# my\_blog

Ein Projekt von Fabian Bächli

# 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.

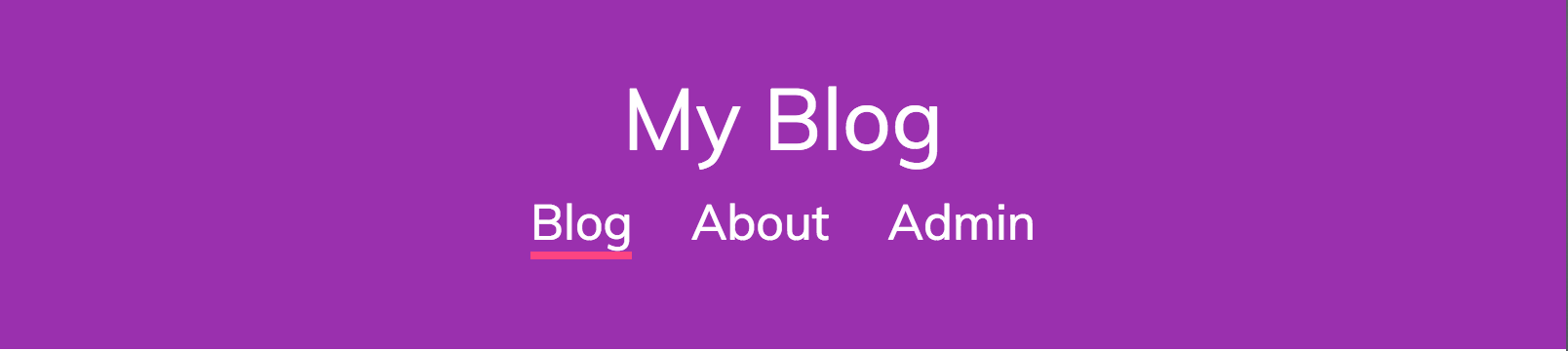
Von der Reihenfolge dieser Dokumentation her, werde ich Ihnen zuerst das Endprodukt näherbringen, mich danach auf die technischen Aspekte fokussieren und zu guter Letzt wird ein Setup-Guide kommen.

# 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

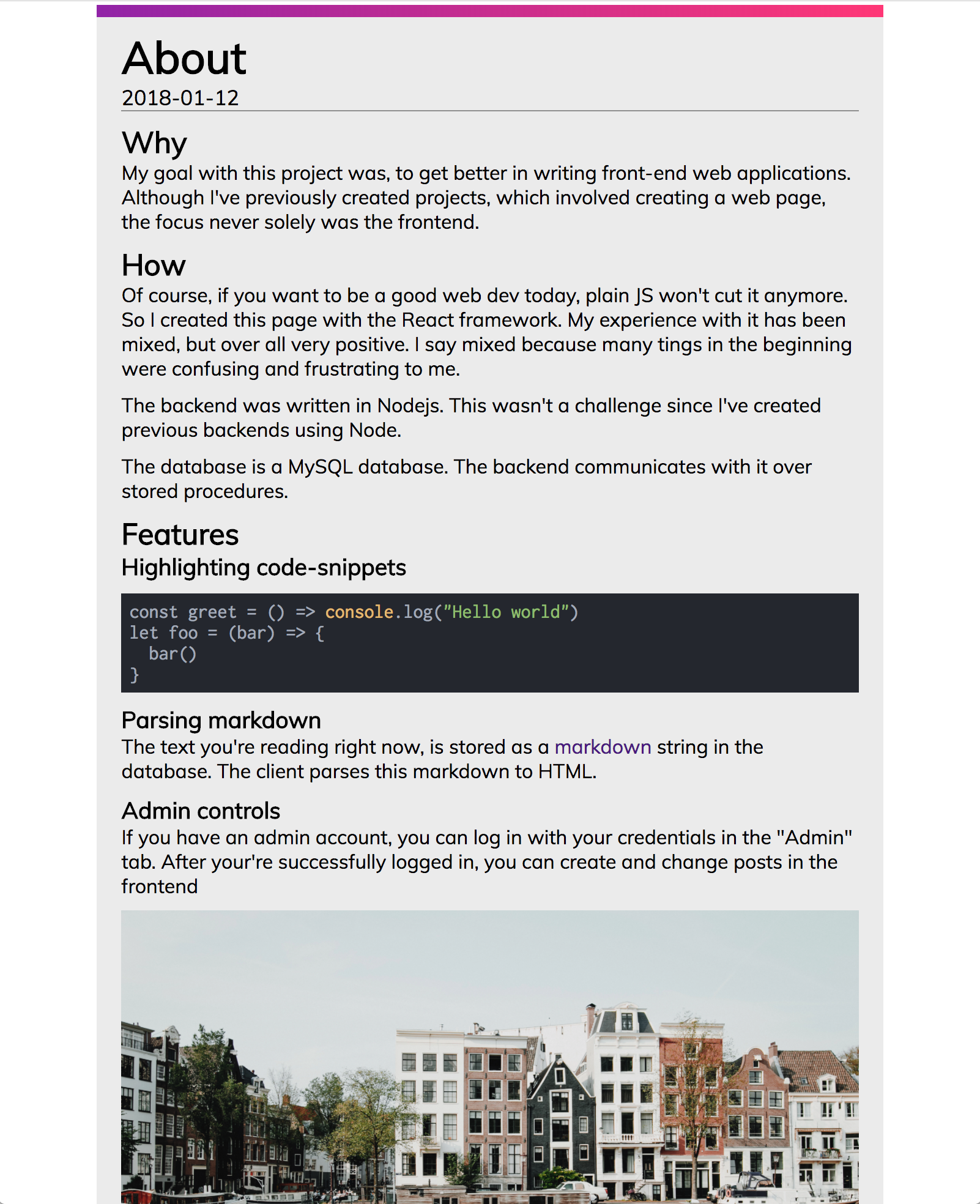
Die Kopfzeile ist simpel gehalten. Sie zeigt einen Titel, drei Untertitel und einen Slider, welcher jeweils den Untertitel unterstreicht, der momentan aktiv ist. Sie sieht so aus:



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:

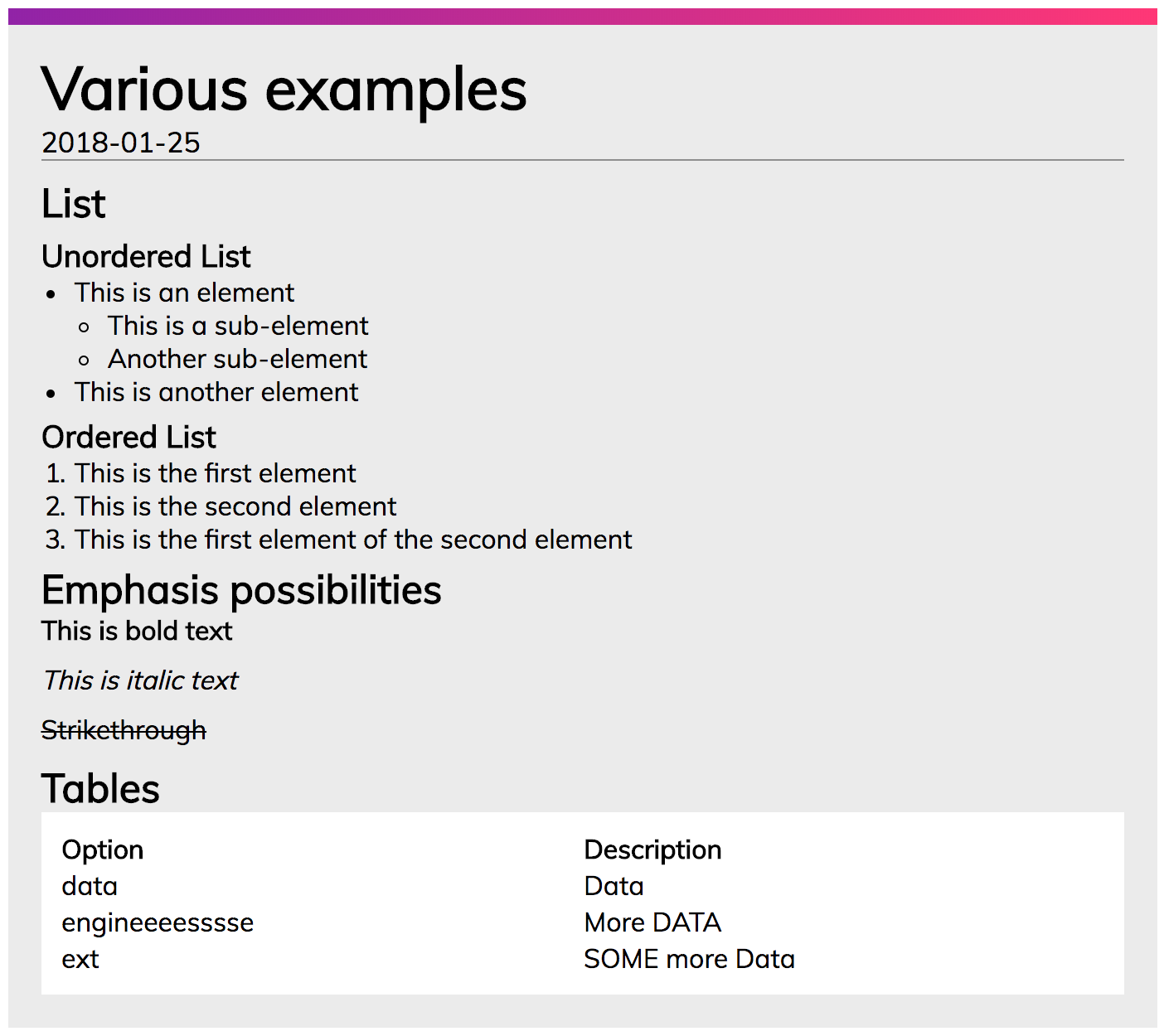


Ein Post kann verschiedenste Elemente beinhalten. Sie sehen in dem oberen Ausschnitt einen Übertitel und mehrere verschachtelte Untertitel, einen Link, der blau gekennzeichnet ist, ein Bild und ein Codeauszug (Der Codeauszug ist nicht einfach