DPS-WEBSHOP

Projekt im Rahmen des Moduls „Digitale Produktionssysteme“ an der HTW-Saar

Saarbrücken, den 5. Juli 2025

Koordination: **Florian Speicher, Tim Wall**

Projektpartner:

Name	Matrikelnummer	E-Mail
Laura Dolibois	5012346	lado00005@htwsaar.de
Mathusan Saravarapavan	5012824	masa00021@htwsaar.de
Florian Speicher	5014185	flsp00003@htwsaar.de
Tim Wall	5014365	tiwa00003@htwsaar.de

Inhaltsverzeichnis

1 Vorhaben

1.1 Anforderungen

Unser Projekt umfasst die Entwicklung eines Webshops für Produkte aus dem Bereich Indoor-Farming, speziell angepasst an die Farming-Station der HTW-Saar. Dabei wurden folgende funktionale Anforderungen definiert:

- **Benutzerfunktionen:**
  - Übersicht und Detailansicht aller Produkte
  - Einsicht und Erstellung von Produktbewertungen
  - Filter- und Suchfunktionen (nach Preis, Kategorie etc.)
  - Warenkorb mit Bezahlfunktion
  - Benutzerkonto für schnellere Bestellungen ohne wiederholte Adresseingabe
  - Verwaltung des Nutzerkontos (Erstellung, Bearbeitung, Löschung)
  - Einsicht in die Bestellhistorie
  - Einbindung einer Datenschutzerklärung gemäß rechtlichen Vorgaben
  - Visualisierung der Farming-Station mit Produktzuordnung und Bildern
- **Administrationsfunktionen:**
  - Statistische Auswertungen (z. B. Verkaufszahlen, Kategorienverteilung)
  - Nutzerverwaltung inkl. Adressdatenänderung und Konto-Löschung
  - Vergabe von Admin-Rechten an bestehende Nutzerkonten
  - Produktverwaltung (Bearbeitung, Lagerbestandsverwaltung, Bilder-Upload)
  - Pflege der Farming-Station-Seite
  - Bestellverwaltung mit Statusänderungen

1.2 Projektplan

Im Folgenden sind alle Arbeitspakete (APs) aufgelistet, die zur Entwicklung des DPS-Webshops umgesetzt wurden. Jedes Arbeitspaket beinhaltet konkrete technische und wissenschaftliche Ziele, die messbar sowie inhaltlich anspruchsvoll sind.

# 1	**Projekt-Setup und Basiskonfiguration**	Start-Datum 30.04.2025	End-Datum 30.04.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: In diesem AP wird die technologische Grundlage für das gesamte Projekt geschaffen. Dies umfasst die Erstellung der Projektstruktur, die Initialisierung des Backends und Frontends sowie die Einrichtung der Versionskontrolle. Es kommen hauptsächlich Kommandozeilenwerkzeuge und Shell-Scripting zum Einsatz, um die Projekte zu generieren und zu konfigurieren. • Anlegen der Ordnerstruktur (/00-backend, /01-frontend) • Initialisierung des Spring Boot-Projekts mittels Spring Initializr (inkl. Web, Data JPA, JDBC) • Initialisierung des Vite/React-Projekts (npm create vite@latest . -- --template react-ts) • Konfiguration der SQLite-Datenbankverbindung in application.properties • Einrichtung eines Git-Repositories im Hauptverzeichnis			
Ergebnis des AP 1: Ein lauffähiges Grundgerüst mit einem leeren, aber initialisierten Backend- und Frontend-Projekt, das unter Versionskontrolle steht.			

# 2a	**Datenmodellierung und JPA-Entitäten-Definition**	Start-Datum 01.05.2025	End-Datum 08.05.2025
Hauptverantwortlich: Tim Wall			
Weitere Beteiligte: Florian Speicher			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Dieses AP legt das Fundament der gesamten Anwendungslogik, indem es die Datenstruktur des Webshops definiert und in Code abbildet. Unter Verwendung von Spring Data JPA werden Java-Klassen erstellt, welche die realen Objekte des Systems (wie Produkte, Kunden, Bestellungen) repräsentieren und auf die SQLite-Datenbanktabellen gemappt werden. Die Vorgehensweise umfasst: • Analyse und Entwurf: Identifikation aller notwendigen Datenbank-Entitäten und ihrer Attribute. Entitäten sind Account, Article, Customer, Images, Order und OrderItem, Reviews, Sessions. Speichern in der src/main/resources/db/initdb.sql-Datei. • Implementierung der Entitäten: Erstellung der Java-Klassen und Annotation mit JPA-Annotationen (@Entity, @Id, @GeneratedValue, @Column), um die Abbildung auf Datenbanktabellen zu steuern. • Definition von Beziehungen: Festlegung der Beziehungen zwischen den Entitäten, z. B. @ManyToOne (viele OrderItem-Einträge gehören zu einer Order) und @OneToMany (ein			

