BITZ - Next Generation

Bitz ist eine innovative Online-Plattform, die sich auf den Kauf, Verkauf und Tausch von Technikprodukten spezialisiert hat. Unser Ziel ist es, Menschen zusammenzubringen, die ihre gebrauchte Technik verkaufen möchten, mit denen, die auf der Suche nach erschwinglichen und hochwertigen Geräten sind. Bitz bietet eine benutzerfreundliche und sichere Umgebung, in der Transaktionen einfach und unkompliziert abgewickelt werden können. Durch unseren Fokus auf Technologie und unsere engagierte Community unterscheiden wir uns von anderen Online-Marktplätzen. Bei Bitz finden Sie eine große Auswahl an Technikprodukten zu wettbewerbsfähigen Preisen und profitieren von einem erstklassigen Kundenerlebnis.

Inhaltsverzeichnis

20. Tailwind CSS (für Styling in der TSX-Syntax)
21. Tailwind Merge
22. Tailwind CSS Animate
23. @geoapify/react-geocoder-autocomplete
24. @geoapify/geocoder-autocomplete
25. React Intersection Observer (Lazy Loading)
26. Canvas Confetti
27. @types/canvas-confetti
28. Sonner
29. Sort-By-Typescript
30. Axios
31. ShadCN (für bereits gut aussehende Komponenten)

III. Backend-Technologien

1. Drizzle (Object Relational Mapper)
2. Neon Database (serverless PostgreSQL-Datenbank)
3. Next Auth.js (Authentifizierung)
4. @auth/drizzle-adapter
5. AWS SDK
6. Twilio
7. @types/twilio
8. Vaul
9. Pusher (
10. Pusher JS
11. Stripe
12. @simplewebauthn/browser
13. @simplewebauthn/server

IV. Weitere Tools

1. Node.js
2. npm
3. Git und Github (Versionskontrolle)
4. Vercel (Deployment)
5. Discord (Kommunikation)
6. Notion (Organisation)
7. Figma (für Styling und Corporate Design)

I. Einleitung

BITZ - Next Generation Bitz ist eine innovative Online-Plattform, die sich auf den Kauf, Verkauf und Tausch von Technikprodukten spezialisiert hat. Unser Ziel ist es, Menschen zusammenzubringen, die ihre gebrauchte Technik verkaufen möchten, mit denen, die auf der Suche nach erschwinglichen und hochwertigen Geräten sind. Bitz bietet eine benutzerfreundliche und sichere Umgebung, in der Transaktionen einfach und unkompliziert abgewickelt werden können. Durch unseren Fokus auf Technologie und unsere engagierte Community unterscheiden wir uns von anderen Online-Marktplätzen. Bei Bitz finden Sie eine große Auswahl an Technikprodukten zu wettbewerbsfähigen Preisen und profitieren von einem erstklassigen Kundenerlebnis.

Dieses Dokument bietet eine detaillierte Dokumentation der in unserem Projekt "Bitz" verwendeten Technologien. Es konzentriert sich speziell darauf, welchen Mehrwert die einzelnen Technologien für unser Projekt bieten und wie sie zu einer besseren Benutzererfahrung, verbesserter Codequalität und effizienter Entwicklung beitragen.

II. Frontend-Technologien

2.1 TypeScript

TypeScript ermöglicht es uns, unseren Frontend-Code typsicher zu gestalten. Dies führt zu einer höheren Codequalität, da Fehler frühzeitig erkannt und behoben werden können.

```
// Definition einer Produkt-Schnittstelle
interface Product {
  id: string;
  name: string;
  price: number;
}

// Verwendung der Schnittstelle in einer Funktion
function displayProduct(product: Product) {
  console.log(`Produktname: ${product.name}`);
  console.log(`Preis: ${product.price}`);
}
```

Durch die Verwendung von TypeScript können wir sicherstellen, dass die Funktion displayProduct immer ein Objekt vom Typ Product erhält. Dies verhindert Laufzeitfehler, die auftreten könnten, wenn die Funktion mit falschen Datentypen aufgerufen wird.

