

# E-Commerce Shop

EBUSINESS APPLIKATION

ISMAL SOLOMBRINO | LUCIANO ALAGIA | ABRAHAM LEGUIZAMON

## Inhaltsverzeichnis

1	Beschreibung.....	2
2	Konzept .....	3
2.1	MERN .....	3
2.2	React.js Frontend .....	3
2.3	Express.js und Node.js Server Tier .....	3
2.4	MongoDB-Datenbank-Tier .....	3
2.5	Redux .....	4
3	Frontend.....	5
3.1	Actions .....	5
3.2	Components.....	7
3.3	Constants .....	9
3.4	Reducers.....	10
3.5	Screens.....	11
3.6	Store.....	20
3.7	Utils .....	20
4	Backend.....	20
4.1	Models .....	20
4.2	Routers.....	20
4.3	Server .....	21
4.4	Utils .....	21
5	Schlusswort.....	22

## 1 Beschreibung

In dieser Dokumentation geht es um die Umsetzung von Konzepten eines minimalen Shops. Es geht nicht darum einen kompletten Shop zu programmieren, sondern einige Aspekte eines Onlineshops genauer zu betrachten und umzusetzen. Diese Projektarbeit wurde von Solombrino Ismail, Abraham Infante Leguizamon und Luciano Alagia umgesetzt.

Unser GitHub Projekt

<https://github.com/SolombrinoIsmail/E-Commerce>

Clonen Sie das Projekt oder laden Sie sich ein ZIP-File herunter.

Öffnen Sie das Projekt in einer Entwicklungsumgebung, wie IntelliJ oder Visual Studio Codes. Um das Projekt zu starten, gehen Sie ins Terminal und wechseln Sie in das Unterverzeichnis «Backend», und tippen Sie `npm install` und `npm start`. Somit wird die Verbindung im Backend aufgesetzt. Um das Frontend zu starten, öffnen Sie ein neues Terminal und gehen Sie in das Frontend Unterverzeichnis, und tippen Sie wieder `npm install` und `npm start`.

Falls der Output folgendermassen erscheint hat alles richtig funktioniert.

```
$ cd backend
$ npm install
$ npm start
```

```
$ cd frontend
$ npm install
$ npm start
```

### Backend

```
added 370 packages from 254 contributors and audited 371 packages in 45.568s

50 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): backend\**\*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node --experimental-modules backend/server.js`
Serve at http://localhost:5000
```

npm install

npm start

### Frontend

```
added 1662 packages from 802 contributors and audited 1667 packages in 106.663s

81 packages are looking for funding
  run `npm fund` for details

found 2 high severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details

i [wds]: Project is running at http://172.24.80.1/
i [wds]: webpack output is served from
i [wds]: Content not from webpack is served from C:\Users\isy-s\
i [wds]: 404s will fallback to /
Starting the development server...
```

npm install

npm start

## 2 Konzept

### 2.1 MERN

Unser Konzept basiert auf das MERN Stack. MERN steht für MongoDB, Express, React, Node nach den vier Schlüsseltechnologien, aus denen sich das Stack zusammensetzt.

- MongoDB - Dokumentendatenbank
- Express (.js) - Node.js Webframework
- React (.js) - ein clientseitiges JavaScript-Framework
- Node (.js) - der führende JavaScript-Webserver

### 2.2 React.js Frontend

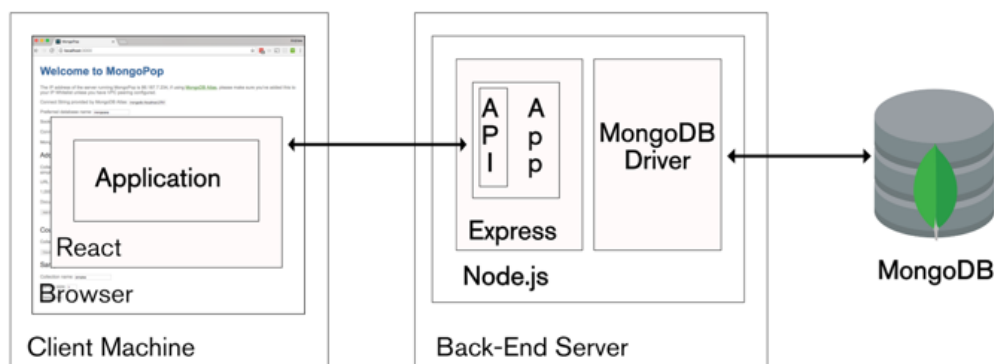
Die oberste Schicht des MERN-Stacks ist React.js, das deklarative JavaScript-Framework zur Erstellung dynamischer clientseitiger Anwendungen in HTML. Mit React können Sie komplexe Oberflächen durch einfache Komponenten aufbauen, sie mit Daten auf Ihrem Backend-Server verbinden und sie als HTML rendern. Reacts Stärke ist der Umgang mit zustandsbehafteten, datengesteuerten Schnittstellen mit minimalem Code und minimalem Aufwand, und es hat all den Schnickschnack, den man von einem modernen Web-Framework erwartet: großartige Unterstützung für Formulare, Fehlerbehandlung, Ereignisse, Listen und mehr.

### 2.3 Express.js und Node.js Server Tier

Die nächste Ebene darunter ist das serverseitige Express.js-Framework, das innerhalb eines Node.js-Servers läuft. Express.js bewirbt sich selbst als "schnelles, rechthaberisches, minimalistisches Web-Framework für Node.js", und genau das ist es auch. Express.js verfügt über leistungsfähige Modelle für das URL-Routing (Abgleich einer eingehenden URL mit einer Serverfunktion) und den Umgang mit HTTP-Anfragen und -Antworten. Indem Sie XML-HTTP-Anfragen (XHRs) oder GETs bzw. POSTs von Ihrem Angular.js-Frontend aus durchführen, können Sie eine Verbindung zu Express.js-Funktionen herstellen, die Ihre Anwendung betreiben. Diese Funktionen wiederum verwenden die Node.js-Treiber von MongoDB, entweder über Callbacks oder über Promises, um auf Daten in Ihrer MongoDB-Datenbank zuzugreifen und diese zu aktualisieren.

### 2.4 MongoDB-Datenbank-Tier

Wenn Ihre Anwendung Daten speichert (Benutzerprofile, Inhalte, Kommentare, Uploads, Events usw.), dann wollen Sie eine Datenbank, die genauso einfach zu bedienen ist wie Angular, Express und Node. Hier kommt MongoDB ins Spiel: JSON-Dokumente, die in Ihrem Angular.js-Frontend erstellt wurden, können an den Express.js-Server gesendet werden, wo sie verarbeitet und (vorausgesetzt, sie sind gültig) direkt in MongoDB gespeichert werden können, um später abgerufen zu werden. Auch hier gilt: Wenn Sie in der Cloud bauen, sollten Sie sich Atlas ansehen.

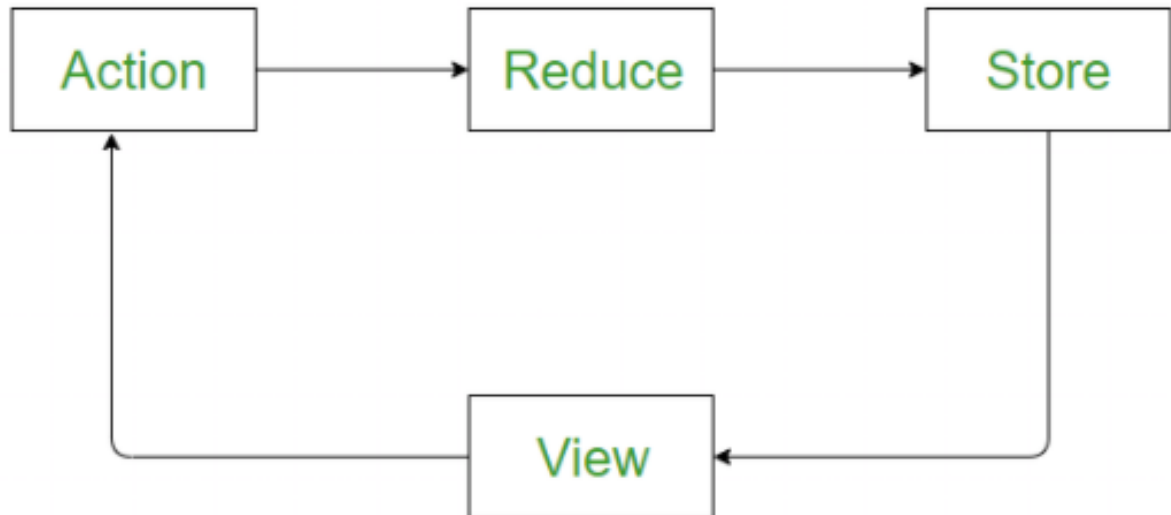


## 2.5 Redux

Redux ist eine zustandsverwaltende Bibliothek, die in JavaScript-Apps verwendet wird. Sie verwaltet einfach den Zustand Ihrer Anwendung oder mit anderen Worten, sie wird verwendet, um die Daten der Anwendung zu verwalten. Sie wird mit einer Bibliothek wie React verwendet.

Baut Teile von Redux

1. Action
2. Reducer
3. Store



## 3 Frontend

Unser Frontend wurde hauptsächlich mit React.js realisiert.

### 3.1 Actions

Aktionen sind ein einfaches JavaScript-Objekt, das Informationen enthält. Aktionen sind die einzige Quelle von Informationen für den Speicher. Aktionen haben ein Typfeld, das angibt, welche Art von Aktion «Befehl» ausgeführt werden soll, und alle anderen Felder enthalten Informationen oder Daten. Und es gibt noch einen weiteren Begriff, der sich Action Creators nennt, das sind die Funktionen, die Aktionen erstellen. Aktionen sind also die Informationen (Objekte) und Action Creator sind Funktionen, die diese Aktionen zurückgeben.

#### 3.1.1 Cart

In unserem Cart.js haben wir theoretisch 5 folgende Aktionen, wobei CART\_ADD\_ITEM\_FAIL nur eine Warnung ist.

- CART\_ADD\_ITEM  
Die addToCart Funktion fügt ein Product dem Warenkorb «cart» hinzu.
- CART\_REMOVE\_ITEM  
Die removeFromCart Funktion entfernt ein Product aus dem Warenkorb «cart» hinzu.
- CART\_SAVE\_SHIPPING\_ADDRESS  
Die saveShippingAddress Funktion speichert die angegebene Adresse für die Lieferung.
- CART\_SAVE\_PAYMENT\_METHOD  
Die savePaymentMethod Funktion speichert die angegebene Zahlungsmethode.
- CART\_ADD\_ITEM\_FAIL  
Falls die addToCart Funktion kleiner als 0 ist, was bedeuten würde, dass da Product nicht mehr verfügbar ist dann kommt die CART\_ADD\_ITEM\_FAIL und es gibt ein Fehler aus.

### 3.1.2 Order

- **CreateOrder**  
Die Funktion CreateOrder fragt den User nach seinem Login, sobald der sich erfolgreich eingeloggt hat werden die Producte im Warenkorb weitergeleitet und vom Warenkorb entfernt. Falls der User nichts im Warenkorb hat, dann taucht ein Fehler auf. Mittels Post sendet die Funktion die Daten dem Web weiter.
- **DetailsOrder**  
Die DetailsOrder Funktion zeigt alle Details zur getätigten Bestellung, vom zugehörigen User. Mittels Get werden die Daten mit dem zugehörigen OrderId aufgerufen.
- **PayOrder**  
Die Funktion PayOrder, fragt nach einer Zahlungsmethode und leitet das paymentResult weiter. Falls das erfolgreich war, dann erscheint eine Meldung ansonst eine Warnung oder Wiederholungsanfrage.
- **ListOrderMine**  
Die ListOrderMine Funktion zeigt alle Details zur getätigten Bestellung, vom zugehörigen User. Mittels Get werden die Daten mit dem zugehörigen OrderId aufgerufen.
- **ListOrders**  
Die Funktion ListOrders zeigt alle Bestellungen, die der Seller getätigt hat.
- **DeleteOrder**  
Mittels DeleteOrder wird eine Bestellung mit der zugehörigen OrderID gelöscht.
- **DeliverOrder**  
Zum Schluss wird mit der DeliverOrder Funktion mittels Put die Bestellung weitergeleitet.

### 3.1.3 Product

- **listProducts**  
Die Funktion listProduct, listet alle Produkte auf, die im Backend gespeichert sind.
- **listProductCategories**  
Die listProductCategories Funktion listet alle Kategorien auf, die im Backend gespeichert sind.
- **detailsProduct**  
Die detailsProduct Funktion zeigt die Details eines Products auf.
- **createProduct**  
Mittels der Funktion createProduct wird ein Produkt im Backen mittels Post gespeichert.
- **updateProduct**  
die updateProduct Funktion gibt uns die möglichkeit Produkte im Backend vom Frontend aus zu manipulieren (updaten).
- **deleteProduct**  
die updateProduct Funktion gibt uns die möglichkeit Produkte im Backend vom Frontend aus zu löschen(delete).
- **createReview**  
Mit der createReview Funktion kann man einem Produkt eine Rezension schreiben.

### 3.1.4 User

- Register  
Mit der Funktion Register kann sich ein User neu registrieren, indem er Name, E-Mail und Passwort eingibt. Die Daten werden dem Backend gesendet und in der Datenbank gespeichert.
- Signin  
Durch die Signin Funktion kann sich ein bereits registrierter User mit seiner E-Mail-Adresse und Passwort anmelden.
- Signout  
Die Signout Funktion, löscht alle Daten, die dem User gehören und lädt die Signin Page wieder.
- DetailsUser  
Die DetailsUser Funktion gibt alle Daten, die einem User mit der zugehörigen ID aus.
- updateUserProfile  
Durch die updateUserProfile wird das ganze Profil eines Users nach Angaben verändert.
- updateUser  
Durch die updateUser Funktion, werden Daten mittels put in die Datenbank neu eingesetzt.
- ListUser  
Werden alle Users mittels Get von der Datenbank aufgelistet.
- DeleteUser  
Mittels DeleteUser wird ein Benutzer von der Datenbank permanent gelöscht.
- ListTopSellers  
Die Funktion ListTopSellers listet alle Sellers (Verkäufer) die als Top Sellers bezeichnet sind.

## 3.2 Components

### 3.2.1 Admin Route

Hier wird überprüft, ob der Angemeldete User eine Admin Berechtigung hat.

### 3.2.2 Checkout Steps

In die Datei CheckoutSteps werden die einzelnen Schritte für den Frontend gespeichert, um die Bezahlung richtig zu betätigen.

### 3.2.3 Loading Box

In Loading Box wird eine Loading Funktion gespeichert, dies zeigt ein "spinner" wenn die Seite noch im Browser am Laden ist.

### 3.2.4 Message Box

Durch diese Funktion wird ein Pop-Up angezeigt, falls man zum Beispiel keine Location gewählt hat und man trotzdem auf Fortfahren drückt.

### 3.2.5 Private Route

Wenn diese Funktion aufgerufen wird, werden alle Infos vom User angezeigt

### 3.2.6 Product

Durch diese Funktion werden alle Daten von einem Produkt angezeigt

### 3.2.7 Rating

Da wir noch ein Rating auf der Webseite implementiert haben brauchen wird diese Datei um das Rating zu verwalten.



### 3.2.8 Search Box

Auf dem Navbar haben wir eine Search bar implementiert. Die Funktion um ein Objekt aus der Datenbank zu suchen haben wir dann hier implementiert-

### 3.2.9 Seller Route

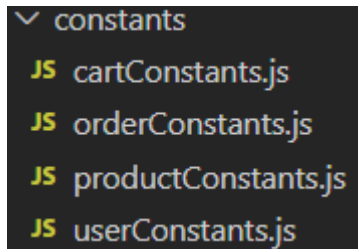
Hier wird überprüft, ob der User der sich angemeldet hat ein Seller ist. Falls es zutrifft hat er das Recht Artikel hinzuzufügen.

### 3.3 Constants

Da einige unserer Variablen/Konstanten mehrmals vorkommen haben wir die in einem Unterordner Constants mit der Jeweiligen JavaScript-File gespeichert.

Im Folgende sehen Sie ein Paar Screenshot unserer Variablen. Da es sehr viele sind beschränken wir uns auf 2 Screenshot.

Ordner Struktur



Screenshot einiger Konstanten

```
export const CART_ADD_ITEM_FAIL = 'CART_ADD_ITEM_FAIL';  
export const CART_ADD_ITEM = 'CART_ADD_ITEM';  
export const CART_REMOVE_ITEM = 'CART_REMOVE_ITEM';  
export const CART_SAVE_SHIPPING_ADDRESS = 'CART_SAVE_SHIPPING_ADDRESS';  
export const CART_SAVE_PAYMENT_METHOD = 'CART_SAVE_PAYMENT_METHOD';  
export const CART_EMPTY = 'CART_EMPTY';
```

### 3.4 Reducers

Wie wir bereits wissen, sagen Aktionen (Actions) nur, was zu tun ist, aber sie sagen nicht, wie zu tun ist. Daher sind Reducers die reinen Funktionen, die den aktuellen Zustand und die Aktion nehmen und den neuen Zustand zurückgeben und dem Speicher sagen, wie zu tun ist.

#### 3.4.1 Cart

Durch die `orderCreateReducer()` Methode werden Produkte auf dem Cart hinzugefügt, indem man Abfragen in die gleiche Methode benutzt.

#### 3.4.2 Order

Die Funktion `orderCreateReducer()` erstellt eine Abfrage, um eine Order zu erstellen.

Die Funktion `orderDetailsReducer()` ist ähnlich wie die vorherige Funktion, diese überprüft ob alle Daten richtig gespeichert werden.

#### 3.4.3 Product

In die File `productReducers` werden die Reducers erstellt, um ein Produkt zu erstellen, anzeigen oder löschen

#### 3.4.4 User

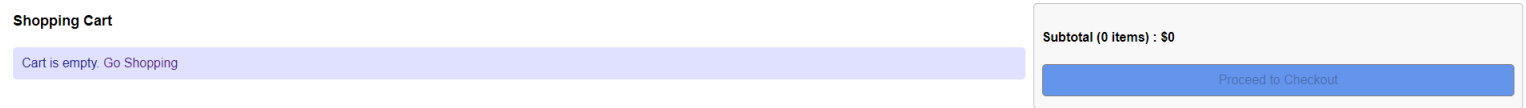
Ähnlich wie bei der Product File werden hier die Reducers erstellt, um ein User anzumelden, abzumelden oder ihn anzuzeigen.

### 3.5 Screens

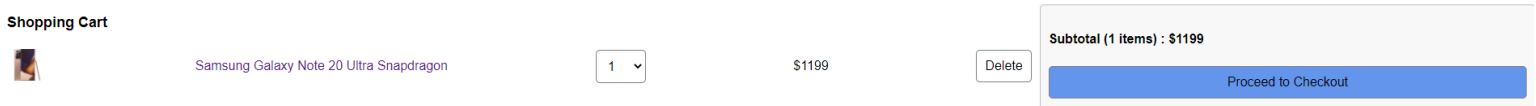
Mit Screen meinen wir die verschiedenen Pages, die die Webseite, in verschiedene Situationen anzeigt

#### 3.5.1 Cart

Normalerweise sieht das Cart so aus



Sobald man ein Product dem Warenkorb hinzufügt dann erscheint der im cart.



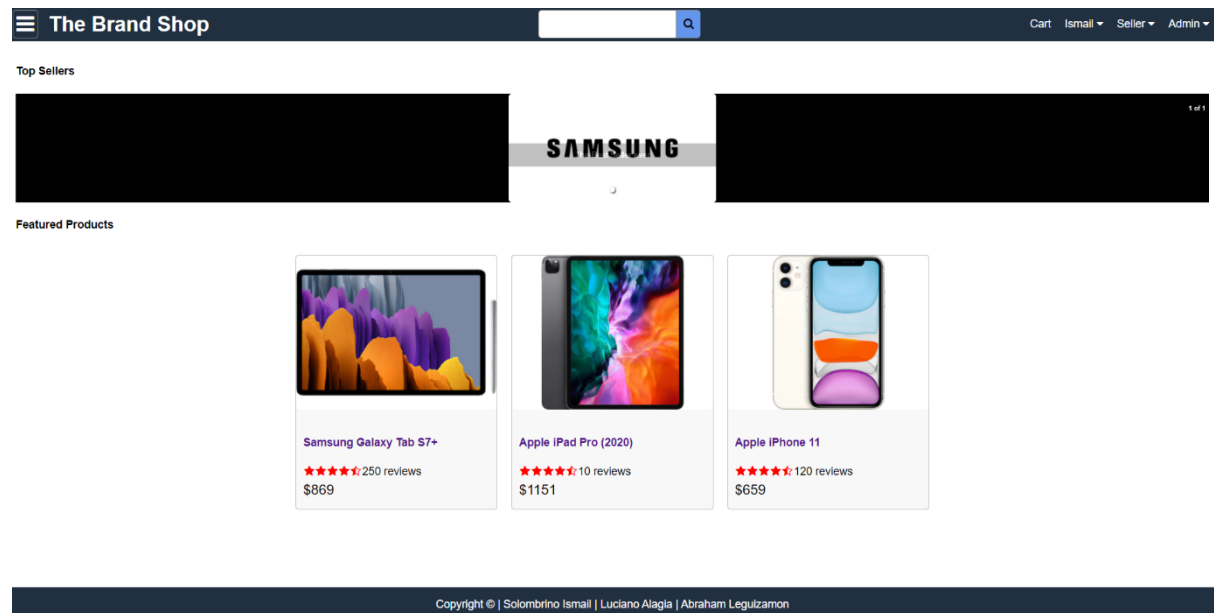
Mann kann dann mit einem Selector die Menge auswählen und mit dem Delete Button das Product löschen.

```
const cart = useSelector((state) => state.cart);
const { cartItems, error } = cart;
const dispatch = useDispatch();
useEffect(() => {
  if (productId) {
    dispatch(addToCart(productId, qty));
  }
}, [dispatch, productId, qty]);

const removeFromCartHandler = (id) => {
  // delete action
  dispatch(removeFromCart(id));
};
```

### 3.5.2 Home

Unsere Starting-Page auch Home-Page



Man sieht hier die 3 featured Produkte und oberhalb die Top Sellers, die in einem Karusell implementiert worden sind.

```
{sellers.length === 0 && <MessageBox>No Seller Found</MessageBox>}
<Carousel showArrows autoPlay showThumbs={false}>
  {sellers.map((seller) => (
    <div key={seller._id}>
      <Link to={`/${seller._id}`}>
        <img src={seller.seller.logo} alt={seller.seller.name} />
        <p className="legend">{seller.seller.name}</p>
      </Link>
    </div>
  ))}
</Carousel>

<h2>Featured Products</h2>
{loading ? (
  <LoadingBox></LoadingBox>
) : error ? (
  <MessageBox variant="danger">{error}</MessageBox>
) : (
  <>
    {products.length === 0 && <MessageBox>No Product Found</MessageBox>}
    <div className="row center">
      {products.map((product) => (
        <Product key={product._id} product={product}></Product>
      ))}
    </div>
  </>
)}
```

### 3.5.3 Order History

Auf Order History kann man seine Bestellungen und ihre Konditionen (bezahlt, geliefert ansehen).

Order History					
ID	DATE	TOTAL	PAID	DELIVERED	ACTIONS
6006c4c3f3b38030f435fa62	2021-01-19	2647.30	No	No	<button>Details</button>
6006d0466f6f772de41d0c3b	2021-01-19	757.85	No	No	<button>Details</button>

### 3.5.4 Order List

Die Order List können nur Sellers ansehen, die das Produkt verkaufen.

#### Orders

ID	USER	DATE	TOTAL	PAID	DELIVERED	ACTIONS
6006c4c3f3b38030f435fa62	Ismail	2021-01-19	2647.30	No	No	<a href="#">Details</a> <a href="#">Delete</a>
6006d0466f6f772de41d0c3b	Ismail	2021-01-19	757.85	No	No	<a href="#">Details</a> <a href="#">Delete</a>
600719fd17a8d90ed4c434fa	Abraham	2021-01-19	1323.65	No	No	<a href="#">Details</a> <a href="#">Delete</a>
60071a2617a8d90ed4c434fc	Luciano	2021-01-19	1378.85	No	No	<a href="#">Details</a> <a href="#">Delete</a>

### 3.5.5 Payment Method

#### Shipping

**Name:** Ismail Solombrino  
**Address:** Hofwiesenweg 2, Winterthur, Zürich, 8405, Schweiz

Not Delivered

#### Payment

**Method:** PayPal

Not Paid

#### Order Items

	Samsung Galaxy Note 20 Ultra Snapdragon	1 x \$1199 = \$1199
--	---	---------------------

#### Order Summary

Items	\$1199.00
Shipping	\$0.00
Tax	\$179.85
<b>Order Total</b>	<b>\$1378.85</b>

PayPal

Debit or Credit Card

Powered by **PayPal**

### 3.5.6 Place Order

#### Payment Method

☒ PayPal

☐ Stripe

Continue

### 3.5.7 Product Edit

#### Edit Product 60069e8deb6b700e10326c80

Name

Samsung Galaxy Tab S7+

Price

869

Image

/images/tablets/samsung/TabS7.webp

Image File

Keine ausgewählt

Category

Tablets

Brand

Samsung

Count In Stock

12

Description

Der neueste und schnellste Prozessor bietet höchste Produktivität und beste Performance beim Gamen. Das Tab S7+ ist auch mit 5G erhältlich - für eine superschnelle Datenübertragung und die niedrigste Latenz. Mit mehr als 10'000 mAh Kapazität hält der Akku auch langen Arbeitstagen stand und du

### 3.5.8 Product List

Products

Create Product

ID	NAME	PRICE	CATEGORY	BRAND	ACTIONS
60069e8deb6b700e10326c80	Samsung Galaxy Tab S7+	869	Tablets	Samsung	<button>Edit</button> <button>Delete</button>
60069e8deb6b700e10326c7f	Apple iPad Pro (2020)	1151	Tablets	Apple	<button>Edit</button> <button>Delete</button>
60069e8deb6b700e10326c7e	Apple iPhone 11	659	Phones	Apple	<button>Edit</button> <button>Delete</button>

1 2

### 3.5.9 Product

[Back to result](#)



#### Samsung Galaxy Note 20 Ultra Snapdragon

★★★★★ 200 reviews

Price : \$1199

Description:

Das Galaxy Note20 Ultra 5G verfügt über ein erstaunliches 6,9"-Edge-Display, das mit einer 120 Hz dynamischen Bildwiederholrate für das flüssigste Bilderlebnis sorgt. Das Galaxy Note20 5G verfügt über ein immersives, flaches 6,7"-Display. Das ikonische Design und die neuen Mystic-Farben mit Matt-Finish fallen überall auf.

★★★★★ 112 reviews

Price

\$1199

Status

In Stock

Qty

1

Add to Cart

Reviews

```
<button
  |   onClick={addToCartHandler}
  |   className="primary block"
  |
  |   Add to Cart
  |
  >
  </button>
```



### 3.5.10 Profile

Sobald man auf User Profile klickt wird folgendes angezeigt:

Im User Profile kann man alle Daten modifizieren und dann mittels update ins Backend posten/sendern.

Falls man ein Seller ist kann man sein Seller Logo, Beschreibung und sein Name anpassen.

#### User Profile

Name

Email

Password

confirm Password

#### Seller

Seller Name

Seller Logo

Seller Description

### 3.5.11 Register

#### Create Account

Name

Email address


Password

Confirm Password

Register


Already have an account? [Sign-In](#)

### 3.5.12 Search



- samsung
- bootstrap
- g
- Login
- turtleneck
- Röhrichtzone

### 3.5.13 Seller



**The Best Seller ever**

★★★★★ 112 reviews

[Contact Seller](#)

best seller

### 3.5.14 Shipping Address

#### Shipping Address

Full Name

Address

City

Postal Code

Country

Location

### 3.5.15 Sign in

#### Sign In

Email address

Password

### 3.5.16 User Edit

#### Edit User Abraham

Name

Abraham

Email

abraham@example.com

Is Seller



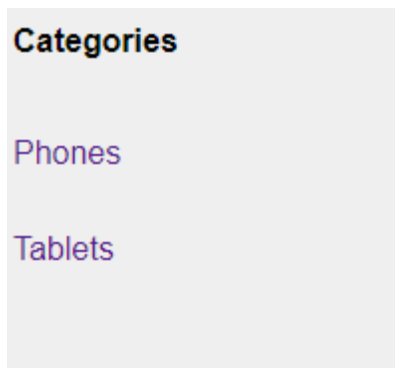
Is Admin



Update

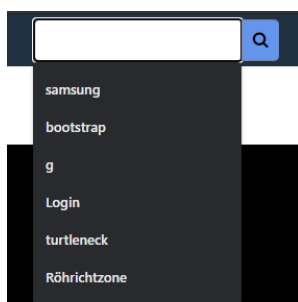
### 3.5.17 User List

Mit Hilfe von dem Menü Button links vom «The Brand Shop», gelingt man in die Kategorien.



```
<button
  type="button"
  className="open-sidebar"
  onClick={() => setSidebarIsOpen(true)}
>
```

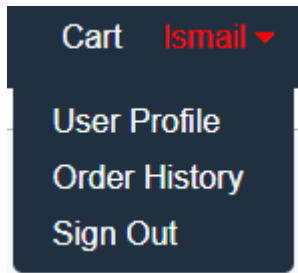
Dank der Search Funktion kann man im Shop nach dem gewünschten Produkt suchen.



Wenn man auf dem «Cart» Menü klickt kommt man in den Warenkorb.



Sobald man auf den UserProfile «hovert» mit der Maus drüber geht, erscheint ein Dropdown Menü, was folgendes anzeigt:



### 3.6 Store

In die Datei Store.js werden alle Reducers importiert und in die entsprechende Funktion implementiert

### 3.7 Utils

Hier werden alle maximale sowie minimale Werte bestimmt. Sowie alle Ratings eingestellt.

## 4 Backend

### 4.1 Models

#### 4.1.1 Order

Hier werden die Parameter für die Objekte definiert. Dies brauchen wir, um die Daten auf MongoDB zu implementieren

#### 4.1.2 Product

Hier werden, wie auf die Order File die Reviews sowie die Produktschemen definiert. Wiederum damit wir diese mit der Datenbank verbinden können.

#### 4.1.3 User

Hier werden die Eigenschaften von Users definiert, damit es wiederum keine Probleme mit der Datenbank entstehen.

### 4.2 Routers

#### 4.2.1 Order

In die File OrderRouter.js werden alle get und post Methoden implementiert, welche gebraucht werden um die Orders zu benutzen. Falls eine Order bezahlt wurde haben wir eine put Funktion, welche die Order auf bezahlt setzt. Sowie eine delete Funktion, wo die Order gelöscht wird.

#### 4.2.2 Product

Hier werden alle get und post Methoden erzeugt und implementiert, welche für die Produkte nötig sind.

#### 4.2.3 Upload

Durch die Methode uploadRouter.post() werden hier alle post Methoden hochgeladen.

#### 4.2.4 User

Ähnlich wie bei der Product Datei, werden hier alle get und post Methoden implementiert, welche gebraucht werden, um die Users zu benutzen. Man kann auch ein User löschen sowie die entsprechenden Daten bearbeiten.

#### 4.3 Server

Durch die Funktionen auf dieser Datei werden Verbindungen erstellt. Mithilfe einer API Key von Google verbinden wir die Webseite mit der Datenbank, sodass wir diese auch im Internet auf MongoDB benutzen können. Um PayPal richtig zu implementieren, mussten wir auch eine Verbindung zu PayPal erstellen.

#### 4.4 Utils

Hier wird überprüft welche Rechte der eingeloggte User hat.

## 5 Schlusswort