

IPA

DOKUMENTATION

INNOVATION FLOW

IN REACT.JS UND STRAPI

Christoph Mayer
Februar 2022

Individuelle Projektarbeit
Mediamatiker Med18a

**«Even the smallest person can change the
course of the future»**

J.R.R. Tolkien, *The Fellowship of the Ring*

Inhaltsverzeichnis

1.	Anmerkungen	6
2.	Ausgangslage	6
2.1.	Vorgaben	6
2.2.	Ziel des Projekts	6
2.3.	Technische Anforderungen	7
3.	Rahmenbediengungen	7
3.1.	Arbeitsplatz	7
3.2.	Equipment	7
3.3.	Code Editor	7
3.4.	Github	8
4.	Projektplanung	8
4.1.	Eingabe	8
4.2.	Zeitplan	8
4.3.	Vorbereitung	9
4.4.	Zieldefinition	9
4.5.	Zielgruppendefinition	9
5.	Konzeption	10
5.1.	Strapi- und Userkonzept	10
5.2.	Dependencies	11
5.3.	Screendesign	11
6.	ECMAScript 6 (ES6)	13
6.1.	Var, Let, Const	13
6.2.	Objects	14
6.3.	Arrow Functions	14
6.4.	Array.map()	15
6.5.	Object Destructuring	16
6.6.	Spread Operator	16
7.	React Main Concepts	17
7.1.	JSX	17
7.2.	Virtual DOM	18
7.3.	Components and Props	18
7.4.	State and Lifecycle	19
7.5.	React Hooks	20
7.6.	Handling Events	20
7.7.	Conditional Rendering	21
8.	Strapi	22
8.1.	Content-Types	22
8.2.	Content Manager	23
8.3.	Roles	23
8.4.	GraphQL	25
8.5.	Apollo Provider	25

9.	Darstellung auf Endgeräten	26
10.	Probleme	28
10.1.	Hook Order	28
10.2.	CORS	28
10.3.	Security	29
11.	Ausblick	30
11.1.	Security	30
11.2.	Scaling	30
11.3.	Datenschutz, Anonymität und rechtliche Rahmenbedingungen	30
11.4.	weitere Funktionalitäten	31
12.	Projektmanagement	32
12.1.	Zielüberprüfung	32
12.2.	Kostenschätzung	32
13.	Reflexion	33
14.	Literaturverzeichnis	34
15.	Anhänge	35
16.	Selbstständigkeitserklärung	36

1. Anmerkungen

Um das Lesen der Dokumentation als möglichst angenehm zu gestalten, wird auf die Verdeutschung von technischen Begriffen verzichtet.

Des Weiteren wird in der Dokumentation nicht der ganze Code erklärt. Es macht keinen Sinn, jede Zeile durcharbeiten und zu erläutern. Dies würde einerseits den Zeitrahmen sprengen, andererseits wäre der Nutzen gering.

Wichtig in der Programmierung ist es, die Konzepte hinter einer Programmiersprache zu verstehen. Hat man die Konzepte einmal verinnerlicht, so stellt man zunehmend fest, dass alle Programmiersprachen dieselbe Grundlage besitzen. Das Lernen einer neuen Sprache wird somit immens erleichtert; es ist lediglich die Syntax, die es noch zu lernen gilt.

Aus diesem Grund wird in der Dokumentation anhand von praktischen Beispielen auf Grundkonzepte von React und Javascript (ES6) eingegangen, die in der App Anwendung fanden. Die Grundlagen wie Schleifen, Strings, Variablen, Arrays, Objekte etc. werden nicht erläutert.

2. Ausgangslage

Brüggli Medien resp. Advery rollt zurzeit ein Programm zur Steigerung der Resilienz auf: Innovation-Flow. Ein geeignetes Ideenmanagement und eine neue Form der Feedbackkultur stehen daher im Dialog. Ich war bereits im Vorfeld bei der Ideensuche beteiligt. Dabei fand eine Sitzung mit verschiedenen Bereichsleitern aus dem Brüggli statt.

Aus der Sitzung ging unter anderem hervor, dass der klassische, analoge Ideenbriefkasten oftmals schlichtweg ignoriert wird. Das Konzept einer digitalen Pinnwand fand daraufhin Anklang, wurde bis anhin aber nicht umgesetzt.

Eine Zusammenfassung der Sitzung wurde im Vorfeld bereits verfasst (siehe Anhang).

2.1. Vorgaben

Aus der Zusammenfassung geht hervor, dass die Förderung einer offenen Feedbackkultur wirtschaftlich zielorientiert wie auch zeitgemäss ist. Da es in der Sitzung lediglich um das Sammeln von ersten Ideen ging, gibt es keine konkreten Vorgaben für die Umsetzung.

2.2. Ziel des Projekts

Das Ziel des Projekts ist die Umsetzung einer Web-App mit einer Ideen-Pinnwand. Mitarbeiter können über diese Pinnwand Feedback zur Firma, Strategien, Prozessen, Kantine etc. abgeben.

2.3. Technische Anforderungen

Das Frontend der Web-App ist mit React, das Backend mit Strapi umgesetzt.

React ist eine JavaScript-Bibliothek (kein Framework) zum Erstellen von Benutzeroberflächen. Es wurde im Jahr 2011 bei Facebook entwickelt. Im Jahr 2017 veröffentlichte Facebook React Version 16.0.0 erstmalig unter der MIT-Lizenz (React Documentation, 2022).

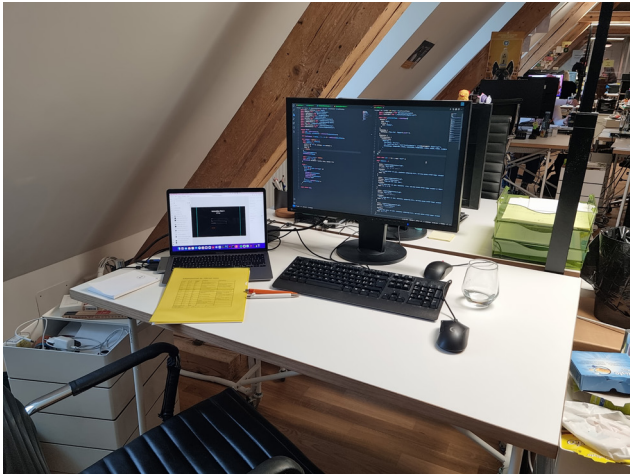
Strapi ist ein sogenanntes Headless-CMS. Es stellt somit kein eigenes Frontend sondern lediglich eine Programmierschnittstelle (API) bereit. Standardmässig verwendet Strapi für die Datenhaltung MongoDB (Strapi Documentation, 2022).

3. Rahmenbediengungen

Das Projekt wurde während 81 Stunden im Rahmen der IPA umgesetzt.

3.1. Arbeitsplatz

Der Arbeitsplatz befand sich in unserem Büro in Romanshorn. Die Atmosphäre in der Agentur ist ruhig, weshalb man sich ohne Probleme auf die Arbeit konzentrieren kann.



3.2. Equipment

Um eine Website umzusetzen braucht es lediglich einen Computer und eine Internetverbindung. Ich habe meinen persönlichen Laptop benutzt, um Restriktionen auf unseren Arbeitscomputern zu umgehen. Des Weiteren nutzte ich einen zweiten Bildschirm, um meine Effizienz zu steigern. Um akustische Störquellen während der IPA zu vermeiden, hatte ich Kopfhörer.

3.3. Code Editor

Ich nutzte eine Reihe von Tools, um mir die Arbeit zu erleichtern. Als Code Editor nutzte ich

Visual Studio Code, einer der führenden Codeeditoren, entwickelt von Microsoft (Visual Studio Code, 2022). Visual Studio Code beinhaltet einen integrierten Marktplatz, bei welchem man hunderte von Extensions herunterladen kann.

Ich nutzte Prettier als Code Formater, sodass mein Code beim speichern automatisch formatiert wird. ReactJs Code Snippets ist eine Erweiterung, die eine Reihe von Code-Snippets zur Verfügung stellt. Diese können dann einfach über einen Kürzel aufgerufen werden. Zuletzt nutzte ich noch Remote-SSH, um eine Verbindung zwischen Editor und Server herzustellen. Dies ermöglicht mir Files auf dem Server direkt mit Visual Studio Code zu bearbeiten.

3.4. Github

Die Webapplikation ist auf Github veröffentlicht. Ich nutzte Github unter anderem, um mir das Deployment auf dem Server zu erleichtern.

Mein Github Profil: <https://github.com/Stoeff25>

Link zum Projekt: <https://github.com/Stoeff25/innovation-flow-app>

4. Projektplanung

Die Projektplanung war ein langer Prozess und begann bereits nach dem ersten Infoabend in der SBW. Wir wurden ausführlich über die kommende IPA aufgeklärt. In den folgenden Wochen hatten wir dann Zeit, uns für ein Projekt zu entscheiden und dementsprechend auch die Eingabe zu erstellen.

4.1. Eingabe

Das Eingabeformular wurde mit den Experten bereits besprochen und befindet sich im Anhang. Die Eingabe verlief ganz gut, wobei ich einige Dinge als nicht ideal empfand:

Einerseits wäre eine Dauer von 90 Stunden geplant gewesen. Da ich zwei Schultage habe, hätte sich meine IPA über ein drittes Wochenende erstreckt. Da dies nicht möglich sei, musste ich mein Projekt um neun Stunden kürzen.

Zweiter Punkt sind weitere Muss-Ziele, die ich einfügen musste. Ich musste die Zeit bereits kürzen, brauchte dennoch aber weitere Ziele. Eine saubere Website in React mit einem Headless CMS ist aufwändiger als es scheint.

Nach einigen Anpassungen hatten wir die Eingabe aber geschafft, was eine grosse Erleichterung war.