Customer kann viele Orders haben). Hierbei werden auch Cascade-Type-Optionen evaluiert, um die Persistenzlogik zu vereinfachen und ein optimiertes Speicherverhalten zu erlauben. • Schema-Generierung: Konfiguration von Hibernate, sodass das Datenbankschema beim Start der Anwendung automatisch aus den definierten Entitäten erstellt oder aktualisiert wird. • Daten-Initialisierung (Seeding): Erstellung einer database.sqlite-Datei im datasource/-Verzeichnis, um die Datenbank für Entwicklungs- und Testzwecke mit initialen Schemata einzupflegen.
Ergebnis des AP 2a: Eine funktionale SQLite-Datenbankstruktur sowie ein SQL-Skript zur Initialisierung der Schemata als auch alle dazugehörigen mit Anmerkungen versehenen JPA-Entity-Klassen im Java-Quellcode.

# 2b	**Backend-Implementierung: REST-API**	Start-Datum 09.05.2025	End-Datum 23.05.2025
Hauptverantwortlich: Tim Wall			
Weitere Beteiligte: -			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Dieses AP fokussiert sich auf die serverseitige Logik. Es werden die Datenstrukturen des Webshops definiert und eine REST-API zur Verfügung gestellt, über die das Frontend Daten abrufen und manipulieren kann. Zum Einsatz kommen JPA für die Datenbank-Abstraktion und Spring Web für die Erstellung der API-Endpunkte. • Identifikation der benötigten REST-API Endpunkte: CRUD (Create: POST, Read: GET, Update: PUT, Delete: DELETE) • Entwicklung von REST-Controllern zur Bereitstellung der API-Endpunkte (z. B. GET /products)			
Ergebnis des AP 2b: Eine funktionale und testbare Spring Boot API, die CRUD-Operationen (Create, Read, Update, Delete) für die Kernobjekte des Webshops (Produkte, Bestellungen etc.) bereitstellt.			

# 3	**Frontend-Implementierung: Mockup**	Start-Datum 01.05.2025	End-Datum 22.05.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: -			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Dieses AP dient zur Entwicklung der Benutzeroberfläche des Webshops und umfasst die Erstellung wiederverwendbarer UI-Komponenten und den initialen Aufbau der einzelnen Seiten ohne Logik.			

Für die Navigation wird react-router-dom und für die Zustandsverwaltung (z. B. des Warenkorbs) die React Context API verwendet. • Erstellung einer logischen Ordnerstruktur für components, pages, helper • Entwicklung von UI-Komponenten (ItemCard, Navbar, Basket) • Aufbau der Hauptseiten (HomePage, ProductPage, CheckoutPage) • Einrichtung des clientseitigen Routings • Umsetzung des Stylings (mit MUI, sx sowie CSS)
Ergebnis des AP 3: Eine interaktive React-Anwendung, die Dummy-Produktdaten anzeigen kann, die Navigation zwischen Seiten ermöglicht und Benutzerinteraktionen lokal verarbeiten kann.

# 4	**Integration von Frontend und Backend**	Start-Datum 23.05.2025	End-Datum 18.06.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: In diesem AP werden Frontend und Backend miteinander verbunden, um über die REST-API grundlegende Funktionen im Frontend bereitzustellen. • Konfiguration von CORS im Spring Boot-Backend und Einrichtung essenzieller fetch und useQuery Befehle im Frontend, um API-Anfragen an das Backend weiterzuleiten. • Implementierung der API-Kommunikation zur Anbindung an das Backend innerhalb des Frontends (u. a. GET /article, POST /order, GET /image)			
Ergebnis des AP 4: Eine interaktive React-Anwendung, die Produktdaten von der API anzeigen, die Navigation zwischen Seiten ermöglichen und Benutzerinteraktionen verarbeiten kann. Das Login fällt nicht unter diesen Punkt.			