2.2 Next.js

Next.js ist ein Framework für serverseitiges Rendering und statische Seitengenerierung in React-Anwendungen. Es bietet uns Funktionen wie serverseitiges Rendering, Routing, Datenabruf und Bildoptimierung, die die Leistung und SEO unserer Anwendung verbessern.

```
// pages/products/[id].js
import { getProductById } from '../api/products';

export async function getServerSideProps({ params }) {
  const product = await getProductById(params.id);
  return { props: { product } };
}

function ProductPage({ product }) {
  return (
    <div>
      <h1>{product.name}</h1>
      <p>{product.description}</p>
    </div>
  );
}

export default ProductPage;
```

In diesem Beispiel verwenden wir getServerSideProps, um Produktdaten serverseitig abzurufen und an die Komponente ProductPage zu übergeben. Dies verbessert die SEO, da Suchmaschinen den vollständigen HTML-Code der Seite crawlen können.

2.3 React

React ist eine JavaScript-Bibliothek zum Erstellen von Benutzeroberflächen. Es ermöglicht uns, unsere Benutzeroberfläche in wiederverwendbare Komponenten zu zerlegen, was die Entwicklung und Wartung vereinfacht.

```
// components/ProductCard.js
function ProductCard({ product }) {
  return (
    <div className="product-card">
      <img src={product.imageUrl} alt={product.name} />
      <h3>{product.name}</h3>
      <p>{product.price}</p>
    </div>
  );
}

export default ProductCard;
```

In diesem Beispiel definieren wir eine wiederverwendbare Komponente ProductCard, die die Details eines Produkts anzeigt. Diese Komponente kann dann an verschiedenen Stellen in der Anwendung wiederverwendet werden.

2.4 React-Dom

React-Dom ist ein Paket, das die Interaktion zwischen React und dem Document Object Model (DOM) ermöglicht. Es ist für die Darstellung von React-Komponenten im Browser unerlässlich.

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

In diesem Beispiel verwenden wir ReactDOM.createRoot, um eine Root-Instanz zu erstellen und die Komponente App im DOM-Element mit der ID "root" zu rendern.

2.5 Next-Intl

Next-Intl ist eine Bibliothek für die Internationalisierung (i18n) in Next.js-Anwendungen. Sie ermöglicht es uns, unsere Anwendung in mehrere Sprachen zu übersetzen und an verschiedene Regionen anzupassen.

```
// pages/index.js
import { useTranslations } from 'next-intl';

function HomePage() {
  const t = useTranslations('HomePage');
  return (
    <div>
      <h1>{t('welcome')}</h1>
      <p>{t('description')}</p>
    </div>
  );
}

export default HomePage;
```

In diesem Beispiel verwenden wir den Hook useTranslations, um auf Übersetzungen für die Komponente HomePage zuzugreifen. Die Funktion t gibt die Übersetzung für den angegebenen Schlüssel zurück.

2.6 React-Hook-Form

React-Hook-Form ist eine Bibliothek zur Formularverarbeitung in React. Sie vereinfacht die Erstellung und Validierung von Formularen und reduziert die Menge an Boilerplate-Code.

```
// components/LoginForm.js
import { useForm } from 'react-hook-form';

function LoginForm() {
  const { register, handleSubmit } = useForm();

  const onSubmit = (data) => console.log(data);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input type="email" {...register('email')} placeholder="E-Mail" />
      <input type="password" {...register('password')} placeholder="Passwort" />
      <button type="submit">Anmelden</button>
    </form>
  );
}

export default LoginForm;
```

In diesem Beispiel verwenden wir useForm, um eine Formularinstanz zu erstellen. Die Funktion register wird verwendet, um Formularfelder zu registrieren, und handleSubmit verarbeitet die Formularübermittlung.

2.7 Zod

Zod ist eine Bibliothek zur Schemavalidierung in TypeScript. Sie ermöglicht es uns, die Struktur und den Datentyp von Daten zu definieren und zu validieren.

```
// schemas/product.ts
import { z } from 'zod';

const productSchema = z.object({
  id: z.string().uuid(),
  name: z.string().min(3),
  price: z.number().positive(),
});

export default productSchema;
```

In diesem Beispiel definieren wir ein Schema für ein Produkt mit Zod. Das Schema gibt an, dass die Eigenschaft id ein UUID-String, name ein String mit mindestens 3 Zeichen und price eine positive Zahl

sein muss.

2.8 Class Variance Authority

Class Variance Authority (CVA) ist eine Bibliothek, die uns hilft, dynamische CSS-Klassen basierend auf dem Zustand von Komponenten zu generieren. Dies ermöglicht es uns, unsere Benutzeroberfläche flexibler und interaktiver zu gestalten.

```
// components/Button.js
import { cva } from 'class-variance-authority';

const buttonStyles = cva(
  'px-4 py-2 rounded-md font-medium',
  {
    variants: {
      intent: {
        primary: 'bg-blue-500 text-white',
        secondary: 'bg-gray-200 text-gray-800',
      },
      size: {
        small: 'text-sm',
        large: 'text-lg',
      },
    },
  }
);

function Button({ intent = 'primary', size = 'small', children }) {
  return (
    <button className={buttonStyles({ intent, size })}>
      {children}
    </button>
  );
}

export default Button;
```

In diesem Beispiel verwenden wir cva, um eine Funktion buttonStyles zu erstellen, die dynamische CSS-Klassen basierend auf den Props intent und size generiert.

2.9 clsx

clsx ist eine Hilfsfunktion zum Kombinieren von CSS-Klassen in React. Sie vereinfacht die bedingte Anwendung von CSS-Klassen und verbessert die Lesbarkeit des Codes.

```
// components/ProductCard.js
import clsx from 'clsx';

function ProductCard({ product, isSelected }) {
  return (
    <div className={clsx('product-card', isSelected && 'selected')}>
      {/* ... */}
    </div>
  );
}

export default ProductCard;
```

In diesem Beispiel verwenden wir clsx, um die CSS-Klasse "selected" bedingt anzuwenden, wenn die Prop isSelected wahr ist.

2.10 Radix UI

Radix UI ist eine Sammlung von unaufdringlichen UI-Komponenten für React. Sie bieten uns eine solide Grundlage für die Erstellung zugänglicher und benutzerfreundlicher Benutzeroberflächen.

Beispiel:

```
// components/DropdownMenu.js
import {
  DropdownMenu,
  DropdownMenuTrigger,
  DropdownMenuContent,
  DropdownMenuItem,
} from '@radix-ui/react-dropdown-menu';

function MyDropdownMenu() {
  return (
    <DropdownMenu>
      <DropdownMenuTrigger>Menü öffnen</DropdownMenuTrigger>
      <DropdownMenuContent>
        <DropdownMenuItem>Option 1</DropdownMenuItem>
        <DropdownMenuItem>Option 2</DropdownMenuItem>
      </DropdownMenuContent>
    </DropdownMenu>
  );
}

export default MyDropdownMenu;
```

In diesem Beispiel verwenden wir Komponenten aus Radix UI, um ein Dropdown-Menü zu erstellen. Die Komponenten sind bereits auf Barrierefreiheit und Benutzerfreundlichkeit ausgelegt.

2.11 Framer Motion

Framer Motion ist eine Animationsbibliothek für React. Sie ermöglicht es uns, flüssige und ansprechende Animationen und Übergänge in unserer Benutzeroberfläche zu erstellen.

Beispiel:

```
// components/Modal.js
import { motion } from 'framer-motion';

const modalVariants = {
  hidden: { opacity: 0, scale: 0.9 },
  visible: { opacity: 1, scale: 1 },
};

function Modal({ isOpen, onClose, children }) {
  return (
    <motion.div
      initial="hidden"
      animate={isOpen ? 'visible' : 'hidden'}
      variants={modalVariants}
    >
      {/* ... */}
    </motion.div>
  );
}

export default Modal;
```

In diesem Beispiel verwenden wir motion.div, um eine animierte Modal-Komponente zu erstellen. Die Animation wird durch die variants-Eigenschaft definiert und durch den Zustand der Prop isOpen gesteuert.

2.12 Embla Carousel React

Embla Carousel React ist eine Bibliothek zum Erstellen von Karussells in React. Sie bietet uns eine flexible und einfach zu bedienende Möglichkeit, Bilder und andere Inhalte in einem Karussell anzuzeigen.

Beispiel:

```
// components/ProductCarousel.js
import { Embla, useEmblaCarousel } from 'embla-carousel-react';

function ProductCarousel({ products }) {
  const [emblaRef] = useEmblaCarousel();
```

```
  return (
    <Embla ref={emblaRef}>
      <div className="embla__viewport">
        <div className="embla__container">
          {products.map((product) => (
            <div key={product.id} className="embla__slide">
              <img src={product.imageUrl} alt={product.name} />
            </div>
          ))}
        </div>
      </div>
    </Embla>
  );
}

export default ProductCarousel;
```

In diesem Beispiel verwenden wir Embla und useEmblaCarousel, um ein Karussell zu erstellen, das Produktabbildungen anzeigt.

2.13 React Three Fiber

React Three Fiber ist eine Bibliothek zum Rendern von 3D-Grafiken in React mit Three.js. Sie ermöglicht es uns, interaktive 3D-Erlebnisse in unserer Anwendung zu erstellen.

Beispiel:

```
// components/Product3DModel.js
import { Canvas } from '@react-three/fiber';

function Product3DModel({ modelUrl }) {
  return (
    <Canvas>
      <mesh>
        <primitive object={modelUrl} />
      </mesh>
    </Canvas>
  );
}

export default Product3DModel;
```

In diesem Beispiel verwenden wir Canvas aus React Three Fiber, um ein 3D-Modell eines Produkts zu rendern.

2.14 Three

Three.js ist eine JavaScript-Bibliothek zum Erstellen und Anzeigen von 3D-Grafiken im Browser. Sie bietet uns eine leistungsstarke API für die Arbeit mit 3D-Objekten, -Materialien, -Lichtern und -Kameras.

Beispiel:

```
// components/Product3DModel.js
import * as THREE from 'three';

function Product3DModel({ modelUrl }) {
  const loader = new THREE.ObjectLoader();
  loader.load(modelUrl, (object) => {
    // Objekt zur Szene hinzufügen
  });

  return (
    <div>
      {/* ... */}
    </div>
  );
}

export default Product3DModel;
```

In diesem Beispiel verwenden wir THREE.ObjectLoader, um ein 3D-Modell aus einer Datei zu laden.

2.15 @types/three

@types/three enthält TypeScript-Typdefinitionen für Three.js. Dies ermöglicht es uns, Three.js in unserem TypeScript-Code typsicher zu verwenden.

Beispiel:

```
// components/Product3DModel.ts
import * as THREE from 'three';

function Product3DModel({ modelUrl }: { modelUrl: string }) {
  const loader = new THREE.ObjectLoader();
  loader.load(modelUrl, (object: THREE.Object3D) => {
    // Objekt zur Szene hinzufügen
  });

  return (
    <div>
      {/* ... */}
    </div>
```

```
    );
  }
  export default Product3DModel;
```

In diesem Beispiel verwenden wir die Typdefinition THREE.Object3D, um den Typ des geladenen 3D-Objekts anzugeben.

2.16 Drei

Drei ist eine Sammlung von Hilfsfunktionen und -komponenten für React Three Fiber. Sie vereinfacht die Verwendung von Three.js in React und bietet zusätzliche Funktionen.

```
// components/Product3DModel.js
import { Canvas } from '@react-three/fiber';
import { OrbitControls } from '@react-three/drei';

function Product3DModel({ modelUrl }) {
  return (
    <Canvas>
      <OrbitControls />
      <mesh>
        <primitive object={modelUrl} />
      </mesh>
    </Canvas>
  );
}

export default Product3DModel;
```

In diesem Beispiel verwenden wir die Komponente OrbitControls aus Drei, um eine Kamerabewegungssteuerung zum 3D-Modell hinzuzufügen.

2.17 React Icons

React Icons ist eine Sammlung von SVG-Icons, die als React-Komponenten verwendet werden können. Sie bietet uns eine große Auswahl an Icons für verschiedene Anwendungsfälle.

```
// components/IconButton.js
import { FaSearch } from 'react-icons/fa';

function IconButton({ icon, onClick }) {
  return (
    <button onClick={onClick}>
      <FaSearch />
    </button>
```

```
    );
  }

  export default IconButton;
```

In diesem Beispiel verwenden wir das Icon FaSearch aus React Icons, um einen Button mit einem Such-Icon zu erstellen.

2.18 Lucide React

Lucide React ist eine weitere Sammlung von SVG-Icons, die für ihre minimalistische Ästhetik und gute Barrierefreiheit bekannt ist.

```
// components/Navigation.js
import { Home } from 'lucide-react';

function Navigation() {
  return (
    <nav>
      <ul>
        <li>
          <a href="/">
            <Home /> Startseite
          </a>
        </li>
        {/* ... */}
      </ul>
    </nav>
  );
}

export default Navigation;
```

In diesem Beispiel verwenden wir das Icon Home aus Lucide React, um einen Link zur Startseite in unserer Navigation zu erstellen.

2.19 Zustand

Zustand ist eine Bibliothek zur Zustandsverwaltung in React. Sie ermöglicht es uns, den Zustand unserer Anwendung auf einfache und effiziente Weise zu verwalten und zwischen Komponenten zu teilen.

```
// components/ShoppingCart.js
import { create } from 'zustand';
```

```
const useShoppingCart = create((set) => ({
  items: [],
  addItem: (item) => set((state) => ({ items: [...state.items, item] })),
  removeItem: (itemId) =>
    set((state) => ({ items: state.items.filter((item) => item.id !== itemId) })),
}));

function ShoppingCart() {
  const { items, addItem, removeItem } = useShoppingCart();

  return (
    <div>
      {/* ... */}
    </div>
  );
}

export default ShoppingCart;
```

In diesem Beispiel verwenden wir zustand, um einen globalen Warenkorb-Zustand zu erstellen. Die Funktionen addItem und removeItem können verwendet werden, um Artikel zum Warenkorb hinzuzufügen oder daraus zu entfernen.

2.20 Tailwind CSS

Tailwind CSS ist ein Utility-First-CSS-Framework. Es bietet uns eine große Sammlung von vorgefertigten CSS-Klassen, mit denen wir unsere Benutzeroberfläche schnell und einfach gestalten können.

```
<button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
  Button
</button>
```

In diesem Beispiel verwenden wir Tailwind CSS-Klassen, um einen blauen Button mit Hover-Effekt zu erstellen.

2.21 Tailwind Merge

Tailwind Merge ist ein Tool, das uns hilft, Tailwind CSS-Klassen in unserem JavaScript-Code zu kombinieren und zu optimieren.

```
// components/Button.js
import { twMerge } from 'tailwind-merge';
```

```
function Button({ variant = 'primary', className, children }) {
  const buttonClasses = twMerge(
    'px-4 py-2 rounded-md font-medium',
    variant === 'primary' && 'bg-blue-500 text-white',
    variant === 'secondary' && 'bg-gray-200 text-gray-800',
    className
  );

  return <button className={buttonClasses}>{children}</button>;
}

export default Button;
```

In diesem Beispiel verwenden wir twMerge, um die Basis-Button-Klassen mit den variantenspezifischen Klassen und den benutzerdefinierten Klassen aus der Prop className zu kombinieren.

2.22 Tailwind CSS Animate

Tailwind CSS Animate ist eine Erweiterung für Tailwind CSS, die uns vorgefertigte CSS-Animationen bietet.

```
<div class="animate-spin">
  {/* ... */}
</div>
```

In diesem Beispiel verwenden wir die Klasse animate-spin aus Tailwind CSS Animate, um ein Element zu drehen.

2.23 @geoapify/react-geocoder-autocomplete

@geoapify/react-geocoder-autocomplete ist eine React-Komponente für die automatische Vervollständigung von Adressen und Orten.

```
// components/AddressInput.js
import { GeocoderAutocomplete } from '@geoapify/react-geocoder-autocomplete';

function AddressInput({ onSelect }) {
  return (
    <GeocoderAutocomplete
      apiKey="YOUR_API_KEY"
      onSelect={onSelect}
      placeholder="Adresse eingeben"
    />
  );
```

```
}

export default AddressInput;
```

In diesem Beispiel verwenden wir GeocoderAutocomplete, um ein Eingabefeld mit automatischer Adressvervollständigung zu erstellen.

2.24 @geoapify/geocoder-autocomplete

@geoapify/geocoder-autocomplete ist die zugrunde liegende Bibliothek für die Adressvervollständigung, die von @geoapify/react-geocoder-autocomplete verwendet wird.

2.25 React Intersection Observer (Lazy Loading)

React Intersection Observer ist eine React-Implementierung der Intersection Observer API. Sie ermöglicht es uns, zu erkennen, wann ein Element im Ansichtsbereich des Benutzers sichtbar ist.

```
// components/LazyLoadedImage.js
import { useInView } from 'react-intersection-observer';

function LazyLoadedImage({ src, alt }) {
  const { ref, inView } = useInView();

  return (
    <img
      ref={ref}
      src={inView ? src : ''}
      alt={alt}
    />
  );
}

export default LazyLoadedImage;
```

In diesem Beispiel verwenden wir useInView, um zu erkennen, wann das Bild im Ansichtsbereich des Benutzers sichtbar ist. Nur dann wird das Bild geladen.

Die Webseite wird durch LazyLoading performanter und verursacht weniger Traffic.

2.26 Canvas Confetti

Canvas Confetti ist eine Bibliothek zum Anzeigen von Konfetti-Animationen im Browser.

```
// components/ConfettiExplosion.js
import confetti from 'canvas-confetti';

function ConfettiExplosion() {
  confetti();
  return null;
}

export default ConfettiExplosion;
```

In diesem Beispiel verwenden wir confetti(), um eine Konfetti-Explosion auszulösen.

2.27 @types/canvas-confetti

@types/canvas-confetti enthält TypeScript-Typdefinitionen für Canvas Confetti.

2.28 Sonner

Sonner ist eine Bibliothek zum Anzeigen von Benachrichtigungen in React.

```
// components/NotificationProvider.js
import { SonnerProvider, useSonner } from 'sonner';

function NotificationProvider({ children }) {
  return <SonnerProvider>{children}</SonnerProvider>;
}

function MyComponent() {
  const { notify } = useSonner();

  const handleClick = () => {
    notify('Erfolgreich gespeichert!');
  };

  return (
    <button onClick={handleClick}>
      Speichern
    </button>
  );
}
```

In diesem Beispiel verwenden wir SonnerProvider und useSonner, um Benachrichtigungen anzuzeigen.

2.29 Sort-By-Typescript

Sort-By-Typescript ist eine Bibliothek zum Sortieren von Arrays in TypeScript.

```
// utils/sorting.ts
import sortBy from 'sort-by-typescript';

const products = [
  { name: 'Produkt A', price: 10 },
  { name: 'Produkt B', price: 20 },
  { name: 'Produkt C', price: 5 },
];

const sortedProducts = sortBy(products, 'price');
```

In diesem Beispiel verwenden wir sortBy, um das Array products nach dem Preis zu sortieren.

2.30 Axios

Axios ist eine Bibliothek zum Erstellen von HTTP-Anfragen in JavaScript.

```
// api/products.js
import axios from 'axios';

export const getProducts = async () => {
  const response = await axios.get('/api/products');
  return response.data;
};
```

In diesem Beispiel verwenden wir axios.get, um eine GET-Anfrage an die API-Route /api/products zu senden.

2.31 ShadCN

ShadCN ist eine Sammlung von vorgestalteten UI-Komponenten, die in unserem Projekt verwendet werden, um schnell ansprechende Benutzeroberflächen zu erstellen. Diese Komponenten sind sorgfältig gestaltet und bieten ein professionelles und modernes Erscheinungsbild. Der Hauptvorteil der Verwendung von ShadCN-Komponenten ist die Zeitersparnis bei der Entwicklung, da wir nicht jede Komponente von Grund auf neu gestalten müssen. Stattdessen können wir die vorhandenen Komponenten von ShadCN nutzen und sie nahtlos in unsere Anwendung integrieren.

Beispiel für die Verwendung von ShadCN Ein Beispiel für den Einsatz von ShadCN in unserem Projekt ist die Implementierung von Dialog- und Sheet-Komponenten. Diese Komponenten werden verwendet, um modale Dialoge und seitliche Überlagerungen in der Benutzeroberfläche zu erstellen.

```
import { Dialog, DialogTrigger, DialogContent } from '@shadcn/ui';

interface DialogContentProps extends React.ComponentPropsWithoutRef<typeof DialogContent> {
  closeBtn?: boolean;
}

const CustomDialog: React.FC<DialogContentProps> = ({ closeBtn, ...props }) => (
  <Dialog>
    <DialogTrigger>Open Dialog</DialogTrigger>
    <DialogContent {...props}>
      {closeBtn && <button onClick={() => console.log('Close Dialog')}>Close</button>}
      {props.children}
    </DialogContent>
  </Dialog>
);

export default CustomDialog;
```

In diesem Beispiel verwenden wir die Dialog, DialogTrigger und DialogContent Komponenten von ShadCN, um einen modalen Dialog zu erstellen. Die DialogContent Komponente wird erweitert, um eine optionale Schaltfläche zum Schließen des Dialogs hinzuzufügen. Durch die Verwendung von ShadCN-Komponenten können wir schnell und effizient ansprechende und konsistente UI-Elemente erstellen, ohne viel Zeit mit dem Design und der Implementierung von Grund auf zu verbringen.

III. Backend-Technologien

3.1 Drizzle ORM

Drizzle ORM ist ein TypeScript-first ORM, das eine typsichere Möglichkeit bietet, mit unserer Datenbank zu interagieren. Es vereinfacht Datenbankabfragen und -migrationen und verbessert die Lesbarkeit und Wartbarkeit unseres Backend-Codes.

```
// schema.ts
import { pgTable, text, integer } from 'drizzle-orm/pg-core';

export const products = pgTable('products', {
  id: text('id').primaryKey(),
  name: text('name').notNull(),
  price: integer('price'),
});

// routes/products.ts
import { db } from '../db';

export const getProducts = async () => {
```

```
    const allProducts = await db.select().from(products);
    return allProducts;
  };
```

In diesem Beispiel definieren wir eine Tabelle products mit Drizzle ORM und verwenden sie, um alle Produkte aus der Datenbank abzufragen.

3.2 Neon Database

Neon ist ein Serverless PostgreSQL-Datenbankdienst. Er bietet uns eine skalierbare, zuverlässige und kostengünstige Datenbanklösung für unsere Anwendung.

3.3 Next Auth.js

NextAuth.js ist eine Authentifizierungsbibliothek für Next.js, die verschiedene Authentifizierungsanbieter wie Google, Facebook, Twitter usw. unterstützt. Sie vereinfacht die Implementierung der Benutzerauthentifizierung und -autorisierung in unserer Anwendung.

```
// pages/api/auth/[...nextauth].js
import NextAuth from 'next-auth';
import GoogleProvider from 'next-auth/providers/google';

export default NextAuth({
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    }),
  ],
});
```

In diesem Beispiel konfigurieren wir NextAuth.js für die Verwendung von Google als Authentifizierungsanbieter.

3.4 @auth/drizzle-adapter

@auth/drizzle-adapter ist ein Adapter, der die Integration von NextAuth.js mit Drizzle ORM ermöglicht. Er ermöglicht es uns, Benutzerdaten und Sitzungsdaten in unserer Neon-Datenbank zu speichern.

3.5 AWS SDK

AWS SDK (Software Development Kit) ermöglicht es uns, auf verschiedene AWS-Dienste wie Amazon S3 (Simple Storage Service) zuzugreifen. Wir verwenden S3 zum Speichern von Benutzeruploads wie Produktbildern.

```
// lib/s3.js
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

const s3Client = new S3Client({
  region: process.env.AWS_REGION,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  },
});

export const uploadImage = async (file) => {
  const params = {
    Bucket: process.env.AWS_S3_BUCKET_NAME,
    Key: file.name,
    Body: file,
  };

  const command = new PutObjectCommand(params);
  await s3Client.send(command);
};
```

In diesem Beispiel verwenden wir das AWS SDK, um ein Bild in unseren S3-Bucket hochzuladen.

3.6 Twilio

Twilio ist eine Cloud-Kommunikationsplattform, die es uns ermöglicht, SMS-Nachrichten zu senden. Wir verwenden Twilio, um Benutzern Bestätigungscodes für die Telefonnummernüberprüfung zu senden.

```
// lib/twilio.js
const twilio = require('twilio');

const client = twilio(
  process.env.TWILIO_ACCOUNT_SID,
  process.env.TWILIO_AUTH_TOKEN
);

export const sendVerificationCode = async (phoneNumber, code) => {
  await client.messages.create({
    body: `Ihr Bestätigungscode lautet: ${code}`,
    from: process.env.TWILIO_PHONE_NUMBER,
    to: phoneNumber,
```

```
    });
};
```

In diesem Beispiel verwenden wir das Twilio SDK, um einen Bestätigungscode per SMS zu senden.

3.7 @types/twilio

@types/twilio enthält TypeScript-Typdefinitionen für das Twilio SDK.

3.8 Vaul

Vaul ist eine Bibliothek zur Verwaltung von Umgebungsvariablen in Next.js. Sie ermöglicht es uns, sensible Daten wie API-Schlüssel und Datenbankverbindungszeichenfolgen sicher zu speichern und zu verwenden.

```
// lib/db.ts
import { neon, neonConfig } from '@neondatabase/serverless';

neonConfig.fetchConnectionCache = true;

if (!process.env.DATABASE_URL) {
  throw new Error('DATABASE_URL environment variable not set');
}

const sql = neon(process.env.DATABASE_URL);

export { sql };
```

In diesem Beispiel verwenden wir process.env.DATABASE_URL, um die Datenbankverbindungszeichenfolge aus den Umgebungsvariablen abzurufen.

3.9 Pusher

Pusher ist ein Dienst für Echtzeitkommunikation. Wir verwenden Pusher, um Benachrichtigungen und andere Echtzeit-Updates an Benutzer zu senden.

```
// lib/pusher.js
import Pusher from 'pusher-js';

const pusher = new Pusher(process.env.PUSHER_APP_KEY, {
  cluster: process.env.PUSHER_APP_CLUSTER,
});
```

```
export default pusher;
```

In diesem Beispiel erstellen wir eine Instanz des Pusher-Clients.

3.10 Pusher JS

Pusher JS ist die JavaScript-Clientbibliothek für Pusher.

3.11 Stripe

Stripe ist eine Plattform für Online-Zahlungen. Wir verwenden Stripe, um Zahlungen von Benutzern zu verarbeiten.

```
// pages/api/checkout_sessions.js
import { stripe } from '../../lib/stripe';

export default async function handler(req, res) {
  const { priceId } = req.body;

  const session = await stripe.checkout.sessions.create({
    mode: 'payment',
    payment_method_types: ['card'],
    line_items: [
      {
        price: priceId,
        quantity: 1,
      },
    ],
    success_url: `${req.headers.origin}/success`,
    cancel_url: `${req.headers.origin}/cancel`,
  });

  res.status(200).json({ sessionId: session.id });
}
```

In diesem Beispiel verwenden wir das Stripe SDK, um eine Checkout-Sitzung zu erstellen.

3.12 @simplewebauthn/browser

@simplewebauthn/browser wird in unserem Projekt für die Implementierung von WebAuthn verwendet. Es ermöglicht uns, sichere und benutzerfreundliche Authentifizierungsmethoden wie biometrische Authentifizierung und Sicherheitsschlüssel zu unterstützen. Ein Beispiel für den Einsatz

von @simplewebauthn/browser ist die Implementierung von WebAuthn-Authentifizierungslogik in unseren Frontend-Komponenten, um Benutzern eine sichere und bequeme Authentifizierung zu ermöglichen.

3.13 @simplewebauthn/server

@simplewebauthn/server wird in unserem Projekt verwendet, um die serverseitige Logik für WebAuthn zu implementieren. Es ermöglicht uns, WebAuthn-Authentifizierungsanforderungen sicher zu verarbeiten und zu überprüfen. Ein Beispiel für den Einsatz von @simplewebauthn/server ist die Implementierung von WebAuthn-Authentifizierungslogik in unseren Backend-Komponenten, um die Sicherheit und Integrität der Authentifizierungsprozesse zu gewährleisten.

IV. Weitere Tools

4.1 Node.js

Node.js ist eine JavaScript-Laufzeitumgebung, die es uns ermöglicht, JavaScript-Code außerhalb eines Webbrowsers auszuführen. Wir verwenden Node.js für unseren Backend-Server und für die Ausführung von Build-Tools.

4.2 npm

npm (Node Package Manager) ist der Paketmanager für Node.js. Wir verwenden npm, um Abhängigkeiten zu verwalten und unsere Anwendung zu erstellen.

4.3 Git und Github

Git ist ein Versionskontrollsystem, das es uns ermöglicht, Änderungen an unserem Code zu verfolgen und mit anderen Entwicklern zusammenzuarbeiten. Github ist eine webbasierte Hosting-Plattform für Git-Repositories. Wir verwenden GitFlow, um unsere Entwicklung zu organisieren. Wir erstellen und nutzen neue Branches für die meisten neuen Features oder Überarbeitungen. Die Branches werden gekennzeichnet mit fix-, feat-, refactor- oder chore-Präfixen.

4.4 Figma (Designprototypen)

Durch die Verwendung von Figma in diesem Projekt wird eine enge Zusammenarbeit zwischen Design und Entwicklung gefördert. Das visuelle Design der Anwendung kann effizient erstellt, getestet und iteriert werden. Die Erstellung von wiederverwendbaren Designkomponenten und Stilen gewährleistet eine konsistente visuelle Identität und erleichtert die Pflege des Designs. Insgesamt trägt Figma dazu

bei, eine ansprechende und benutzerfreundliche Oberfläche zu schaffen, die den Anforderungen und Erwartungen der Benutzer entspricht.