4.2. Zeitplan

Bereits bei der Eingabe mussten wir eine grobe Zeitplanung mitgeben. Eine detaillierte Zeit-

planung wurde dann kurz vor der IPA gefertigt und mit den Experten besprochen. Der Zeitplan wurde während dem Projekt stetig um den IST-Zustand ergänzt. Der grobe wie auch der detailierte Zeitplan ist im Anhang und in der Abgabe enthalten.

4.3. Vorbereitung

Bereits seit Monaten arbeite ich mich in die Konzepte von Javascript und React ein, was diesem Projekte natürlich zugute kam. React ist eine Komponentenbasierte Bibliothek, was bedeutet, dass wenn man ein paar Grundkomponenten erstellt hat, stetig auf diesen aufbauen kann.

Ausserdem musste ich mich nicht mehr in die vielen, diversen Dokumentationen einarbeiten, da ich darin mittlerweile eingespielt bin. Im Literaturverzeichnis sind alle Dokumentation, die ich für die Arbeit gebraucht habe, gelistet.

Auch mit Strapi hatte ich bereits meine Erfahrungen gemacht. Ich wusste daher bereits, wie man mit der API von Strapi umgehen muss, wie man es einrichtet und auch wie man GraphQL einbindet.

4.4. Zieldefinition

Ziel des Projekts ist die Umsetzung einer Website mit einer Ideen-Pinnwand. Mitarbeiter können über diese Pinnwand Feedback zur Firma, Strategien, Prozessen, Kantine etc. abgeben.

Das Screendesign wird mithilfe von Adobe xD erstellt und die Umsetzung erfolgt lokal. Nach Abschluss wird die App auf einem Server gehostet und über eine Domain erreichbar sein.

Aufgrund von internen Sicherheitsbedenken und unnötiger Komplexität wird das Login lokal verwaltet. Es werden im Strapi (Headless CMS) interne User mit Passwörtern angelegt, über die das Login geregelt wird.

MUSS-Kriterien:

- Lokale Umsetzung in Strapi + React.js (inkl. HTML/CSS)
- Kurzes Konzept der Webseite
- Screendesign in Adobe xD
- Übersicht der Feedbacks
- Hinzufügen, Bearbeiten und Löschen von Feedbacks
- Hosting auf einem Server (ohne Login im internen Netzwerk)

KANN-Kriterien:

- Login-Interface
- Liken und Kommentieren von Feedbacks

4.5. Zielgruppendefinition

Ein massgebender Punkt für den Erfolg des Produktes ist eine Übereinstimmung zwischen dem Angebot und der zugehörigen Zielgruppe zu finden. Im ersten Schritt muss also die potentielle Zielgruppe definiert werden.

In diesem Fall ist die Zielgruppe alle Mitarbeiter des Brüggli resp. Brüggli Medien. Das Brüggli beschäftigt Menschen in jedem Alter, weshalb das Alter der Zielgruppe sehr weit gestreut ist. Nichtsdestotrotz fällt auf, dass das Brüggli gerade sehr viele junge Menschen beschäftigt, da es ein sehr grosser Ausbildungsbetrieb ist.

Im nächsten Schritt findet die Kundensegmentierung statt, bei der man die Zielgruppen nach verschiedenen Merkmalen unterscheidet.

Soziodemografie:

Hierunter fallen beispielsweise Geschlecht, Altersstruktur, Einkommen aber auch die Bildung der Zielgruppe. Das Brüggli beschäftigt sehr viele junge Menschen, was beim Produkt sicherlich beachtet werden soll.

Geografie:

Die Geografie bestimmt die räumliche Verteilung der Zielgruppe. In diesem Fall ist es klar, dass die Zielgruppe primär Schweizer sind. Natürlich gibt es aber auch einige Mitarbeiter aus angrenzenden Gebieten wie Österreich oder Deutschland. Im Grunde kann man aber sagen, dass es vor allem Deutschschweizer sind.

Psychografie:

Die psychografischen Merkmale widmen sich der Lebensstile und den Werten der Zielgruppe. Das Brüggli beschäftigt eine Vielzahl an Individuen, weshalb eine klare Unterteilung schwer fällt.

Verhalten:

Das Verhalten widmet sich den unterschiedlichen Einkaufsstrategien der Zielgruppe. Beim Innovation Flow geht es aber nicht darum, Einnahmen zu generieren, weshalb dieser Punkt ausser acht gelassen werden kann.

5. Konzeption

In der Konzeption ging es darum, sich erste Gedanken zur Umsetzung zu machen. Einerseits musste ein Konzept für das Backend her, andererseits brauchte es aber auch ein Screendesign. Ausserdem musste ich bereits zu Beginn definieren, welche Dependencies ich nutzen werde.

5.1. Strapi- und Userkonzept

Bei der Installation von Strapi ist eine erste Kollektion für Users bereits definiert. Darin gibt es Felder wie z. B. Username, Email, Password und weitere. Die Datenarchitektur dieser Kollektion wird nicht verändert, wobei sie aber um ein paar Punkte ergänzt wurde.

Des Weiteren braucht es noch eine Kollektion für Feedbacks sowie für die Kommentare. Folgende Felder sind für die Umsetzung notwendig (Standardarchitektur von User nicht abgebildet).

User	Feedback	Comment
feedbacks	title	content
comments	content	feedback*
commentLike*	comments*	user*
feedbackLike*	users*	likes*
	likes*	

Die Felder ohne Stern sind allesamt Textfelder; entweder für einen kurzen Titel oder für längeren Inhalt. Die Felder mit Stern sind Relationen, die in Strapi erstellt werden können. Weiteres zu Relationen im Kapitel 8.1 Content-Types.

5.2. Dependencies

Es mussten einige Dependencies definiert werden, um die Seite umsetzen zu können. Es ist daher nicht schlecht, wenn diese schon zu Beginn definiert werden. Im folgenden ist eine Liste mit den installierten Dependencies zu sehen.

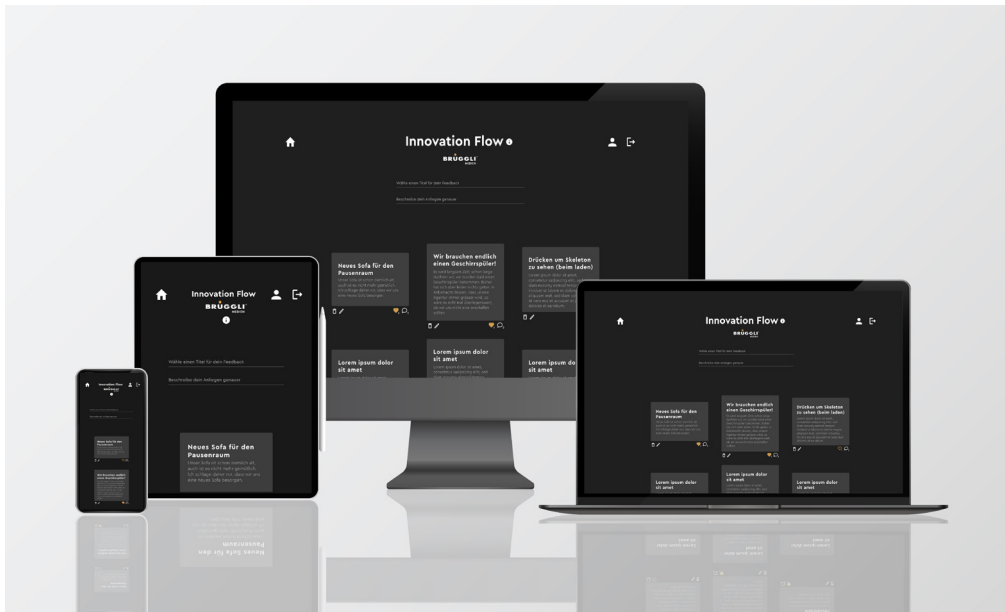
- MUI; React UI Bibliothek
- Font-Awesome; Icon Bibliothek
- Joi; Schema und Validation von Inputs
- React Router; Routing für React Apps
- React Toastify; React Notifications (Toast / Snackbar)
- Apollo Client; Client für HTTP-Requests
- GraphQL Plugin für Strapi; query language für APIs

5.3. Screendesign

Das Screendesign wurde in Adobe xD umgesetzt. Alle benutzten Komponenten sind von MUI bereits vorgefertigt, wie üblich bei einem CSS-Framework. Um diese im Adobe xD nicht alle nachbauen zu müssen, habe ich das Adobe xD Template von MUI gekauft. Darin sind alle Komponenten wie Buttons, Inputs, Cards etc. enthalten. Ausserdem sind die zugehörigen States auch bereits definiert, wie beispielweise der Button bei einem Hover.

Mit dem Template konnte sehr viel Zeit gespart werden, da ich mir ganz einfach meine gewünschten Komponenten aussuchen und schliesslich zusammensetzen konnte. Danach mussten nur noch Farben und Schriften angepasst werden und schon stand das Screendesign im Grunde.

Im letzten Schritt fehlte dann nur noch, dass ganze interaktiv zu machen, indem ich verschiedene Komponenten respektive Seiten miteinander verknüpfte. Das Screendesign ist in der Abgabe enthalten (als PDF und xD-File) und kann betrachtet werden. Ausserdem befindet sich noch eine Version auf Papier im Anhang.



Ich habe das Screendesign im Dark-Mode umgesetzt. Das Template für MUI hätte ich auch in der Light-Version, ich arbeite aber grundsätzlich lieber im Dark-Mode, da mir die Augen nach einiger Zeit sonst schmerzen, wenn ich alles im Light-Mode mache. Mein Editor wie auch meine Laptop-Präferenz sind ebenfalls im Dark-Mode.

Das schöne an MUI ist, dass die Implementierung von verschiedenen Themes sehr einfach ist. Um das Theme von Dark auf Light zu ändern, muss man lediglich in einer createTheme Komponente Light statt Dark mitgeben (als String) und schon passt sich die ganze Seite an.

```
createTheme({
  palette: {
    mode: 'dark',
    primary: {
      main: '#F59F2F',
    },
  },
})
```

Im nächsten Schritt könnte man dann sogar verschiedene Themes implementieren, welche alle eine eigene Palette haben. Statt also nur Dark / Light, könnte man noch ein Advery Theme oder ein Brüggl Theme einbinden und die mit einem Switch-Case Statement abfragen. Da Themes aber den Zeitrahmen sprengen würden, habe ich lediglich den Dark-Mode sowie eine eigene Farbe für Auszeichnungen implementiert (Orange von Brüggl Medien CI/CD). Das CI/CD von Advery und Brüggl Medien ist in der Abgabe enthalten.

6. ECMAScript 6 (ES6)

ECMAScript 6 (auch ECMAScript 2015) ist die sechste Version von JavaScript und die erste Grundlegende Überarbeitung seit ES5 im Jahre 2009 (ECMAScript6 Einführung, 2021). Die neu eingeführten Konzepte sind gerade beim Arbeiten mit Komponentenbasierten Frameworks / Libraries von grosser Bedeutung. Seit ES6 gab es bereits sechs weitere Überarbeitungen, keine von denen aber hatte eine solche Signifikanz für die Syntax, wie es ES6 hatte.

Im folgenden werden wir auf ein paar Konzepte eingehen, die auch für meine Arbeit an Bedeutung fanden. Ausserdem möchte ich anfügen, dass ich mich an der Struktur des Kurses «Mastering React» von Mosh Hamedani orientiere. Ich hatte diesen Kurs über Monate intensiv bearbeitet, die folgenden ES6 Konzepte wurden darin alle im Kapitel «ES6 Refresher» behandelt und sind fundamental für React (Mastering React, Mosh Hamedani, 2022).

6.1. Var, Let, Const

Vor ES6 wurden alle Variablen mit dem Keyword «var» initialisiert. Im Gegensatz zu Objektorientierten-Programmiersprachen wie Java oder C, muss man in JavaScript seine Variablen keinen bestimmten Datentypen zuweisen. Die Zuweisung im Speicher geschieht im Hintergrund und wir als Entwickler müssen uns darüber keine Gedanken machen.

Das «var» Keyword bringt aber ein Problem mit sich, welches bei ES6 angegangen wurde, indem neue Keywords namens «let» und «const» eingeführt wurden.

Das Problem mit «var» möchte ich in einem Beispiel aufzeigen. Der Code im linken Bild führt zur Ausgabe im rechten Bild, sofort ist auch nun zu sehen, was der Unterschied zwischen «let» und «var» ist.

```
for (let i = 0; i < 5; i++) { }  
console.log(i)  
  
for (var i = 0; i < 5; i++) { }  
console.log(i)
```

```
undefined  
5
```

Obwohl «i» innerhalb der Schleife definiert wurde, ist «i» als var ausserhalb davon immer noch aufrufbar. Das ist ein Problem, dass gerade bei komplexeren Anwendungen zu Fehlern führen kann, da man gerade für Zählervariablen beispielsweise oftmals die gleichen Namen benutzt. Ein Datentyp sollte sich so nicht verhalten, weshalb «let» eingeführt wurde. Korrekterweise ist «i» nun auch undefined, wird es ausserhalb der Schleife aufgerufen.

Das Keyword «const» verhältet sich im Grunde gleich wie «let», nur dass es eine Konstante ist, was bedeutet, dass der Wert nach Initialisierung nicht mehr verändert werden kann. Versucht man es, so gibt es eine Fehlermeldung.

Grundsätzlich kann man also sagen: nutze «const» immer wenn es geht, setze «let» ein, wenn die Variable verändert werden muss. Hat man nicht einen zwingenden Grund dafür, «var» zu verwenden, dann sollte man immer davon ablassen.

6.2. Objects

Objekte wurden durch ES6 um einiges handlicher, auf die genauen Änderungen werden wir aber nicht eingehen. Im folgenden gehen wir auf ein Objekt unter Services/authService.js ein.

Im Bild ist eine Funktion saveUser() zu sehen, mithilfe derer ein paar Userdaten im Local Storage gespeichert werden. Im Local Storage kann man Daten im Browser des Benutzers speichern, sodass nicht immerwieder der Server angefragt werden muss.

Die Konstante «user» ist ein Objekt, welches im Local Storage gespeichert wird. Diese Objekt hat verschiedene Eigenschaften wie «id», «email» oder «username». Ausserdem könnten auch Funktionen in Objekten definiert werden, welche man dann als Methoden bezeichnet.

```
export function saveUser(data) {  
  const user = {  
    id: data.user.id,  
    email: data.user.email,  
    username: data.user.username,  
  };  
  localStorage.setItem("user", JSON.stringify(user));  
  localStorage.setItem("jwt", data.jwt);  
}
```

```
user.email  
user["email"]
```

Möchte man also die Email des Users aus dem Objekt lesen, so gibt es zwei Möglichkeiten, welche man als «Dot-Notation» resp. «Bracket-Notation» bezeichnet.

Die «Dot-Notation» ist die herkömmliche, so habe ich auch die meisten Eigenschaften aus Objekten gelesen. Diese ist im rechten Bild auf der oberen Zeile zu sehen. Man setzt einfach einen Punkt und kann dann alle Eigenschaften und Methoden des Objekts aufrufen.

Die «Bracket-Notation» ist vorallem dann nützlich, wenn wir zu Laufzeit nicht wissen, welche Eigenschaft wir aufrufen wollen. Im unteren Beispiel sieht man, dass «email» als String weitergegeben wird. Dies kann man z. B. für Search-Bars oder Filterfunktionen nutzen, wo die Daten meist als Strings vorliegen.

6.3. Arrow Functions

«Arrow Functions» wurden in ES6 eingeführt und haben die Syntax von Funktionen massgebend verändert. Im letzten Kapitel wurde eine Funktion saveUser() angeschaut, die nach altem Standard definiert wurde. Ich mache das hier, weil es sich um eine Hilfsfunktion in einem separaten Modul handelt. Innerhalb der React-Anwendung wurden nahezu überall «Arrow Functions» verwendet.

Im folgenden Ausschnitt aus Routes/Login.jsx ist eine doSubmit() Funktion zu sehen, die als «Arrow Function» definiert wurde. Im rechten Bild habe ich die Funktion zum Vergleich noch zu einer herkömmlichen Funktion umgeschrieben.

Im Grunde fällt einfach das «function» Keyword weg, stattdessen definiert man die Funktion als eine Konstante mit «const» und übergibt die Parameter nach dem Gleichheitszeichen. Dann folgt der Pfeil und anschliessend der Inhalt der Funktion in geschwungenen Klammern.

```
const doSubmit = ({ email, password }) => {
  login({
    variables: {
      email,
      password,
    },
  });
};
```

```
function doSubmit ({ email, password }) {
  login({
    variables: {
      email,
      password,
    },
  });
};
```

«Arrow Functions» sind in React-Anwendungen viel schöner zu lesen und handzuhaben, gerade wenn sie in JavaScript-Funktionen wie z. B. filter() benutzt werden. Folgend eine Zeile aus der Komponente form.jsx:

```
newData.filter((i, z) => (names[z] = i.name));
```

Man kann die Funktion viel intuitiver lesen, den Pfeil könnte man als «geht zu» übersetzen. In diesem Sinne heisst es, Parameter (i und z) gehen zu Return-Statement (names[z] = i.name).

Im unteren Bild sieht man die gleiche Zeile übersetzt in eine herkömmliche Funktion. Hier wird klar, wie unschön solcher Code werden kann, gerade wenn es komplexer wird. Wenn es Verschachtelungen gibt, führt dies zu wiederholten «function» und «return» Keywords, was es schwieriger macht, sich auf den Inhalt der Funktion selbst zu fokussieren.

```
newData.filter(function(i,z){return names[z] = i.name})
```

6.4. Array.map()

ES6 hat eine neue Methode für Arrays namens map() eingeführt, welche man in React für das Rendern von Listen benötigt. Immer wenn man also eine Liste rendern möchte, kommt die map-Methode zum Einsatz. Ich möchte dies anhand einem Code-Ausschnitt aus der Komponente components/feedbacks.jsx aufzeigen.

```
{feedbacks.map((feedback) => (
  <React.Fragment key={feedback.id || feedback.title}>
    <Grid item xs={12} sm={6} md={3}>
      <FeedBackCard feedback={feedback} />
    </Grid>
  </React.Fragment>
))}
```

Dieser Code-Ausschnitt ist für das Rendern der Feedback-Karten im Frontend verantwortlich. Alle Feedbacks und ihre Daten sind in einem Array «feedbacks» gespeichert. Mit der map-Methode kann man durch alle Feedbacks «loopen» und diese entsprechend als Karten anzeigen.

Innerhalb des `React.Fragment`-Tags wird dann für jeden Platz im Array, also für jedes Feedback, eine eigene Karte zurückgegeben. Ausserdem befindet sich das Ganze noch in einem `Grid`-Tag, welches eine Komponente der `React Library MUI` ist.

Die `map`-Methode ist im Grunde wie eine `For`-Schleife, aber eben nutzbar als Methode von Arrays. Wie man sieht, nutze ich hier auch die vorher behandelte «Arrow Function». Die Konzepte greifen in `React` also wirklich ineinander.

6.5. Object Destructuring

Ein weiteres Konzept, was ebenfalls oft in `React`-Anwendungen verwendet wird, nennt sich «Object Destructuring». Diese Konzept findet beispielweise in der Komponente `components/common/input.jsx` Anwendung.

Wir haben ein Objekt namens «`states`», welches verschiedene Eigenschaften besitzt. Mit `Object-Destructuring` können wir diese Eigenschaften in einer Zeile einer neuen Variable zuweisen. Wir haben nun neue Variablen resp. Konstanten mit den Namen «`errors`», «`setErrors`» etc. definiert.

```
const { errors, setErrors, showAddIcon, setShowAddIcon } = states;
```

Unser Code wird durch «Object Destructuring» schöner und besser lesbar, ausserdem kann man einiges an Repetition sparen. Zum Vergleich sieht man im unteren Bild noch das Äquivalent ohne `Object-Destructuring`. Auch hier sieht man wieder sofort, wie unschön solcher Code werden kann.

```
const errors = states.errors;  
const setErrors = states.setErrors;  
const showAddIcon = states.showAddIcon  
const setShowAddIcon = states.setShowAddIcon
```

6.6. Spread Operator

Ein weiteres, modernes `JavaScript`-Feature ist der «Spread Operator». Dazu betrachten wir folgende Zeilen im unteren Bild, die im Frontend unter `components/common/icons/modify.jsx` benutzt wurde.

Wie bereits erwähnt, ist «`feedbacks`» ein Array, welches alle Feedback-Daten beinhaltet. Teilweise ist es notwendig, diesen Array zu klonen, um damit weiterarbeiten zu können. Hierfür nutzte ich den «Spread Operator» (die drei Punkte). Alle Daten die im Array «`feedbacks`» gespeichert sind, werden nun in ein neues Array namens «`allFeedbacks`» geklont.

```
const allFeedbacks = [...feedbacks];  
const updatedFeedback = { ...feedback };
```

Das gleiche ist auch mit Objekten möglich, wie in der unteren Zeile zu sehen ist.

7. React Main Concepts

In diesem Kapitel beschäftigen wir uns mit den Grundkonzepten von React, welche allesamt auch in der App zur Anwendung kamen. Ich habe mich für dieses Kapitel an der offiziellen React Dokumentation orientiert (React Documentation, 2022), wobei ich aber nur ausgewählte Themen behandle.

7.1. JSX

React hat eine spezielle Syntax namens JSX eingeführt.

```
const element = <h1>Hello, world!</h1>;
```

Diese Syntax repräsentiert weder einen String noch HTML, es ist JSX, eine JavaScript-Erweiterung für React. Mit JSX kann man Daten strukturieren und sinnvoll anordnen; für deren Darstellung ist aber immer noch herkömmliches HTML zuständig.

Obwohl man React-Applikationen nicht mit JSX aufbauen muss, so wird es doch empfohlen, da es die meisten Entwicklern als hilfreich empfinden. JSX bietet visuelle Hilfestellung innerhalb des JavaScript-Codes, erweitert aber auch blosses HTML mit den vorzügen von JavaScript.

Da ich MUI als CSS-Framework verwendet habe, sind in meinem Code selten HTML-Tags zu finden. Stattdessen benutze ich Komponenten von MUI, wie bspw. die «Typography» Komponente im Bild unten. Hier verwende ich einen div-Tag, welche ich mit der Eigenschaft «component» definieren kann (Mui Documentation, 2022). Typography repräsentiert hier also einen div, wobei mit «variant» aber das aussehen bestimmt wird (hier h5).

Die Variable «title» wurde in meinem Code definiert und hier wird die Macht von JSX auch deutlich. Man muss seine Variablen lediglich in geschwungene Klammern schreiben und schon werden sie innerhalb eines HTML-Tags gerendert. Was immer man also in der Variable Title definiert, wird hier angezeigt.

```
<Typography gutterBottom variant="h5" component="div">  
  {title}  
</Typography>
```

JSX verschmilzt quasi JavaScript mit HTML. Das ist auch ein Grund, weshalb ich React so mag, denn mit HTML und vanilla / jQuery stoss ich oft an meine Grenzen. Die Manipulation von DOM-Elementen ist umständlich und führt zu Fehlern.

7.2. Virtual DOM

Im Gegensatz zu vanilla oder jQuery wird in React das DOM-Element nicht direkt manipuliert, stattdessen gibt es einen virtual DOM (kurz VDOM), den man manipuliert.

Der virtual DOM ist ein programmatisches Konzept. Die Idee dahinter ist, eine «virtuelle» Repräsentation der UI im Speicher zu halten (memory). React synchronisiert im Hintergrund die VDOM dann mit der «echten» DOM.

Dieses Konzept ermöglicht schnelle Applikationen, auch wenn sie gross werden, wie z. B. bei Facebook, Netflix oder Spotify.

Statt den ganzen DOM zu synchronisieren, wie bei alten HTML und PHP Seiten, wird nur der Inhalt synchronisiert, der sich auch verändert hat. Die Website fühlt sich dann an, als müsste sie nicht laden. Das Surferlebnis wird nahtlos.

7.3. Components and Props

Mit Komponenten können UIs in unabhängige, wiederverwendbare Teile gestückelt werden. Jede Komponente wird dann in Isolation betrachtet.

Konzeptionell gesehen sind Komponenten wie JavaScript-Funktionen. Sie akzeptieren eine beliebige Eingabe (gennant «props» für properties) und gibt React-Elemente (in JSX) zurück.

Alle Files mit der .jsx Erweiterung sind Komponenten in React. Folgend sieht man die Komponente components/common/renderComments.jsx.

```
import React from "react";
import Like from "../icons/like";
import { SingleComment } from "../singleComment";

const RenderComments = (comment, comments, doLike) => {
  const { avatar, username, secondary, likes } = comment;

  const handleLike = () => {
    doLike(comment, comments);
  };

  return (
    <React.Fragment key={comment.id}>
      <SingleComment
        avatar={avatar}
        username={username}
        secondary={secondary}
      />
      <Like likes={likes} onClick={handleLike} />
    </React.Fragment>
  );
};

export default RenderComments;
```

Eine Komponente hat immer die gleiche Struktur. Ganz oben finden sich alle import-Statements wie Dependencies oder andere Komponenten. Dann wird die Komponente in einer «Arrow-Function» definiert (hier RenderComments). Zu beachten ist, dass Komponente grossgeschrieben werden müssen. Als Parameter der Funktion übergibt man seine Props (Daten, welche aus höhere Instanz übergeben werden). Innerhalb der Komponente kann man noch beliebig viele Funktionen oder Variablen definieren (z. B. handleLike). Am Ende der Komponente folgt das Return-Statement, welches ein React-Element zurückgibt.

7.4. State and Lifecycle

Wie bereits in Kapitel 7.2 erwähnt, erstellt React im Hintergrund einen virtual DOM der mit dem realen DOM synchronisiert wird. Immer wenn wir den virtuellen DOM verändern, wird er mit dem realen DOM abgeglichen und synchronisiert. Die Änderungen sind im realen DOM dann sichtbar.

Das Konzept der «States and Lifecycles» dreht sich unter anderem darum, wie man den virtual DOM während seinen verschiedenen Lebenszyklen (Lifecycles) manipuliert. Die drei Zyklen, die eine Komponente durchläuft, sind «Mounting», «Updating» und «Unmounting».

Es ist ein No-Go, ein DOM Element direkt zu manipulieren, wie man es z. B. in vanillaJS mit `document.getElementById()` machen könnte. Stattdessen manipuliert man den virtuellen DOM, indem man seine «States» setzt. Die Synchronisation und Anzeige im realen DOM ist die Aufgabe von React.

«States and Lifecycles» ist ein Konzept in React, welches gerade am Anfang schwierig zu verstehen ist. Es wird schnell komplex, weshalb wir das Thema nur anschnitten.

In älteren React-Versionen sind Komponenten immer mit Klassen definiert worden, da nur Klassen einen «State» haben können, nicht aber Funktionen. Innerhalb der Klasse konnte man dann seine Methoden definieren, das Return-Statement und auch die verschiedenen Lebenszyklen ansprechen (on Mount, on Update, on Unmount). Im Constructor hat man dann seine «States» definiert.

Seit React 16.8 gibt es ein neues Feature in React genannt «React Hooks». Mit «Hooks» lassen sich «States» und auch weitere React Funktionen nutzen, ohne dabei eine Klasse schreiben zu müssen. Ich habe in meinem Projekt keine einzige Klasse (Class Components), es wurde ausschliesslich mit Funktionen (Functional Components) gearbeitet. Um den virtual DOM zu manipulieren habe ich «Hooks» verwendet. Mit «Hooks» kann man sich in Funktionen «einhaken», woher auch der Name kommt.

7.5. React Hooks

Das Vermeiden von Klassen macht den Code viel schöner, da einiges wegfällt. Beispielsweise gibt es keinen Constructor mehr, das `super(props)` Keyword um «this» nutzen zu können fällt weg, dauerndes Wiederholen von «this» im Code wird hinfällig, mit Context-Hooks kann man Prop-Drilling vermeiden uvm. Hooks sind eine sehr nützliche Erweiterung von React, man muss aber dann auch etwas umdenken, wenn man sich das Arbeiten mit Klassen gewöhnt ist.

In folgendem Ausschnitt aus der Komponente `routes/layout.jsx` sind einige Hooks zu sehen.

```
const [feedbacks, setFeedbacks] = useState([]);
const feedbackStates = [feedbacks, setFeedbacks];
const { loading, error, data, refetch } = useQuery(APOLLO_GET_FEEDBACKS);

const user = getCurrentUser();

useEffect(() => {
  if (!loading && !error && data) {
    const feedbacks = changeObjectStructure(data);
    setFeedbacks(feedbacks);
  }
}, [loading, error, data]);
```

Die erste Zeile ist die Definition unseres «States», welches wir mit dem `useState()`-Hook definieren. Wir setzen den «State» hier einfach zu einem leeren Array.

Mit «Object Destructuring» können wir zwei Konstanten aus `useState()` ziehen, die wir manipulieren können, um den virtuellen DOM anzupassen (`feedbacks` für die Daten, `setFeedbacks` zum Update der Daten).

Der nächste «Hook» der zu sehen ist, ist der `useQuery()`-Hook auf Zeile drei. Dies ist keine Funktion von React, sondern ein «Hook» der von «Apollo» zur Verfügung gestellt wird (Apollo Documentation, 2022).

Schliesslich gibt es noch den `useEffect()`-Hook, welcher für die «Lifecycles» verwendet wird. Dieser wird in einer «Arrow-Function» definiert, wobei innerhalb der Funktion dann die Logik folgt. In einem Array wird der Hook dann konfiguriert. Mein `useEffect()`-Hook im Beispiel wird immer dann ausgeführt, wenn sich die Werte `loading`, `error` oder `data` ändern, in diesem Sinne also bei jedem Update. Würde man den Array leer lassen, würde die Funktion nur beim Laden der Komponente ausgeführt werden (on Mount).

7.6. Handling Events

Der Umgang mit «Events» ist mit React Elementen ziemlich ähnlich zu dem mit DOM-Elementen, es gibt jedoch einige Unterschiede in der Syntax:

- camelCase Notation statt Kleinbuchstaben
- Funktion wird mit JSX übergeben, nicht als String

Im folgenden Ausschnitt ist ein Font-Awesome Element aus `components/common/icons/like.jsx` abgebildet. Font-Awesome ist eine Icon-Bibliothek, über die man mit dem `i`-Tag (HTML) diverse Icons abbilden kann (Font Awesome Documentation, 2022). Die Klasse bestimmt das Aussehen des Icons (z. B. heart, info etc.).

```
<i onClick={onClick} className={classes} aria-hidden="true" />
```

Der `i`-Tag ist im Grunde wie ein `button`-Tag, man kann also ein `onClick`-Event auslösen. Wie man sieht, nutze ich hier die camelCase Notation, ausserdem nutze ich JSX um Variablen (`classes`) und Funktionen (`onClick`) zu übergeben.

Zum Vergleich habe ich den Tag wieder umgeschrieben, diesmal in normales HTML. Wir nutzen Strings zur Übergabe, ausserdem ist alles kleingeschrieben. Die Klasse ist hier keine Variable mehr sondern Hardcode. Das bedeutet, dass man diese Klassen nicht verändern kann, ausser man manipuliert das DOM-Element direkt, was in React nicht erlaubt ist. Mit JSX können wir Klassen konditionell rendern, also beispielsweise bei Klick auf den Like-Button, soll eine Klasse angefügt werden, die das Herz rot macht.

```
<i onclick="onClick()" class="clickable heartIcon fa fa-heart" aria-hidden="true" />
```

7.7. Conditional Rendering

Im letzten Beispiel wurde «Conditional Rendering» bereits angedeutet. In Font-Awesome kann man bei der Klasse «`fa-heart`» einfach ein «`-o`» anfügen, dann besteht das Herz nur noch aus Outlines.



class: fa-heart-o



class: fa-heart

Das ist der ideale Fall, um von «Conditional Rendering» Gebrauch zu machen.

Im folgenden Ausschnitt sieht man wieder Code aus der `like.jsx` Komponente. Eine Variable namens «`classes`» wird definiert, diese enthält alle CSS-Klassen, die wir für unser Styling brauchen. In der zweiten Zeile nutze ich den «Ternary Operator». Der «Ternary Operator» repräsentiert im Grunde eine If-Anweisung. In diesem Fall definiere ich eine Konstante, welche prüft, ob «`user`» eingeloggt ist. Falls ja, wird die JavaScript Funktion `find()` ausgeführt, ansonsten wird «`null`» zurückgegeben.

Ist der User also eingeloggt, wird die `find()`-Funktion ausgeführt, welches die likes mit denen des Users abgleicht. In der letzten Zeile ändern wir dann einfach unsere Variable «`classes`» dementsprechend, also wird das «`-o`» bspw. entfernt, wenn der User ein Like setzt.

```
let classes = "clickable heartIcon fa fa-heart";
const findById = user ? likes.find((id) => id == user.id) : null;
if (!findById) classes += "-o";
```

8. Strapi

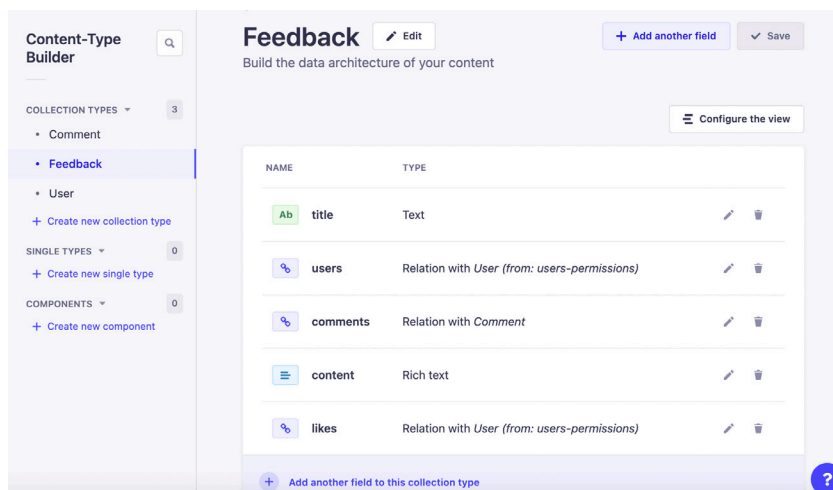
Für das Backend habe ich Strapi benutzt, ein CMS und eine Webframework, das als Open-Source-Software entwickelt wird. Strapi ist kein traditionelles CMS, sondern ein sogenanntes Headless-CMS. Es stellt somit kein eigenes Frontend bereit, sondern lediglich eine Programmierschnittstelle (API), über die man seine Daten an die Anwendung schicken kann (Strapi Documentation, 2022).

Standardmässig nutzt Strapi MongoDB als Datenbank. Strapi kann einfach über die CLI installiert werden (mit npm oder yarn), in der Dokumentation von Strapi ist die Installation ausführlich erklärt.

8.1. Content-Types

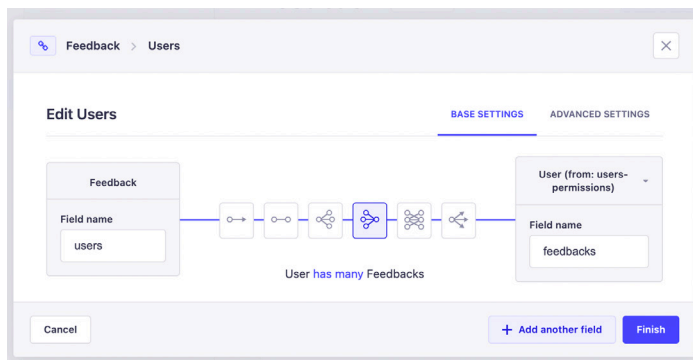
Im «Content-Type Builder» von Strapi kann man sogenannte Kollektionen erstellen. Eine Kollektion entspricht einem Block von verschiedenen Feldern, es repräsentiert unsere Datenarchitektur. Bereits in der Konzeption haben wir definiert, welche Kollektionen für die Umsetzung notwendig sein werden.

Im unteren Bild betrachten wir die Architektur der Feedback-Kollektion. Sie besteht aus den Feldern «title» und «content» für Text und den diverser Relationen.



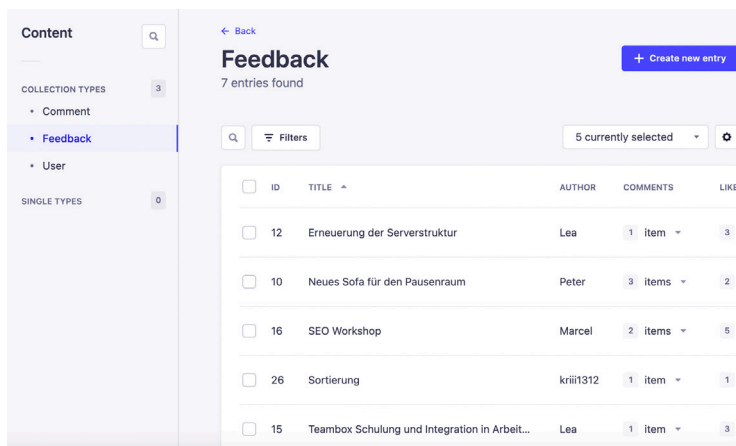
Relationen repräsentieren im Grunde die Beziehung zwischen Tabellen und Datensätzen in einer Datenbank. In Strapi können wir Relationen nutzen, um unsere «Content-Types» miteinander zu verknüpfen.

Im nächsten Bild sehen wir die Relation zwischen den Usern und unseren Feedbacks, eine sogenannte «One-To-Many» Relation. Ein User kann also mehrere Feedbacks schreiben, ein Feedback kann aber nicht von mehreren Usern verfasst worden sein.



8.2. Content Manager

Haben wir unsere Kollektionen erstellt und die Datenarchitektur definiert, so kann man die Kollektionen im «Content Manager» mit Inhalt befüllen. Dieser Bereich wäre für den Betreiber der App erreichbar, sodass er Änderungen im Inhalt vornehmen kann, User blocken kann etc. Der «Content-Type Builder» aus dem vorherigem Kapitel ist für den Moderator nicht mehr erreichbar. Dies aus Sicherheitsgründen, da eine Veränderung der Datenarchitektur verheerende Folgen in der App haben kann, sodass unter Umständen Daten verloren gehen oder die App abstürzt.





8.3. Roles

Um über das Frontend Änderungen im Inhalt vorzunehmen spricht man die API von Strapi an. Strapi bietet von Haus aus Endpoints, die man über die REST API ansprechen kann (Strapi Documentation REST API, 2022).

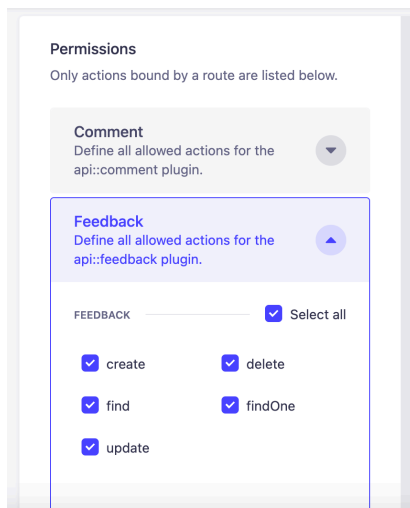
Strapi bietet ausserdem die Möglichkeit, GraphQL als Plugin zu installieren, sodass man statt der REST API auch GraphQL nutzen kann.

In beiden Fällen muss man in Strapi Rollen definieren. In den Rollen werden Berechtigungen definiert, welche Nutzer der App haben sollten. Standardmässig sind bereits zwei Rollen vorhanden, wobei aber weitere erstellt werden können.

Authenticated regelt die Berechtigungen von Usern, die auf der Website eingeloggt sind. Public regelt die Berechtigung aller anderen.

NAME	DESCRIPTION	USERS	
Authenticated	Default role given to authenticated user.	5 users	
Public	Default role given to unauthenticated user.	0 users	

Innerhalb der Rollen kann man die verschiedenen Berechtigungen für seine Plugins und Kollektionen definieren. In der unteren Abbildung sind die Berechtigungen von Authenticated zu sehen. Ist ein User also eingeloggt, ist er berechtigt, neue Feedbacks zu erstellen, zu löschen, zu suchen etc.



Permissions
Only actions bound by a route are listed below.

Comment
Define all allowed actions for the api::comment plugin.

Feedback
Define all allowed actions for the api::feedback plugin.

FEEDBACK ☒ Select all

- ☒ create
- ☒ delete
- ☒ find
- ☒ findOne
- ☒ update

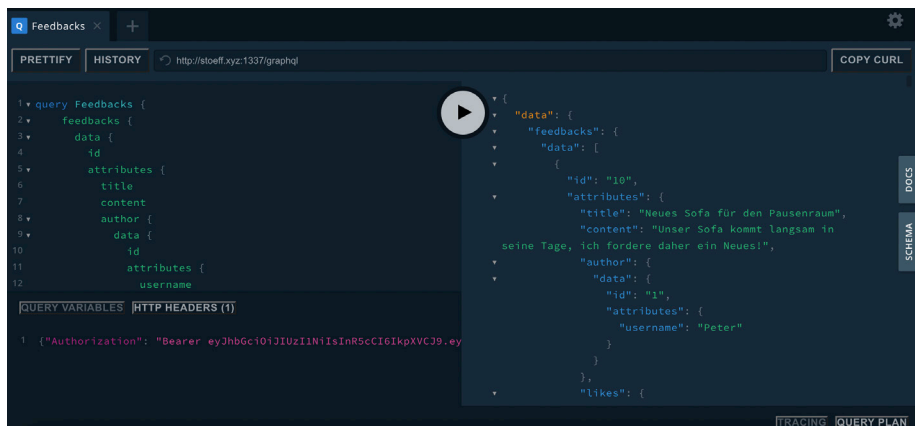
Diese Implementation hat ein Problem, welches ausgenutzt werden könnte. Die Rolle Authenticated ist nicht User-spezifisch. Das heisst, obwohl man im Frontend nur Feedbacks löschen kann, die man selbst erstellt hat, so hat man im Backend doch die Berechtigung, jedes beliebige Feedback zu löschen. Spricht man den API-Endpoint also direkt an, könnte man allen Inhalt der App löschen; man muss lediglich eingeloggt sein, resp. einen gültigen jwt-Token im Header vorweisen.

Um dieses Problem zu lösen, müsste man die Module im Backend bearbeiten, es würde also dann schon ins Backend Development gehen. In der Strapi Dokumentation gibt es eine «isOwner Policy», mithilfe derer man eine solche Funktionalität umsetzen könnte (isOwner Strapi Documentation, 2022).

Eine solche Aufgabe wäre ein eigenes Projekt in sich, vor allem da Strapi kürzlich ein Update hatte. Die Dokumentation ist also nicht auf dem neuesten Stand, in der isOwner Policy gab es

8.4. GraphQL

Nach der Installation mit Strapi hat man direkt einen API-Endpoint für einen Playground, bei dem man erste Datenabfragen testen kann. In der unteren Abbildung sieht man links die Abfrage für alle Feedbacks, in der rechten Spalte wird das Ergebnis angezeigt.



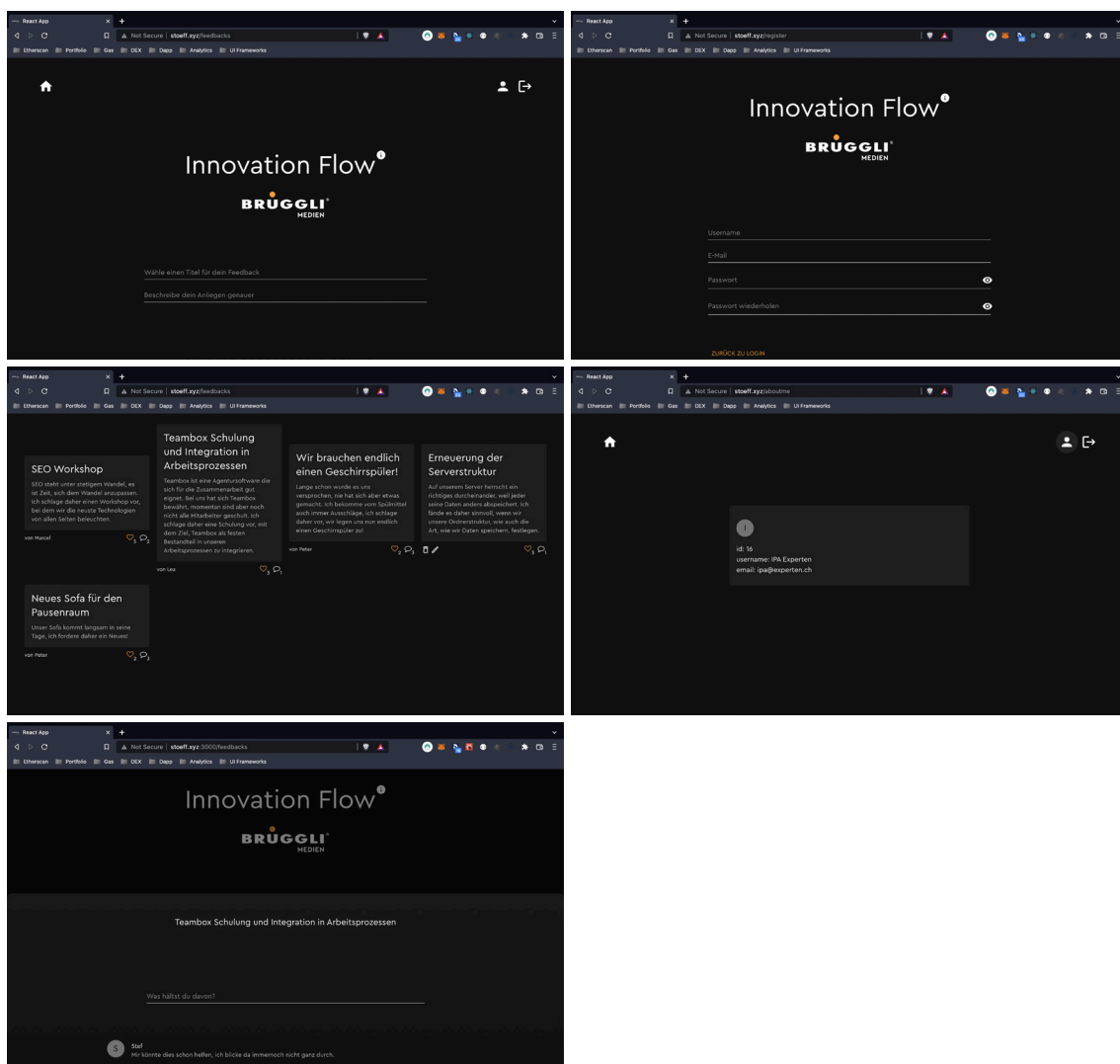
Wie bereits erwähnt, bietet Strapi API-Endpoints. Im Frontend muss man dann sogenannte «HTTP-Requests» durchführen, um Daten aus dem Server zu lesen oder zu manipulieren. Mit GraphQL kann ich lediglich die Struktur der Daten bestimmen.

Ich benutze für mein Projekt einen weniger bekannten, aber dafür sehr mächtigen Provider namens Apollo. Apollo bietet eine eigene Hooks wie `useQuery()` und `useMutation()`, mit denen man Daten abfragen und manipulieren kann. Beim Arbeiten mit «Functional Components» in React, wie ich es mache, ist dies wirklich nützlich. Des Weiteren ist es ein leichtes, GraphQL-Queries in Apollo zu integrieren. Die Dokumentation deckt alle Themen ausführlich ab (Apollo Client Documentation, 2022).

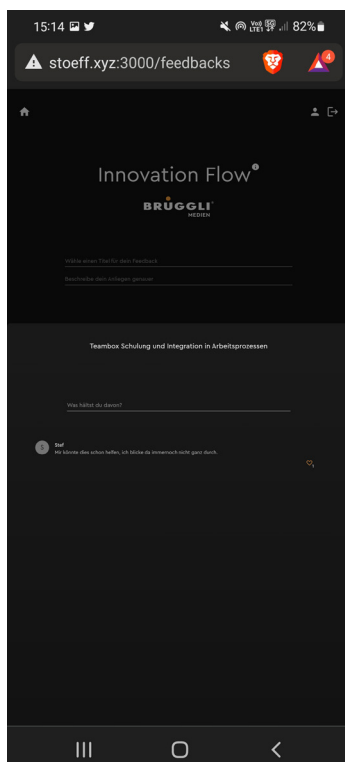
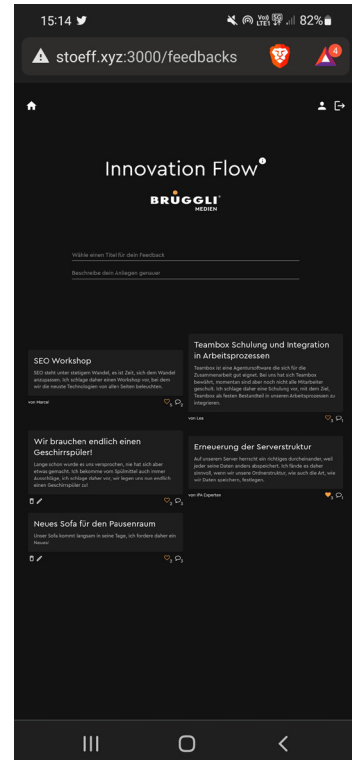
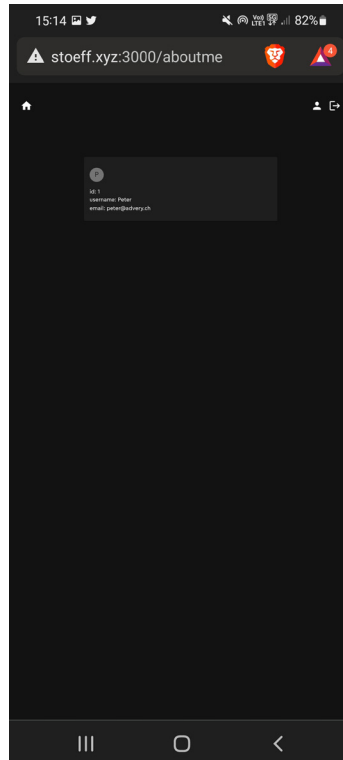
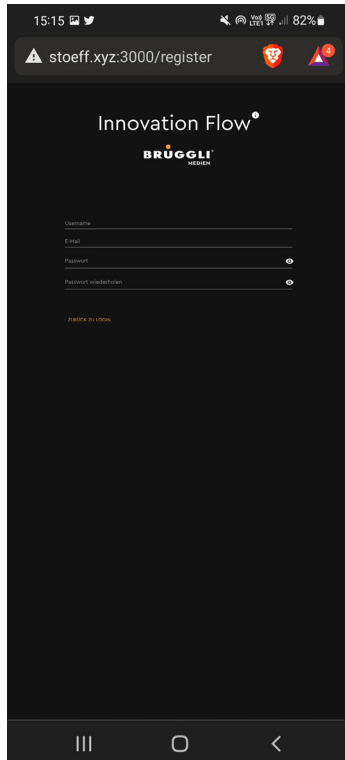
9. Darstellung auf Endgeräten

Da ich MUI als CSS-Framework verwendet habe, ist die Darstellung auf verschiedenen Endgeräten in der Regel bereits ziemlich gut umgesetzt. Ich habe die Website während der Umsetzung stetig getestet und ausgebessert. Im folgenden sind ein paar Screenshots der Website auf verschiedenen Geräten zu sehen.

Laptop MacBook Air (2560 x 1600)



Samsung Galaxy S21 (1080 x 2400)

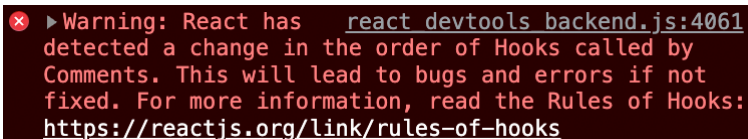


10. Probleme

Bevor man die App effektiv im Unternehmen nutzen könnte, müssten noch einige Probleme angegangen werden.

10.1. Hook Order

Laut Journal stiess ich am 02. März an ein Problem mit der Kommentarspalte. Die App funktionierte ohne Strapi im Hintergrund problemlos, nun war ich dabei, meine Dummy-Daten mit den richtigen Daten von Strapi zu ersetzen. Dies funktionierte auch soweit für die Feedbacks, bei den Kommentaren stiess ich aber auf ein Problem, dass mich etwas Zeit gekostet hatte.



```
⌘ ▶Warning: React has react_devtools_backend.js:4061  
detected a change in the order of Hooks called by  
Comments. This will lead to bugs and errors if not  
fixed. For more information, read the Rules of Hooks:  
https://reactjs.org/link/rules-of-hooks
```

Wie im letzten Kapitel bereits erwähnt, nutzte ich Apollo für die Datenabfrage. Apollo bietet eigene Hooks, um Daten abzufragen und zu manipulieren. Es gibt gewisse Regeln beim Arbeiten mit Hooks, die es zu befolgen gilt (Hook Rules React Documentation, 2022).

Mein Fehler lag in der Reihenfolge der Hooks, man kann sie nicht ineinander verschachteln. Korrekterweise wurde ich also in der Fehlermeldung auf die Dokumentation verwiesen.

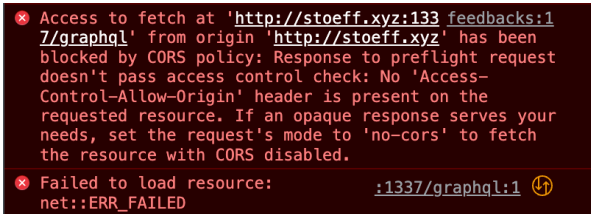
Ich musste also alle Hooks auf die gleiche Ebene bringen, sodass es in den Kinderelementen keine Fehler mehr gibt. Obwohl die App nun soweit funktioniert, wäre es dennoch empfehlenswert, die App nochmals gründlich zu überarbeiten.

Speziell also die Reihenfolge von Hooks innerhalb der Komponenten, und allgemeines «State Management» sind Themen, denen man noch nachgehen müsste.

10.2. CORS

Dies ist ein Problem, dem ich während der IPA nicht mehr auf den Grund gehen konnte. Eventuell bedürfte es sogar interne Gespräche mit unserer Informatik.

Meine App funktioniert zurzeit nicht im Netzwerk vom Brüggli. Über mein Handy oder auch Zuhause ist die App erreichbar. Sobald ich mich aber mit dem W-Lan des Brügglis verbinde, bekomme ich folgenden CORS-Fehlermeldung (nächste Seite).



Ich habe versucht, den richtigen Header zu setzen, aber es scheint nicht zu funktionieren. Unser Entwickler meinte ebenfalls, dass CORS-Probleme ziemlich komplex werden können, und es dann eventuell mehr braucht. Ich konnte diese Problem daher nicht lösen.

Natürlich müsste man das aber angehen, bevor man die App live schalten würde. Es ist nicht ideal, dass man eine Feedback-App für die eigene Firma, nicht im eigenen Netzwerk erreichen kann.

10.3. Security

Bereits im Kapitel 8.3 Roles habe ich das Problem der Authenticated-Role angesprochen. Ist jemand dem Brüggli also Böse gesinnt, wäre es nicht mal schwer, die App unbrauchbar zu machen. Es bräuchte also noch einiges an Backend Development.

Ein Vorteil von Strapi ist aber, dass es ein Headless CMS ist, und im Grunde sind Headless CMS sicherer, einfach weil das Backend und das Frontend voneinander getrennt sind.

In diesem Sinne wäre es daher auch ratsam, beim Hosting das Frontend vom Backend zu trennen. Momentan ist die Website über meine private Domain erreichbar, welche ich auf einem Server von Hetzner gehostet habe (Hetzner Documentation, 2022). Frontend und Backend befinden sich somit auf dem gleichen Server.

Innerhalb der App müssten die Rechte angepasst werden, dies muss aber genau geplant werden, eventuell sogar zusammen mit der Informatik des Brüggli. Ich habe die Rollen «Authenticated» und «Public» von Strapi benutzt, es gibt aber die Möglichkeit, noch eigene zu erstellen.

Dies wäre gerade dann wichtig, wenn wir die App bspw. zu verschiedenen Abteilungen zuweisen. Das Brüggli ist gross und hat sogar verschiedene Standorte, eine entsprechende Rollenverteilung im Backend würde daher Sinn machen. Der Chef der Logistik sollte nicht ein Feedback löschen können von einem Koch, jedoch aber von seinem Azubi, der die App nur dafür missbraucht, um Dampf abzulassen.

11. Ausblick

Wir haben bereits Probleme angesprochen, die noch gelöst werden müssten, bevor man die App effektiv live schaltet. In einem weiteren Schritt könnte man die App dann ausbauen, verbessern und um diverse Funktionalitäten erweitern.

11.1. Security

Wir haben uns bereits mit dem Problem der Authenticated-Role befasst. Dies repräsentiert nur eine potentielle Gefahrenquelle. Security ist aber ein riesiges Thema, für das es unter Umständen sogar Spezialisten braucht. Neben der Backend Security, die man verbessern und ausbauen könnte, gibt es auch im Frontend Verbesserungspotential, denn auch da gibt es Sicherheitsbedrohungen. Im Web findet man schnell Ansätze, um seine React-App sicherer zu machen (Shachee Swadia, 2021).

11.2. Scaling

Ein weiteres, wichtiges Thema wäre die Skalierung der App. Es wurde bereits erwähnt, dass man aus Sicherheitsgründen die Datenbank (Strapi) vom Frontend trennen soll (beim Hosting). Dies kommt natürlich auch der Performance der App zugute.

Es gibt des Weiteren eine Reihe von Dingen die man tun könnte, um die Performance zu verbessern. Es könnte beispielsweise notwendig werden, eine State-Management-Library wie Redux zu verwenden, gerade wenn die App grösser wird. Auch bzgl. der Skalierung finden sich im Web Ansätze, die man weiter verfolgen könnte (React Proxify Scale React Apps, 2021).

Ein interessanter Ansatz finde ich das Nutzen von Design Patterns, welche sogar von einem User schon für React zur Verfügung gestellt werden (React Design Patterns Github, 2021), sodass man die Logik nicht selbst aufbauen muss.

Die Grundlage für Design Patterns liefert das Buch «Elements of Reusable Object-Oriented Software» aus den 1990er-Jahren, geschrieben von den «Gang of Four», kurz GoF. Jeder gute Entwickler sollte sich früher oder später mit Design Patterns befassen. Sie liefern Lösungen für wiederkehrende Probleme in Softwaredesign. GoF hat in ihrem Buch 23 Design-Patterns definiert, welche in die Kategorien «Creational Patterns», «Structural Patterns» und «Behavioral Patterns» aufgeteilt werden (Design Patterns, GoF, 1995)

11.3. Datenschutz, Anonymität und rechtliche Rahmenbedingungen

Das Brüggli ist ein Ausbildungs- und Integrationsbetrieb, der von der Invalidenversicherung der Schweiz finanziert wird. Da unsere Firma viele Menschen mit körperlichen oder psychischen Beeinträchtigungen beschäftigt, haben wir strenge, interne Datenrichtlinien.

Man müsste also einiges abklären, einerseits mit der Informatik bzgl. Datenschutz, dann müsste man wahrscheinlich noch die Geschäftsleitung des Brüggli über die App informieren. Nicht zuletzt müssten eventuell sogar die Invalidenversicherung resp. die Kantone ihre Zustimmung geben, je nachdem in welchem Umfang wir Daten von den Klienten sammeln.

Das bedeutet nicht, dass wir gezielt Daten sammeln werden, gerade aber wenn man die App um weitere Funktionalitäten erweitert und es zu einer Social-Media App ausarbeitet, müsste dies abgeklärt werden.

11.4. weitere Funktionalitäten

Wie bereits angedeutet, könnte man die App zu einer richtige Social-Media App ausarbeiten. Der Gedanke war schon zu Beginn, die App wie eine Social-Media App wirken zu lassen.

Wir beschäftigen sehr viele junge Menschen, und genau diese wollen wir ja ansprechen. Folgend habe ich bereits eine Liste erstellt, mit möglichen Funktionalitäten, die man noch umsetzen könnte. Natürlich ist das Ganze beliebig erweiterbar.

- Bilder in Feedbacks hinzufügen
- Profilbilder
- Anonymes Board (ohne Anmeldung)
- E-Mail Bestätigung
- Reset Password / Change Password / Change Email
- Emojis
- Reactions mit Emojis
- Timeline in About Me
- Userprofile erweitern
- Feedback-Kategorien
- Filtern nach Kategorien
- Paginierung
- Öffentliche Profile
- Profilbeschreibung
- Messenger
- Themeing
- Live Fetch
- Notifications
- Ups- and Downvotes
- Mobile App mit React Native
- Testing (Jest, React Testing Library)
- Auth and Security
- Next.js
- isOwner Policy
- Timer bei edit/delete

12. Projektmanagement

Die meisten Dokumente zum Projektmanagement sind im Anhang zu finden. Als Grundlage resp. das Briefing zum Projekt ist meine Zusammenfassung aus der Innovation-Flow Sitzung. Der Zeitplan wie auch der Soll/Ist Vergleich finden sich ebenfalls im Anhang.

12.1. Zielüberprüfung

In der Eingabe wie auch im Kapitel 4.4 Zieldefinition wurden die Kriterien, die es zu erreichen gilt, definiert. Im folgenden prüfen wir, ob die Ziele erreicht wurden.

MUSS-Kriterien:

- ✓ Lokale Umsetzung in Strapi + React.js (inkl. HTML/CSS)
- ✓ Kurzes Konzept der Webseite
- ✓ Screendesign in Adobe xD
- ✓ Übersicht der Feedbacks
- ✓ Hinzufügen, Bearbeiten und Löschen von Feedbacks
- ✓ Hosting auf einem Server (ohne Login im internen Netzwerk)

KANN-Kriterien:

- ✓ Login-Interface
- ✓ Liken und Kommentieren von Feedbacks

Alle Ziele wurden erreicht und die App funktioniert soweit einwandfrei.

12.2. Kostenschätzung

Im folgenden gehen wir auf die Kostenschätzung ein. Neben der Arbeitszeit gab es noch weitere Kosten.

Entwicklung und Umsetzung	80.5 h x 90 Fr. = 7245 Fr.
Hilfestellung von Webentwickler zu CORS	0.5 h x 150 Fr. = 75 Fr.
Besprechungen mit Vorgesetzter	1.5 h x 150 Fr. = 225 Fr.
MUI Adobe xD Template	65 Fr.
Wirobindung	30 Fr.
Total	7640 Fr.

13. Reflexion

In dieser Arbeit habe ich jegliche Konzepte in React resp. JavaScript geübt, vertieft und erfolgreich angewendet. Ausserdem habe ich den Umgang mit dem Headless CMS Strapi vertieft und mich mit der Integration von GraphQL befasst. Es gibt mir ein gutes Gefühl und es macht mich stolz, ein funktionierendes Produkt entwickelt zu haben.

Die letzten Wochen waren stressig und es brauchte einiges an Konzentration, was selbstverständlich ist bei einer wichtigen Arbeit in diesem Umfang. Dies stellte für mich auch die grösste Hürde dar.

Den obwohl ich weiss, dass ich brauchbare Qualität liefern kann, so ist es der Stress und Druck, der mir zu schaffen macht. Und obwohl es einen Fehltag gab, so konnte ich die Arbeit trotzdem erfolgreich abschliessen, was ein Erfolg ist.

Ich stiess natürlich immer wieder auf technische Probleme während dem Programmieren, die ich aber alle mithilfe von Dokumentationen und dem Web lösen konnte. Nichtsdestotrotz müssten einige Dinge angegangen werden, die aber in der Dokumentation erwähnt wurden.

Fachlich gesehen gab es in diesem Sinne keine Probleme. Ich konnte mir im Zeitplan gar etwas Vorsprung erarbeiten, sodass diese Zeit dann in die Dokumentation einfliessen konnte. Dies stellte sich als ideal heraus, denn ich hatte zu wenig Zeit für die Dokumentation reserviert.

In Ausblick auf die Zukunft hoffe ich natürlich, dass man die App noch weiter ausbauen und effektiv nutzen wird. Ich denke für einen Entwickler gibt es keine bessere Bestätigung als die Adoption der eigenen App. Bereits zehn aktive User wären einen grossen Erfolg in meinen Augen.

14. Literaturverzeichnis

MUI Documentation. (2022).

von mui.com: <https://mui.com/getting-started/installation>

Anumadu Moses. (2021).

von strapi.io: <https://strapi.io/blog/a-deep-dive-into-strapi-graph-ql>

Strapi and GraphQL Documentation. (2022).

von docs.strapi.io: <https://docs.strapi.io/developer-docs/latest/developer-resources/content-api/integrations/graphql.html#install-the-graphql-plugin>

React Router v6 Documentation. (2022).

von reactrouter.com: <https://reactrouter.com/docs/en/v6>

Apollo Client Documentation. (2022).

von apollographql.com: <https://www.apollographql.com/docs/react/get-started/>

React Toastify Documentation. (2022).

von github.io: <https://fkhadra.github.io/react-toastify/introduction>

Font Awesome Documentation. (2022).

von fontawesome.com: <https://fontawesome.com/docs/web/use-with/react/>

MUI Theming. (2022).

von mui.com: <https://mui.com/customization/theming/>

React Documentation. (2022).

von reactjs.org: <https://reactjs.org/>

Strapi Documentation. (2022).

von strapi.io: <https://strapi.io/>

Visual Studio Code. (2022).

von code.visualstudio.com: <https://code.visualstudio.com/>

JOI Documentation. (2022).

von joi.dev: <https://joi.dev/>

ECMAScript6 Einführung. (2021).

von wiki.selfhtml.org: <https://wiki.selfhtml.org/wiki/JavaScript/Tutorials/ES6>

Mastering React, Mosh Hamedani. (2022).

von codewithmosh.com: <https://codewithmosh.com/p/mastering-react>

Hook Rules React Documentation. (2022).

von reactjs.org: <https://reactjs.org/docs/hooks-rules.html>

Thinking in React. (2022).

von reactjs.org: <https://reactjs.org/docs/thinking-in-react.html>

Strapi Documentation REST API. (2022).

von docs.strapi.io: <https://docs.strapi.io/developer-docs/latest/developer-resources/database-apis-reference/rest-api.html#api-parameters>

Fetch API Mozilla Documentation. (2022).

von developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

NPM Axios Package. (2022).

von npmjs.com: <https://www.npmjs.com/package/axios>

isOwner Policy Strapi Documentation. (2022).

von docs.strapi.io: <https://docs.strapi.io/developer-docs/latest/guides/is-owner.html>

Github Issue Report Strapi, missing isOwner Policy. (2021).

von github.com: <https://github.com/strapi/documentation/issues/600>

Hetzner Documentation. (2022).

von docs.hetzner.com: <https://docs.hetzner.com/de/>

Shachee Swadia. Best Practices for React Security. (2021).

von freecodecamp.org: <https://www.freecodecamp.org/news/best-practices-for-security-of-your-react-js-application/>

React Proxify Scale React Apps. (2021).

von proxify.io: <https://proxify.io/articles/build-large-scale-React-apps>

React Design Patterns Github. (2021).

von github.com: <https://github.com/themithy/react-design-patterns>

Design Patterns. GoF. (1995). Elements of Reusable Object-Oriented Software. Gang of Four:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison Wesley Verlag.

15. Anhänge

- IPA-Eingabe
- Komponenten-Baum
- Zusammenfassung aus Innovation-Flow Sitzung
- Screendesign
- Arbeitsjournal
- Zeitplan (grob und detail)
- Soll/Ist Vergleich
- Besprechungsprotokolle

nur elektronisch:

- CI/CD Brüggl Medien
- CI/CD Advery
- Device Mockup
- JSX und JS-Files aller Komponenten (Code)

16. Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst, alle beanspruchten Hilfsmittel und Quellen angegeben und keine unerlaubten Hilfen in Anspruch genommen habe. Ich habe zur Kenntnis genommen, dass bei Verstößen gegen diese Erklärung ein Abzug vorgenommen wird.

Romanshorn, 11.03.2022

