchem es erlaubt ist, sich von allen IPs her zu verbinden (Docker Container haben ja eine separate IP Adresse). Dieser User wird in diesem Script auch erstellt. Der Benutzername ist «fabian» und das Passwort: «abc123». Mit diesen Verbindungsangaben, wird sich der Docker Container später auf die Host Datenbank verbinden. Des Weiteren, wird natürlich die Datenbank und die Stored Procedures erstellt. Ausserdem wird ein Testeintrag eingefügt, welchen sie später sehen werden.

Nachdem Sie also den Codeblock unter dem Titel «Database» im File setup.md ausgeführt haben, sollte bei ihnen ein Datenbankschema namens my\_blog erstellt worden sein. Wenn dies der Fall ist, können wir mit dem nächsten Schritt weiterfahren.

## SCHRITT 2 – DOCKER CONTAINER ERSTELLEN

Das Erstellen des Docker Containers' ist relativ straight forward; Wenn sie sich im Directory my\_blog/implementation befinden, führen sie den zweiten Command in der Sektion «Docker Container» im File setup.md aus. Stellen Sie im Vorhinein sicher, dass der Docker Daemon läuft.

Nachdem das geschafft ist, machen Sie ihre IP Adresse im lokalen Netz ausfindig. Kopieren Sie den letzten Command aus dem File setup.md und ersetzen Sie die <> Klammern und ihren Inhalt mit ihrer lokalen IP Adresse. Der Command sieht dann z.B. so aus:

```
docker run --add-host=host_addr:192.168.1.102 -p 8080:8080 -d fabianbaechli/my_blog
```

Dieser Command fügt bei dem Docker Container einen DNS Eintrag hinzu, der sagt, dass der Hostname «host\_addr» die Adresse hat, die Sie eingefügt haben. Nachdem der Command ihnen ein Hash der Container Instanz zurückgegeben hat, sind Sie startklar. Verwenden Sie den Command «docker ps -a» um alle Ihre Container Instanzen zu sehen. Dort sollte dann etwas in dieser Form stehen (erster Eintrag):

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b657ac7ca033	fabianbaechli/my_blog	"node index.js"	2 minutes ago	Up 2 minutes	0.0.0.0:8080→8080/tcp	agitated_leakey
d289d05f5f49	148d666b6c85	"node index.js"	26 minutes ago	Exited (1) 21 minutes ago		gifted_noether
7b888c01ae7c	08839c0cfbce	"node index.js"	28 minutes ago	Exited (1) 28 minutes ago		zen_bell
86cadada35664f	08839c0cfbce	"node index.js"	30 minutes ago	Exited (1) 30 minutes ago		hardcore_colden
b36f5b17b9b7	4b03d720bab9	"node index.js"	5 days ago	Exited (137) 4 days ago		sad_goldwasser

Sie können auf die Lognachrichten der App per «docker logs <Container ID>» zugreifen. Wenn dort keine Fehlermeldungen sind, wird die App nun unter «localhost:8080» gehostet. In der App können Sie sich als Admin mit Username «admin» Passwort «abc123» anmelden.

## REFLEXION

Wie es sich vielleicht schon am Ton der Doku ablesen lässt, bin ich persönlich sehr zufrieden mit meinen Leistungen in diesem Modul und vor allem bezüglich dieses Projekts. Während ich im letzten Modul erst zu spät begann, mit meinen Teamkameraden ernsthaft zu arbeiten, beschäftigte ich mich hier von Projektbeginn bis Projektende (heute) intensiv mit meinem Ziel. Ich denke, dass ich einen sehr guten Zwischenweg gefunden habe, der sowohl meine Ambitionen und die Anforderungen des Moduls befriedigt.

Ich bin froh, dass ich mich darauf einliess, React zu lernen; es hat mein Horizont sehr erweitert und mich zu einem sichereren Programmierer gemacht.

Was ich während dem Entwickeln der nächsten App beachten möchte ist, dass ich weniger lange an einem Problem sitze. Es breitet sich in diesen Momenten eine Rastlosigkeit bei mir aus und ich bin nicht mehr produktiv. Die Probleme mit dem Subheader Slider haben mich ein ganzes Wochenende gekostet, an dem ich schlecht gelaunt war und im Nachhinein habe ich das Problem gelöst, ohne es in diesem Moment zu versuchen.