ein Bild, sondern wirklicher Text, den man auch kopieren kann).

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

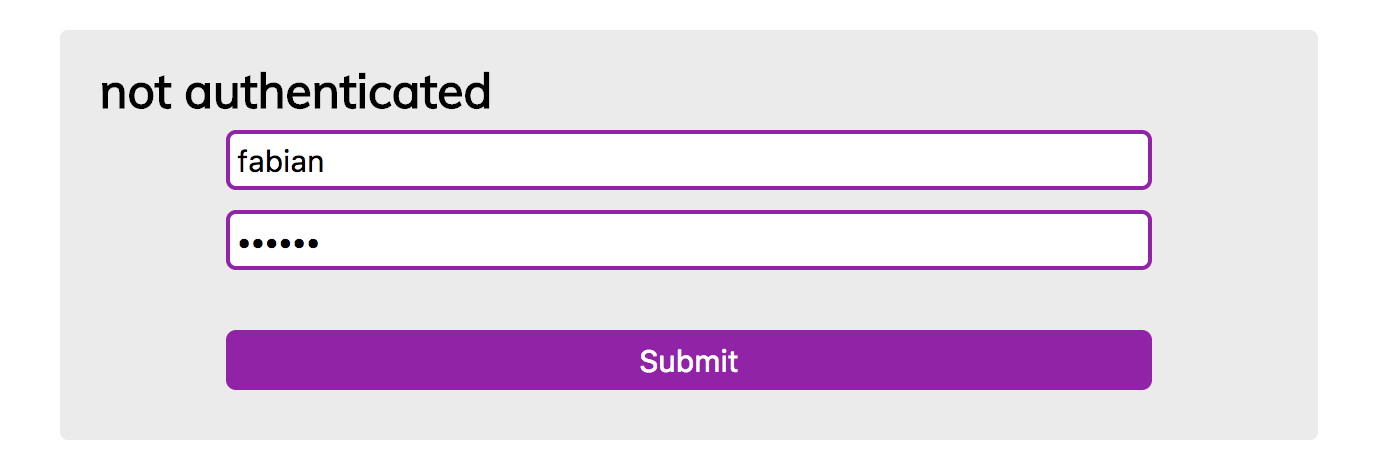
* Geordnete und ungeordnete Listen
  + mit Unterelementen
* Tabellen
* Diverse Textstylings, Hervorhebungen

Zu Demonstration dieser Features, habe ich einen Eintrag erstellt:



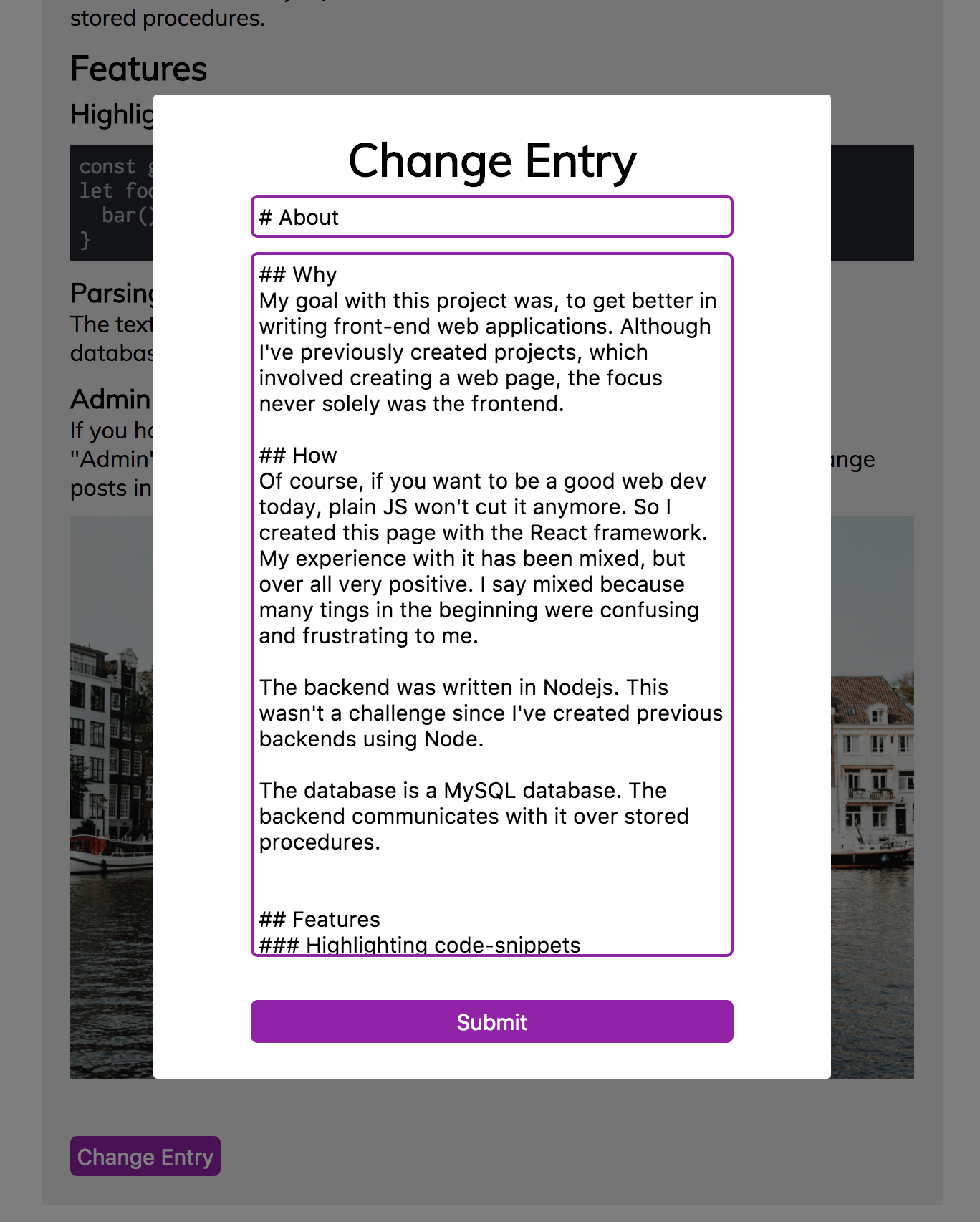
## DIE ADMIN SEITE

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



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.

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, erschein einem ein Popup, 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 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.

### 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. 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 findet sich im File ‘my\_blog/implementation/database.js’. Die Hauptklasse (‘my\_blog/implementation/index.js’) kann auf die Funktionen der Datenbankklasse so zugreifen:



### 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.

## SUB-HEADER-SLIDER PROBLEMS

The sub-header-slider caused me a big headache. It is a [html hr](https://developer.mozilla.org/de/docs/Web/HTML/Element/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 get's 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](https://www.npmjs.com/package/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 mod 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: ((public\_g)^(private\_key)) mod 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: ((public\_key\_of\_other\_party)^(private\_key)) mod public\_n

I find this a very good visualization of the process:

