WISS Schulen für Wirtschaft
Informatik und Immobilien

Ungebremste E-Commerce-Power
Kein Standard von der Stange

WACHSTUM OHNE LIMIT DANK FLEXIBLER TECHNIK, DIE SICH DEINEN BEDÜRFNISSEN ANPASST

VON ROMAN FLÜKIGER

Inhaltsverzeichnis

Projektidee: Onlineshop mit Admin-Funktionen

Stell dir vor, du betreibst einen vielversprechenden Modehandel – aber deine Kundinnen und Kunden erleben lange Ladezeiten, ein austauschbares Design und einen umständlichen Checkout. Jeden Monat brechen 20 % deiner potenziellen Neukunden den Kauf ab. Zusätzlich bist du in einem starren Abo-Modell gefangen, das dir kaum Spielraum für eigene Ideen lässt. Genau hier setzt meine Lösung an.

Ich entwickle einen kompakten Onlineshop; blitzschnell, individuell und flexibel erweiterbar. Auf der Startseite können deine Besucher Produkte nach Kategorien filtern und direkt anzeigen lassen. Das moderne React-Frontend sorgt für eine reaktive Benutzerführung und kurze Ladezeiten.

Im Admin-Bereich kannst du mit validierten Formularen neue Kategorien und Produkte anlegen, bearbeiten oder löschen – inklusive smarter Eingaberegeln wie Preise in 5 Rappen-Schritten.

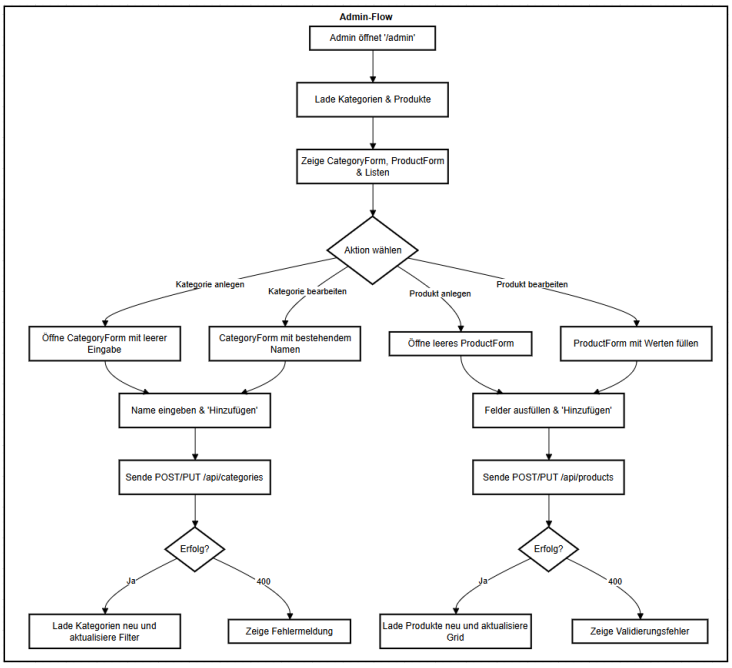
Im Backend sorgt Java Spring Boot in Kombination mit einer MySQL-Datenbank für eine robuste Datenhaltung und einer klar strukturierten REST-API. Das technische Gerüst ist bewusst so aufgebaut, dass sich später weitere Funktionen wie Benutzerverwaltung, Bestellungen oder Zahlungsintegration ergänzen lassen.

Mein Ziel ist es, einen performanten und zuverlässigen Prototypen zu bieten, der die Grundlage für einen skalierbaren Onlineshop darstellt.

Storyboard der Applikation

Beim ersten Aufruf der Startseite lädt die Anwendung automatisch alle Produkte und Kategorien sowie den Slider. Der Besucher sieht oben den Bild-Slider, darunter eine Filterleiste mit Kategorien und ein responsive Grid aus Produkt-Cards. Klickt er auf eine Kategorie, wird die Liste lokal gefiltert, sodass nur passende Produkte angezeigt werden.

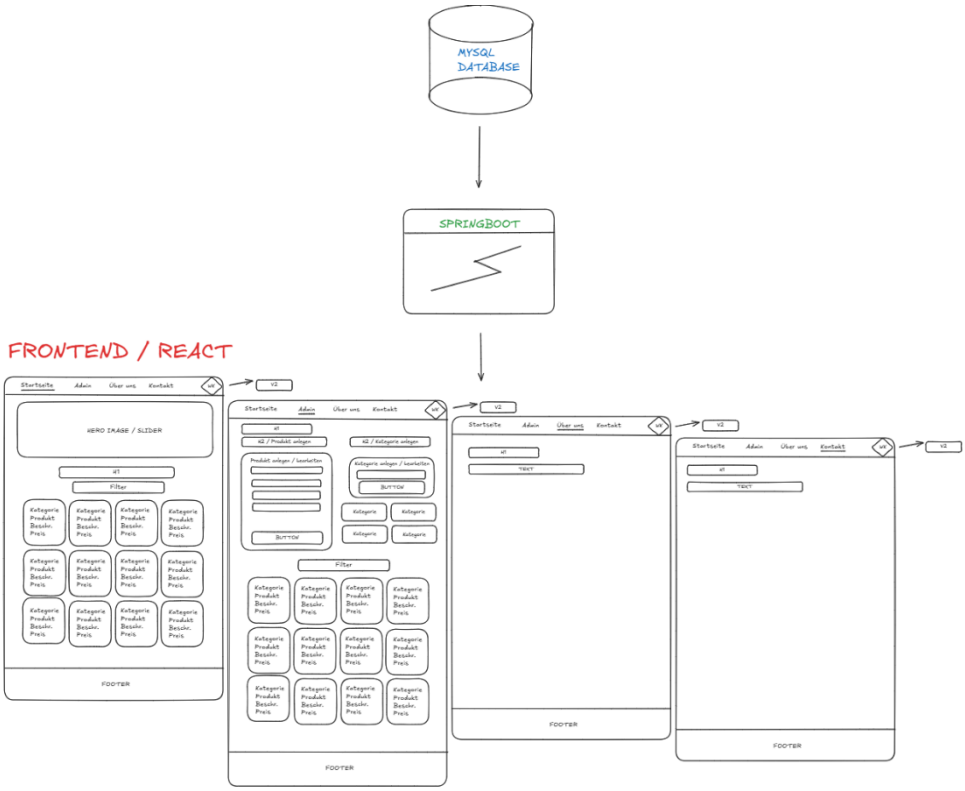
Im Admin-Bereich stehen zwei nebeneinander angeordnete Formulare. Eines zur Verwaltung der Kategorien und eines zur Verwaltung der Produkte. Zu Beginn lädt die Admin-Seite alle bestehenden Kategorien und Produkte. Möchte der Administrator eine neue Kategorie anlegen, öffnet er das CategoryForm, gibt einen eindeutigen Namen ein und speichert. Die Anwendung ruft POST auf, empfängt die neue Kategorie mit ID und aktualisiert danach automatisch den Filter. Analog dazu kann der Admin Produkte über das Produkt Formular anlegen oder bestehende bearbeiten. Nach dem Ausfüllen der Felder (Name, Beschreibung, Preis in 0.05-Schritten, Bild-URL, Kategorie) sendet das Frontend POST oder PUT und aktualisiert die Produktliste ohne manuellen Reload. Das Löschen von Kategorien ist nur möglich, wenn keine Produkte mehr darauf verweisen.

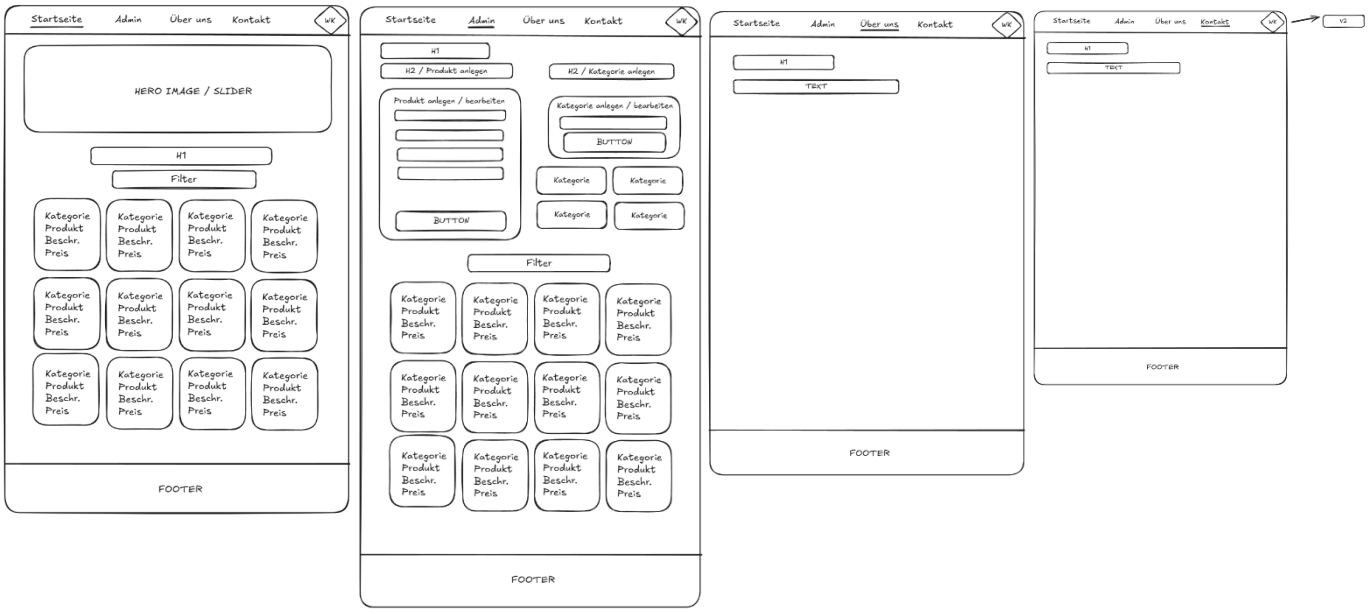


Screen-Mockups (Zeichnungen der Applikation)

In den gezeichneten Screen-Mockups habe ich das User-Interface der Anwendung visualisiert. Auf der Startseite sieht man den Slider, die Kategorie-Filterleiste und die Produkt-Cards im Grind. Die Admin-Seite zeigt übersichtlich das Category Form und das Product Form neben der jeweiligen Liste mit Buttons zum Bearbeiten und Löschen. Ergänzend kommen die Über uns und Kontakt Seiten dazu, die aber noch nicht fertig ausgearbeitet sind.

Die Mockup-Bilder in voller Auflösung sind im Ordner \OnlineshopProject\Documentation\Skizzen zu finden.





Anforderungskatalog

Im Folgenden liste ich die Kernaufgaben meiner Projektidee auf. Jede Aufgabe entspricht einer in sich abgeschlossenen Funktionalität, die ich implementiere und teste.

1. Kategorisierung der Produkte

1.1 **Kategorie anlegen**

- Eingabe eines eindeutigen Namens.
- Validierung: Name darf nicht leer und muss einzigartig sein.
- Anzeige der neuen Kategorie in der Admin-Übersicht und im Filter auf der Startseite.

1.2 **Kategorie bearbeiten**

- Vorbefüllung des Formulars mit aktuellen Werten.
- Anpassung des Namens einer bestehenden Kategorie.
- Wenn der Name geändert wird, behalten alle zugewiesenen Produkte weiterhin diese Kategorie.

1.3 **Kategorie löschen**

- Nur möglich, wenn der zu löschenden Kategorie keine Produkte zugewiesen sind.
- Fehlermeldung im Frontend, falls die Löschung wegen zugeordneter Produkte verweigert wird.

1.4 **Kategorien auflisten**

- Admin-Seite: vollständige Liste aller Kategorien mit Bearbeiten-/Löschen-Buttons.
- Startseite: Listung der Kategorien im Filter.

2. Produktverwaltung

2.1 **Produkt anlegen**

- Eingabemaske für Name, Beschreibung, Preis (min. 0.00 Schritte zu 0.05 CHF), Bild-URL, Kategorie.
- Validierung: Pflichtfeld.

2.2 **Produkt bearbeiten**

- Vorbefüllung des Formulars mit aktuellen Werten.
- Speichern der Änderungen über PUT.

2.3 **Produkt löschen**

- Sofortiges Entfernen aus der Datenbank via DELETE.
- Aktualisierung der Produktliste ohne manuelles Neuladen.

2.4 **Produkte auflisten**

- Anzeige als Grid.
- Responsives Layout: 3-4 Cards pro Zeile je nach Bildschirmbreite.

2.5 **Produkte filtern**

- Kategorie-Filter: Auswahl einer Kategorie zeigt nur zugehörige Produkte.

3. Frontend

3.1 **Routings**

- Seiten: Startseite, Admin, Über uns, Kontakt.
- Aktiver Link wird CSS-basiert unterstrichen.

3.2 **State-Handling**

- useState speichert Eingaben (z. B. Formular-Felder) und Zustände (z. B. Lade-Anzeige) in der Komponente.
- useEffect sorgt dafür, dass beim ersten Laden der Seite automatisch die Produkte und Kategorien aus dem Backend geholt werden.

3.3 **Komponenten-Modularität**

- Wiederverwendbare Komponenten zum Einsatz auf unterschiedlichen Seiten.
- Styling: pro Komponente/Seite eigene .css-Datei und globale Styling-Dateien.

3.4 **Form-Validierung**

- HTML5-Validierung für die Prüfung der Eingaben.
- Alerts für Feedback bei fehlerhaften Eingaben.

4. Backend

4.1 **REST-API**

- Endpoints für Produkte und Kategorien mit CRUD.
- CORS erlaubt Anfragen von http://localhost:5173, damit das React-Frontend im Browser mit dem Backend sprechen kann.

4.2 **Datenmodell & JPA**

- Zwei Java-Klassen: Product gehört zu genau einer Category, und eine Category kann viele Products haben.
- Datenbank-Regel: category_id muss gesetzt sein und blockiert das Löschen einer Kategorie, wenn noch Produkte darauf verweisen.

4.3 **Validierung**

- Im Java-Code prüfen Annotations wie @PositiveOrZero und ein eigener Check, dass der Preis in 5 Rappen-Schritten ist.
- In der Datenbank sorgt eine CHECK-Regel dafür, dass price ≥ 0 und price*100 mod 5 = 0 gilt.

4.4 **Error-Handling**

- Ein zentraler ControllerAdvisor fängt fehlende oder ungültige Eingaben ab.

5. Qualitätssicherung

5.1 **Unit-Tests (Backend)**

- Controller mithilfe von MockMvc prüfen
- Repository-Mapping mit @DataJpaTest testen

5.2 **Komponententests (Frontend)**

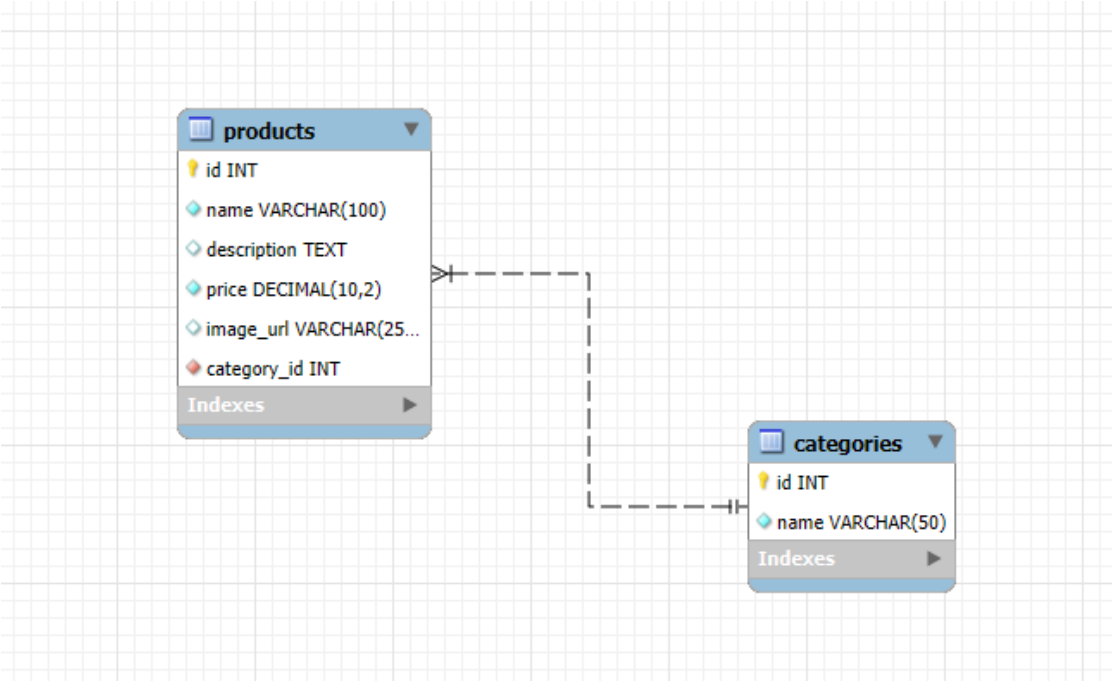
- UI-Bausteine Vitest und React Testing Library testen

Mit diesem Katalog decke ich alle wesentlichen Funktionen meines Systems ab – von der Datenhaltung über die Geschäftslogik bis zur modernen Benutzeroberfläche und der Qualitätssicherung.

Backend

ERD-Modell in der Datenbank

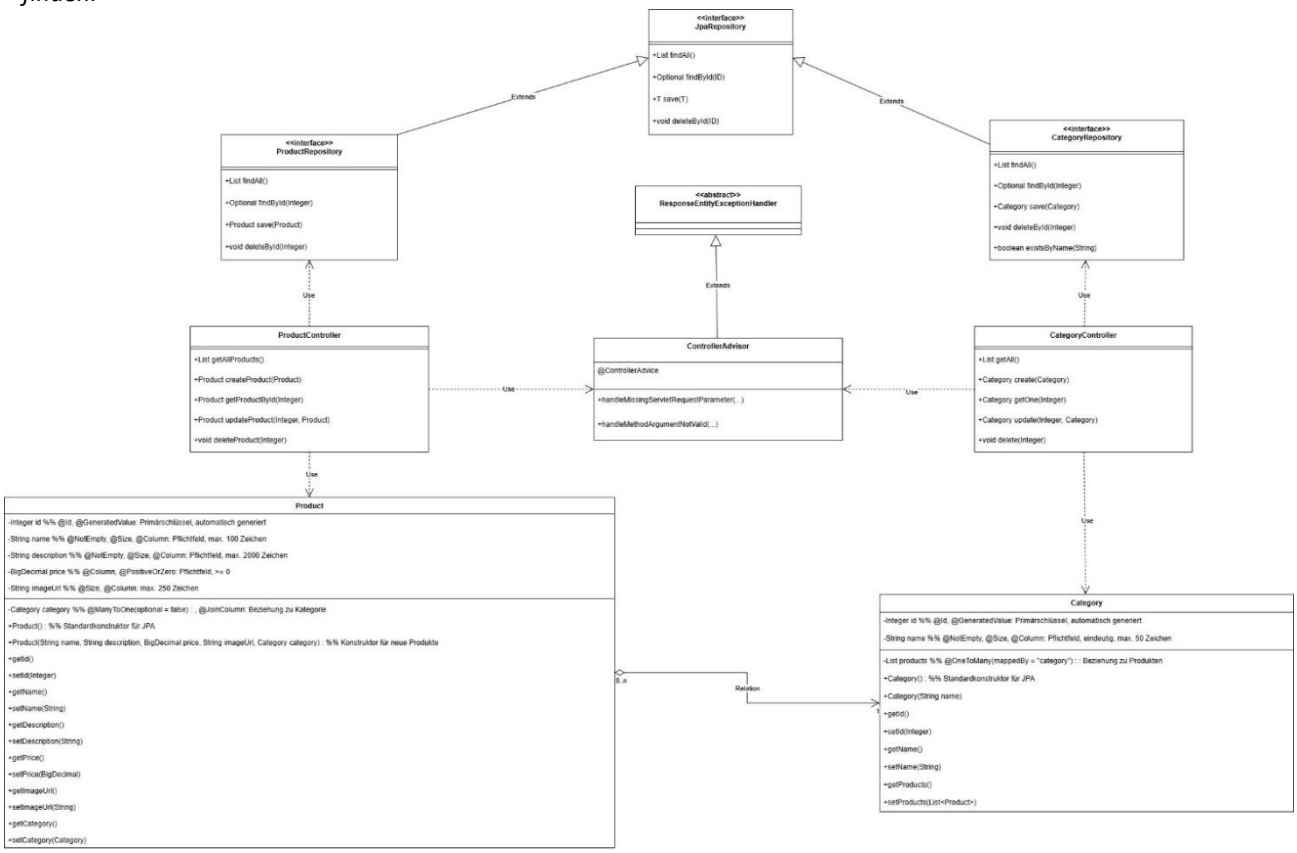
Das Entity-Relationship-Diagramm (ERD) zeigt in grafischer Form, welche Tabellen (Entitäten) es gibt, welche Spalten (Attribute) sie besitzen und wie sie durch Primär- und Fremdschlüssel (Beziehungen) miteinander verknüpft sind. Eine Kategorie kann mehrere Produkte enthalten, jedoch kann ein Produkt nur eine Kategorie haben.



Klassendiagramm Java

In diesem Abschnitt präsentiere ich das UML-Klassendiagramm meiner Java-Anwendung. Es visualisiert die wichtigsten Klassen sowie deren Attribute, Methoden und Beziehungen untereinander. So bekommt man einen klaren Überblick über den Aufbau und die Architektur des Backends.

Das Klassendiagramm in voller Auflösung ist im Ordner \OnlineshopProject\Documentation\Klassendiagramm zu finden.



REST-Schnittstellen

Hier sind alle REST-Endpunkte meines Backends aufgelistet, mit den verwendeten HTTP-Methoden, den URL-Pfaden und einer kurzen Beschreibung ihres Zwecks. So erhält man auf einen Blick sämtliche Informationen, um die Schnittstellen in Front- oder Dritt-Systemen zu integrieren.

1. Produkte

Methoden	URL	Beschreibung
GET	*/api/products*	Alle Produkte abrufen
GET	*/api/products/{id}*	Einzelnes Produkt nach ID abrufen
POST	*/api/products*	Neues Produkt anlegen
PUT	*/api/products/{id}*	Bestehendes Produkt komplett oder teilweise aktualisieren
DELETE	*/api/products/{id}*	Produkt löschen

1.1 GET /api/products

- Request: keine
- Response 200 OK

1.2 GET /api/products/{id}

- Request: Pfadparameter id (Integer)
- Response 200 OK: ein einzelnes Produkt
- Error 404 Not Found: wenn kein Produkt mit dieser ID existiert

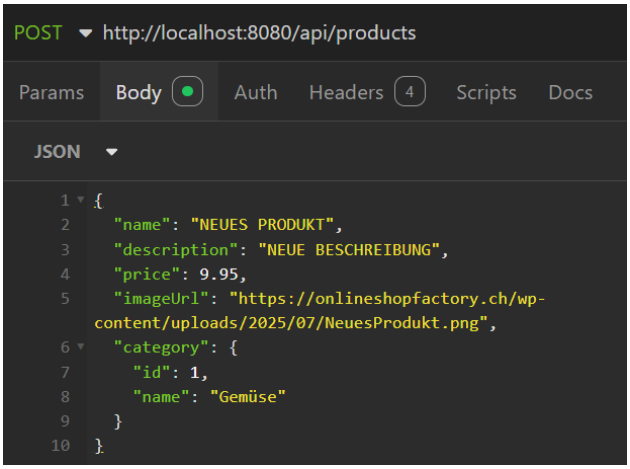

```
[
  {
    "id": 1,
    "name": "Gemüse Produkt 1",
    "description": "Beschreibung für Gemüse Produkt 1",
    "price": 9.90,
    "imageUrl": "https://onlineshopfactory.ch/wp-content/uploads/2025/07/GemuesePic.png",
    "category": {
      "id": 1,
      "name": "Gemüse"
    }
  },
  {
    "id": 2,
    "name": "Gemüse Produkt 2",
    "description": "Beschreibung für Gemüse Produkt 2",
    "price": 9.90,
    "imageUrl": "https://onlineshopfactory.ch/wp-content/uploads/2025/07/GemuesePic.png",
    "category": {
      "id": 1,
      "name": "Gemüse"
    }
  },
```

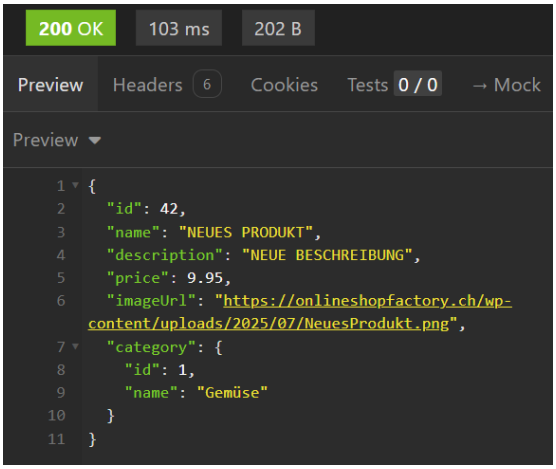


```
{
  "id": 1,
  "name": "Gemüse Produkt 1",
  "description": "Beschreibung für Gemüse Produkt 1",
  "price": 9.90,
  "imageUrl": "https://onlineshopfactory.ch/wp-content/uploads/2025/07/GemuesePic.png",
  "category": {
    "id": 1,
    "name": "Gemüse"
  }
}
```

1.3 POST /api/products

- Request Body (JSON)
- Response 200 OK: das neu angelegte Produkt mit generierter id
- Error 400 Bad Request: Beispielsweise Validierungsfehler





1.4 PUT /api/products/{id}

- Request Body: gleich wie bei Post
- Response 200 OK: das aktualisierte Produkt
- Error 400 Bad Request: Validierungsfehler
- Error 404 Not Found: keine Entität mit der gegebenen ID

1.5 DELETE /api/products/{id}

- Request: Pfadparameter id
- Response 200 OK: leere Antwort oder {}
- Error 404 Not Found: wenn das Produkt nicht existiert

2. Kategorien

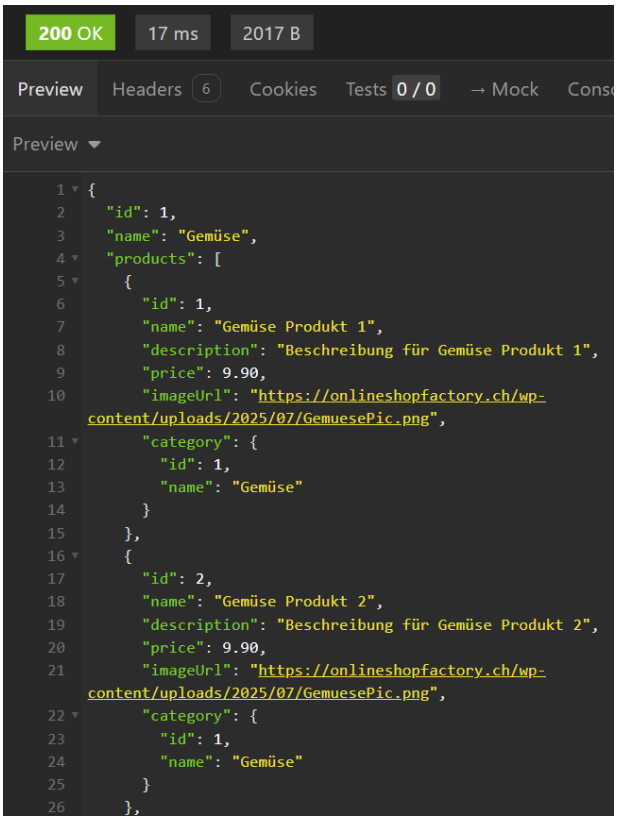
Methode	URL	Beschreibung
GET	*/api/categories*	Alle Kategorien abrufen
GET	*/api/categories/{id}*	Einzelne Kategorie nach ID abrufen
POST	*/api/categories*	Neue Kategorie anlegen
PUT	*/api/categories/{id}*	Bestehende Kategorie bearbeiten
DELETE	*/api/categories/{id}*	Kategorie löschen (nur möglich, wenn keine Produkte in dieser Kategorie vorhanden sind)

2.1 GET /api/categories

- Request: keine
- Response 200 OK

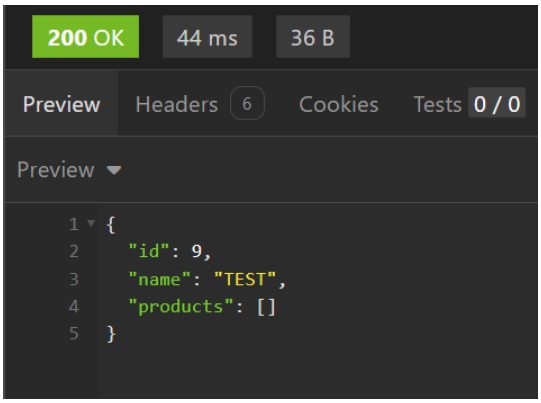
2.2 GET /api/categories/{id}

- Request: Pfadparameter id (Integer)
- Response 200 OK
- Error 404 Not Found: wenn ID nicht existiert



2.3 POST /api/categories

- Request Body (JSON)
- Response 200 OK: das neu angelegte Produkt mit generierter id
- Error 400 Bad Request: Beispielsweise Validierungsfehler



2.4 PUT /api/categories/{id}

- Request Body (JSON): { ''name'': ''geänderterName''}
- Response 200 OK: aktualisierte Kategorie
- Error 400 Bad Request: Beispielsweise Name leer oder bereits vorhanden
- Error 404 Not Found: ID nicht gefunden

2.5 DELETE /api/categories/{id}

- Request: Pfadparameter id
- Response 200 OK: leere Antwort oder {}
- Error 400 Bad Request: wenn noch Produkte verknüpft sind
- Error 404 Not Found: ID nicht vorhanden

3. Datentypen

3.3 Product

Feld	Typ	Beschreibung	Validierung
id	Integer	Eindeutige ID (DB-generiert)	-
name	String	Produktname	Nicht leer, Size 1 - 100
description	String	Freitext-Beschreibung	Nicht leer, Size 1 – 2000
price	BigDecimal	Preis in CHF	≥ 0, Schritt 0.05, PositiveOrZero
imageUrl	String	URL zum Produktbild	Optional, Size max. 250
category	Category	Zugeordnete Kategorie	Nicht null

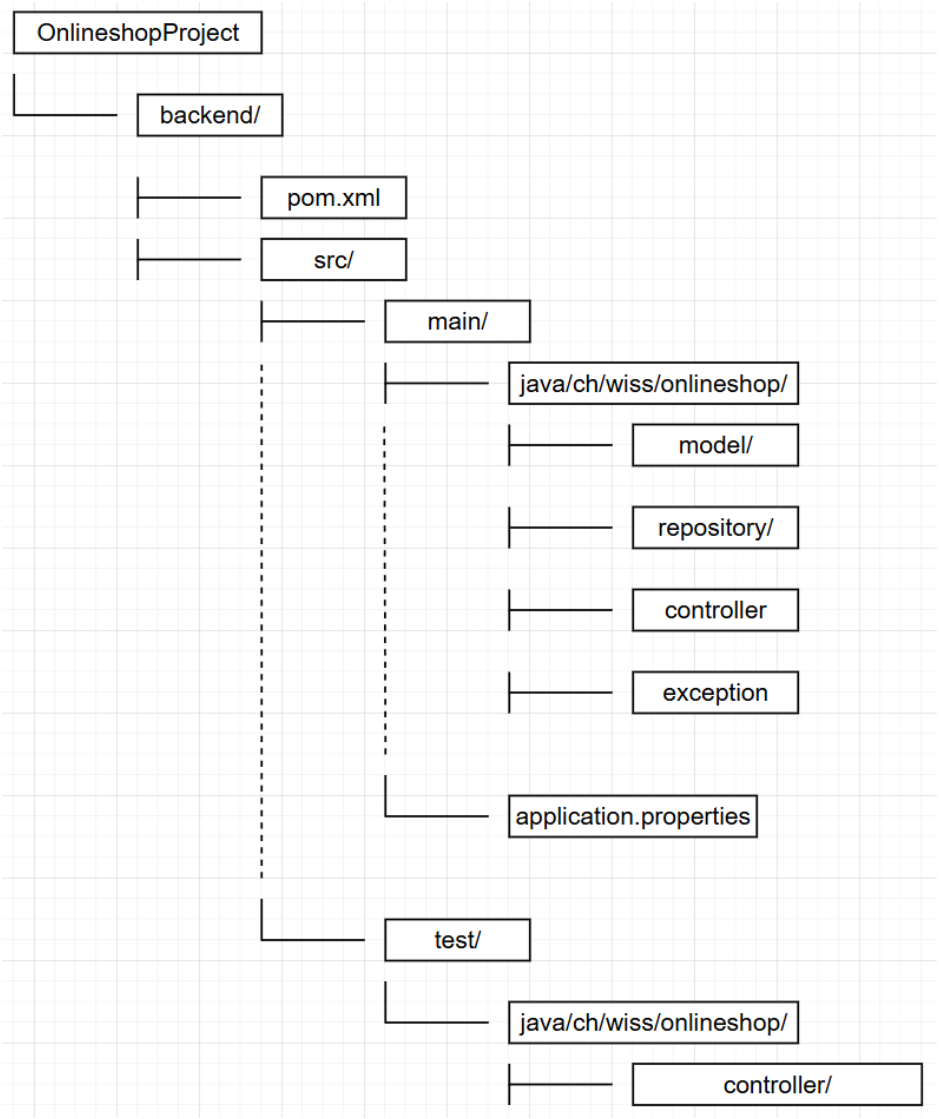
3.4 Category

Feld	Typ	Beschreibung	Validierung
id	Integer	Eindeutige ID (DB-generiert)	-
name	String	Kategoriename	Nicht leer, Size 1 - 50
products	List<Product>	Produktliste	Cascade

Hinweis: Im Response serialisiere ich in Product das gesamte Objekt { id, name }, nicht die products-Liste, um Zyklen zu verhindern.

Ordner- und Dateistruktur

Folgend habe ich eine Übersicht zu den Dateien und Ordnern innerhalb von meinem Projekt erstellt. Diese Darstellung dient dem besseren Verständnis der Struktur. Im gesamten Projekt befinden sich noch mehr Dateien und Ordner als gezeigt, diese Grafik soll nur die wichtigsten Punkte vom Backend beleuchten.



Testplan für das Backend

Dieser Testplan beschreibt die Tests für die Backend-Komponenten des Onlineshop-Projekts. Ziel ist es, die Korrektheit und Robustheit der REST-API für Produkte und Kategorien sicherzustellen, insbesondere im Hinblick auf die Validierung.

1. Testobjekte

- ProductController (Produkt-API)
- CategoryController (Kategorie-API)

2. Testfälle

2.1 ProductController

Funktionale Tests:

- **Alle Produkte abrufen**
    - Erwartung: Die API gibt eine Liste aller Produkte mit den korrekten Feldern zurück.
- **Produkt aktualisieren**
    - Erwartung: Ein bestehendes Produkt kann mit neuen Werten aktualisiert werden. Die Rückgabe enthält die aktualisierten Werte.

Validierungstests:

- **Produkt ohne Name erstellen**
    - Erwartung: Die API lehnt das Erstellen eines Produkts ohne Namen ab und gibt einen Fehler 400 zurück.
- **Produkt mit negativem Preis erstellen**
    - Erwartung: Die API lehnt das Erstellen eines Produkts mit negativem Preis ab und gibt einen Fehler 400 zurück.
- **Produkt mit zu langem Namen erstellen**
    - Erwartung: Die API lehnt das Erstellen eines Produkts mit einem Namen > 100 Zeichen ab (Fehler 400).
- **Produkt mit zu langer Beschreibung erstellen**
    - Erwartung: Die API lehnt das Erstellen eines Produkts mit einer Beschreibung > 2000 Zeichen ab (Fehler 400).

2.2 CategoryController

Funktionale Tests:

- **Alle Kategorien abrufen**
    - Erwartung: Die API gibt eine Liste aller Kategorien zurück.
- **Einzelne Kategorie abrufen**
    - Erwartung: Die API gibt die gewünschte Kategorie zurück, wenn sie existiert.
- **Kategorie erstellen**
    - Erwartung: Eine neue Kategorie kann mit gültigen Daten erstellt werden.

- **Kategorie aktualisieren**
    - Erwartung: Eine bestehende Kategorie kann mit gültigen Daten aktualisiert werden.
- **Kategorie löschen**
    - Erwartung: Eine Kategorie kann gelöscht werden, die API gibt Status 200 zurück.

Validierungstests:

- **Kategorie mit leerem Namen erstellen**
    - Erwartung: Die API lehnt das Erstellen einer Kategorie mit leerem Namen ab (Fehler 400).
- **Kategorie mit zu langem Namen erstellen**
    - Erwartung: Die API lehnt das Erstellen einer Kategorie mit einem Namen > 50 Zeichen ab (Fehler 400).
- **Kategorie mit leerem Namen aktualisieren**
    - Erwartung: Die API lehnt das Aktualisieren einer Kategorie mit leerem Namen ab (Fehler 400).

3. Testdurchführung

- Für jede Testmethode wird das Verhalten der API anhand von Mock-Daten geprüft.
- Die Validierung wird durch gezielte ungültige Eingaben getestet.

4. Akzeptanzkriterien

- Alle Tests müssen erfolgreich durchlaufen.
- Die API muss bei ungültigen Eingaben immer einen Fehler 400 zurückgeben.
- Die Rückgaben der API müssen den Erwartungen entsprechen (korrekte Felder, Statuscodes).

Mit diesen Tests wird sichergestellt, dass die wichtigsten Funktionen und Validierungen der Produkt- und Kategorie-API zuverlässig funktionieren.

Frontend

React Komponenten

In diesem Teil beschreibe ich den Aufbau und die Verwendung der einzelnen React-Komponenten im Frontend. Jede Komponente ist wiederverwendbar und hat im Ordner /styles/components ihre eigene CSS-Datei. Genaue Details zu den Komponenten sind im Code zu finden, das Github-Repository kann gemäss Installationsanleitung heruntergeladen werden.

MainNavBar.jsx

- Zeigt oben auf jeder Seite das Logo und die Hauptnavigation.
    - Nutzt NavLink von React Router, damit der aktuelle Menüpunkt automatisch unterstrichen wird.
    - Enthält Links zu Startseite, Admin, Über uns und Kontakt.
    - CSS: MainNavBar.css regelt Farben, Abstände und das responsive Verhalten.

ProductCard.jsx

- Stellt ein einzelnes Produkt als kleine Karte dar.
    - Erwartet ein product-Objekt mit Feldern wie name, description, price, imageUrl und category.
    - Zeigt Bild, Name, Beschreibung, Preis und Kategorie an.
    - Wenn onDelete und/oder onEdit übergeben werden, blendet es zwei Buttons ein (Löschen und Bearbeiten) und ruft die jeweilige Funktion mit der Produkt-ID oder dem gesamten Objekt auf.
    - CSS: ProductCard.css definiert Rahmen, Schatten und Hover-Effekt.

ProductList.jsx

- Gibt eine Liste von ProductCard-Komponenten als Grid aus.
    - Nimmt products (Array) und categories (für Filter) als Props sowie onDelete/onEdit weiter.
    - Oberhalb des Grids zeigt es eine Zeile mit Buttons, um nach Kategorie zu filtern.
    - Flexbox-Layout in ProductList.css sorgt für responsive Spalten, je nach Bildschirmbreite 3-4 Karten nebeneinander.

ProductForm.jsx

- Formular zum Anlegen oder Bearbeiten eines Produkts.
    - Props:
        - initialProduct (wenn ein Produkt bearbeitet wird)
        - onSubmit(product) (wird nach erfolgreichem Speichern aufgerufen)
    - State mit useState für Name, Beschreibung, Preis, Bild-URL, Kategorie.
    - useEffect befüllt die Felder im Edit-Modus, leert sie im Create-Modus.
    - Frontend-Validierung: Pflichtfelder, Preis ≥ 0 und Schrittweite 0.05.
    - Sendet POST oder PUT via Fetch ans Backend und übergibt das Ergebnis an onSubmit.
    - CSS: ProductForm.css steuert Layout der Inputs und Fehlermeldungen.

CategoryForm.jsx

- Formular zum Anlegen oder Bearbeiten einer Kategorie.
    - Props:
        - initialCategory (für Edit)
        - onSubmit(category)
    - Textfeld für den Kategorienamen.
    - useEffect füllt das Textfeld im Edit-Modus.
    - Sendet POST oder PUT ans Backend.
    - Zeigt per Alert wenn der Name schon existiert oder leer ist.
    - CSS: CategoryForm.css regelt Abstand und Button-Stil.

MainSlider.jsx

- Zeigt auf der Startseite ein grosses Bild mit dem Titeltext.
    - Props:
        - imageUrl (Pfad zum Bild)
        - alt (Alternativtext für Barrierefreiheit)
    - Bewusst nur das Grundgerüst – später kann man hier Autoplay, Pfeile oder Dots ergänzen.
    - CSS: MainSlider.css sorgt dafür, dass das Bild immer die volle Breite ausfüllt und die Caption gut lesbar ist.

Footer.jsx

- Fusszeile auf jeder Seite.
    - Zeigt zentriert den gewünschten Text an.
    - Keine Props erforderlich.
    - CSS: Footer.css definiert Hintergrund- und Textfarbe sowie Abstände.

Ordner- und Dateistruktur

In der folgenden Grafik stelle ich die Ordner- und Dateistruktur vom Frontend dar. Im gesamten Ordner /frontend-vite befinden sich aber noch mehr Ordner und Dateien, hier werden nur die wichtigsten Teile davon beleuchtet.

Testplan für das Frontend

Dieser Testplan beschreibt die Komponententests für das Frontend des Onlineshop-Projekts. Ziel ist es, die Korrektheit und Robustheit der wichtigsten React-Komponenten sicherzustellen, insbesondere im Hinblick auf die Formularvalidierung, die Anzeige von Daten und die Benutzerinteraktionen.

1. Testobjekte

- **ProductForm** (Produkt-Formular)
- **CategoryForm** (Kategorie-Formular)
- **ProductCard** (Produktkarte)
- **CategoryCard** (Kategorie-Karte)

2. Testfälle

2.1 ProductForm

- **Felder ausfüllen und absenden**
    - Erwartung: Nach dem Ausfüllen aller Pflichtfelder und Absenden wird die Callback-Funktion onSubmit mit dem korrekten Payload aufgerufen.
    - Es wird geprüft, ob die Werte für Name, Beschreibung, Preis, Bild-URL und Kategorie korrekt übergeben werden.

2.2 CategoryForm

- **Kategorie anlegen**
    - Erwartung: Nach Eingabe eines gültigen Namens und Absenden wird onSubmit mit dem richtigen Wert aufgerufen.
    - Es wird geprüft, ob ein Alert mit dem Text „Kategorie angelegt: …“ erscheint.

2.3 ProductCard

- **Anzeige und Interaktion**
    - Erwartung: Die Komponente zeigt Name, Beschreibung und Preis des Produkts korrekt an.
    - Beim Klick auf „Löschen“ wird die Callback-Funktion onDelete mit der Produkt-ID aufgerufen.
    - Beim Klick auf „Bearbeiten“ wird die Callback-Funktion onEdit mit dem Produkt-Objekt aufgerufen.

2.4 CategoryCard

- **Anzeige und Interaktion**
    - Erwartung: Die Komponente zeigt den Kategorienamen korrekt an.
    - Beim Klick auf „Löschen“ wird die Callback-Funktion onDelete mit der Kategorie-ID aufgerufen.
    - Beim Klick auf „Bearbeiten“ wird die Callback-Funktion onEdit mit dem Kategorie-Objekt aufgerufen.
    - Es wird geprüft, dass die Buttons nur angezeigt werden, wenn die zugehörigen Props gesetzt sind.

3. Testdurchführung

- Die Tests werden automatisiert mit Vitest und React Testing Library durchgeführt.
- Für jede Komponente werden die wichtigsten Anwendungsfälle und Interaktionen simuliert.
- Die Validierung wird durch gezielte Eingaben und das Auslassen von Pflichtfeldern getestet.

4. Akzeptanzkriterien

- Alle Tests müssen erfolgreich durchlaufen.
- Die Komponenten müssen bei gültigen Eingaben korrekt reagieren und bei ungültigen Eingaben Fehler anzeigen oder Aktionen verhindern.
- Die Callback-Funktionen müssen mit den erwarteten Werten aufgerufen werden.

Mit diesen Tests wird sichergestellt, dass die wichtigsten Komponenten und Benutzerinteraktionen im Frontend zuverlässig funktionieren und die Validierung korrekt umgesetzt ist.

Installationsanleitung

Im Ordner \OnlineshopProject\Documentation befindet sich die ausführliche Installationsanleitung. Hier die wichtigsten Schritte im Überblick:

1. Voraussetzung überprüfen

- Java 21, Maven 3.9.10, Node.js & npm, Git, VSCode/Codium, MySQL Workbench.
    - Mit java --version, mvn -v, node -v, npm -v und git --version kontrollieren.

2. Repository klonen

- git clone https://github.com/Promitecter/OnlineshopProject-Modul-294-295.git
  cd OnlineshopProject

3. MySQL-Datenbank anlegen

- Script CreateDatabase.sql in MySQL Workbench ausführen.
- Zugangsdaten (Host, User, Passwort) merken.

4. Backend konfigurieren & starten

- cd backend
  # application.properties: Datenbank-Credentials eintragen
- mvn clean install
- mvn spring-boot:run
    - läuft auf http://localhost:8080

5. Frontend installieren & starten

- cd frontend-vite
- npm install
- npm run dev
    - läuft auf http://localhost:5173

6. Anwendung im Browser öffnen

- Frontend: http://localhost:5173
- Backend: http://localhost:8080

7. Automatisierte Tests ausführen

- Backend
    - cd backend
    - mvn test
- Frontend
    - cd frontend-vite
    - npm test

8. Mögliche Tasks zur Fehlerbehebung

- Prüfe Ports (8080, 5173), Datenbank-Zugangsdaten, Verzeichniswechsel (cd backend / cd frontend-vite).
- Bei Problemen mvn clean install oder npm install erneut ausführen.

Hilfestellungen

Während der Umsetzung dieses Projekts habe ich eng mit meinem Auszubildenden-Kollegen Sven Landert zusammengearbeitet. Wir haben uns gegenseitig bei der Fehlersuche unterstützt, Code-Reviews durchgeführt und gemeinsam Lösungsansätze erarbeitet. Durch diese Zusammenarbeit lernten wir im Team und konnten spannende Diskussionsrunden führen, was uns beiden sehr bei der Umsetzung der Arbeiten geholfen hat.

Zusätzlich kamen moderne KI-Assistenten zum Einsatz:

- ChatGPT half mir beim Erstellen der Dokumentation und zur Klärung von Fragen.
- GitHub Copilot nutzte ich zur Code-Analyse und zum Kommentieren meines Codes.

Darüber hinaus verwendete ich offizielle Dokumentationsseiten und Youtube Lernvideos:

- Spring Boot Guide (docs.spring.io) für Konfiguration, REST-Controller und JPA.
- React- und Vite-Dokumentation (react.dev, vitejs.dev) für Setup, Routing und Testing.
- Mit Tutorials auf YouTube-Kanälen, wie Brocode oder freeCodeCamp.org, konnte ich direkt sehen, wie ich gewisse Probleme angehen muss. Ich merkte, dass mir das Lernen mit Videos viel Spass macht und ich oft konzentrierter bin, als wenn ich eine lange Dokumentation lesen muss.

Für die Erstellung von Diagrammen griff ich auf Draw.io zurück. Dort entdeckte ich Mermaid, was mir bei der Erstellung der Diagramme eine grosse Hilfe war.

Durch dieses Zusammenspiel aus kollegialer Zusammenarbeit, KI-Assistenz und Dokumentation der Software-Anbieter konnte ich effektiv lernen, Fehler schneller beheben und die Architektur meines Onlineshops sauber und nachvollziehbar dokumentieren.

Fazit

Dieses Projekt hat mir gezeigt, wie wichtig die strukturierte Herangehensweise an eine solche Aufgabe ist. Ich habe verstanden, dass ich mir durch die saubere Aufstellung eines Plans und die Festlegung klarer Rahmenbedingungen enorm viel Arbeit im Code ersparen kann. Ich stellte aber auch fest, dass ich solche Projekte nicht schon am Anfang komplett fertig denken kann. Während dem Coden kamen mir oft auch Ideen in den Sinn, die ich beim Start noch nicht berücksichtigt hatte, oder an die ich überhaupt nicht dachte.

Auf jeden Fall nehme ich viele Learnings aus diesem Projekt mit. Die Erarbeitung hat mich nicht nur fachlich, sondern auch persönlich weitergebracht und gefordert.