# 5a	**Backend-Implementation: Login**	Start-Datum 24.05.2025	End-Datum 08.06.2025
Hauptverantwortlich: Tim Wall			
Weitere Beteiligte: -			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Dieses AP konzentriert sich auf die serverseitige Erstellung eines sicheren Account-Systems. Der Schwerpunkt liegt auf der korrekten Speicherung von Benutzerpasswörtern mittels BCrypt, das automatisch einen Salt für jedes Passwort generiert und diese hasht, wodurch eine hohe Sicherheit gegen Angriffe geboten wird. Die Umsetzung erfolgt in folgenden Schritten: • Abhängigkeit hinzufügen: Die benötigte Abhängigkeit wird zur pom.xml hinzugefügt, um die notwendigen Sicherheitsbibliotheken zu integrieren.			

- Service erstellen: Ein PasswordService wird erstellt, welcher sicherstellt, dass das Passwörter nach BCrypt-Standards gesaltet, gehasht und gespeichert werden.
- Sicherheitskonfiguration
- Service-Logik: Im AccountService wird eine Registrierungsmethode implementiert. Diese Methode nimmt die Benutzerdaten entgegen, hasht das übermittelte Passwort mit dem PasswordService und speichert anschließend den neuen Benutzer in der Datenbank. Auch wird ein SessionService implementiert, welcher sich um das Verwalten von Sessions sorgt.
- Endpunkt hinzufügen: Der Endpunkt für Accounts und Sessions wird mit den bekannten CRUD-Operationen gefertigt.

Ergebnis des AP 5a: Ein sicherer API-Endpunkt zur Benutzerregistrierung. Die Passwörter werden niemals im Klartext gespeichert, sondern ausschließlich als BCrypt-Hash in der SQLite-Datenbank abgelegt. Ein weiterer Endpunkt steht bereit, um die Daten des angemeldeten Benutzers abzurufen.

# 5b	**Frontend-Implementation: Login**	Start-Datum 07.06.2025	End-Datum 15.06.2025

Hauptverantwortlich: Florian Speicher

Weitere Beteiligte: -

Must- oder May-Have? Must-Have

Inhaltsbeschreibung gesamt:
Dieses AP bindet das im Backend erstellte Account-System in die React-Oberfläche ein. Es wird die Bibliothek React Query (bzw. @tanstack/react-query) verwendet, um die Kommunikation mit den API-Endpunkten effizient zu gestalten und den Server-State (z. B. den eingeloggten Benutzer) zu verwalten.

- Das Design umfasst ein Formular, welches die Registrierungsdaten in einem anschaulichen MUI-Fenster aufnimmt und per Tastenklick an den POST /account-Endpunkt sendet. Des Weiteren sollen Fehler abgefangen und dem Nutzer ordentlich präsentiert werden. Dieses Formular soll auch das Login-System per POST /session bedienen können.
- UseAccount ist ein selbstgeschriebener Hook, der useQuery verwendet, um Daten vom GET /account-Endpunkt abzurufen. Dieser holt die Daten des eingeloggten Benutzers und stellt sie global zur Verfügung. Er kann genutzt werden, um UI-Elemente wie "Login" vs. "Logout" anzuzeigen.

Ergebnis des AP 5b: Eine voll funktionsfähige Registrierungs- und Login-Funktionalität im Frontend. Die Anwendung kann Benutzer registrieren, den Authentifizierungsstatus verwalten und die Benutzeroberfläche dynamisch an den Status des eingeloggten Benutzers anpassen.

# 6	**Frontend: Cookie-basierte Persistenz**	Start-Datum 14.06.2025	End-Datum 15.06.2025

Hauptverantwortlich: Florian Speicher

Weitere Beteiligte: -

Must- oder May-Have? May-Have

Inhaltsbeschreibung gesamt:
Dieses Arbeitspaket fokussiert sich auf die Implementierung einer persistenten Warenkorb- und Account-Session im Frontend durch den Einsatz von Cookies. Ziel ist es, dass sowohl der Warenkorb (Basket) als auch der Login-Status eines Users über Seiten-Reloads und Browser-Sitzungen hinweg erhalten bleiben.
Folgende Implementationen sind zu fertigen:

- BasketProvider: Entwicklung eines React Context Providers (BasketProvider), der den Warenkorb-Zustand verwaltet. Bei jeder Änderung des Warenkorbs wird der aktuelle Zustand als JSON-String in einem Cookie gespeichert. Beim Initialisieren des Providers wird geprüft, ob ein Basket-Cookie existiert, und ggf. der gespeicherte Zustand wiederhergestellt.
- AccountProvider: Entwicklung eines verbesserten React Context Providers (AccountProvider), der den Login-Status sowie User-Informationen verwaltet. Nach erfolgreichem Login werden Session-Daten (z. B. E-Mail und Session-Token) in einem Cookie gespeichert. Beim Initialisieren des Providers wird geprüft, ob ein gültiges Auth-Cookie existiert, und der Login-Status entsprechend gesetzt.

Ergebnis des AP 6: Ein implementierter BasketProvider und AccountProvider, die jeweils ihren Zustand persistent per Cookie speichern und laden. Die Anwendung merkt sich den Warenkorb und den Login-Status des Users auch nach einem Seiten-Reload oder Browser-Neustart.

# 7	**Farming-Station-Seite**	Start-Datum 05.06.2025	End-Datum 18.06.2025

Hauptverantwortlich: Laura Dolibois

Weitere Beteiligte: Tim Wall

Must- oder May-Have? May-Have

Inhaltsbeschreibung gesamt:
Entwicklung einer Benutzeroberfläche des Webshops, welche es dem Nutzer ermöglicht an einer Visualisierung der Farming-Station die passenden Artikel zu kaufen. Dabei sollen bereits existierende UI-Komponenten genutzt werden, um die Code-Komplexität gering zu halten.

- Erstellung einer anschaulichen Nutzeroberfläche für Farming-Station-Komponenten
- Anpassung des Datenbankschemas für das Speichern der Farming-Station-Bilder in Relation zu den Artikeln
- Erweiterung des Backends um einen Farming-Station-Endpunkt, durch welchen die Bilder der Farming Station gespeichert und geändert werden können
- Integration des Endpunktes mit der Benutzeroberfläche durch useQuerys

Ergebnis des AP 7: Eine Nutzeroberfläche mit Artikeln, welche bei Produktauswahl ein passendes annotiertes Bild der Farming-Station aus dem Backend anzeigt. Außerdem besteht die Option, alle benötigten Artikel zeitgleich in den Warenkorb hinzuzufügen.

# 8	**Frontend: Theming**	Start-Datum 31.05.2025	End-Datum 07.06.2025
Hauptverantwortlich: Mathusan Saravarapavan			
Weitere Beteiligte: Florian Speicher			
Must- oder May-Have? May-Have			
Inhaltsbeschreibung gesamt: Dieses AP widmet sich der Erstellung einer globalen Theming-Lösung, die es dem Benutzer ermöglicht, zwischen einem hellen (Light Mode) und einem dunklen (Dark Mode) Design zu wechseln. Die Implementierung nutzt die React Context API, um den Theme-Status zentral zu verwalten und für alle Komponenten zugänglich zu machen. Die gewählte Einstellung wird im localStorage des Browsers gespeichert, um die Auswahl des Benutzers über Sitzungen hinweg beizubehalten. • Globale Verfügbarkeit: Der ThemeProvider wird in der App-Datei (App.tsx) so integriert, dass er die gesamte Anwendung umschließt. Dadurch kann jede Komponente auf den Theme-Kontext zugreifen. • Die korrekte Integration innerhalb der Komponenten wird an den jeweiligen Code-Stellen vorgenommen. Sie ist nicht für jede Komponente notwendig, da die Themes teilweise von den Parents stammen, wird jedoch bevorzugt.			
Ergebnis des AP 8: Ein global funktionaler Dark-/Light-Mode-Switch in der NavBar, welcher sinnvolle Theme-Farben bereitstellt. Jede Komponente hat einen zugewiesenen Dark-Mode.			

# 9	**Frontend: Localisation**	Start-Datum 23.05.2025	End-Datum 31.05.2025
Hauptverantwortlich: Laura Dolibois			
Weitere Beteiligte: Mathusan Saravarapavan, Florian Speicher, Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Dieses AP befasst sich mit der Localisation (i18n), welche im Frontend vorgenommen wird. Es soll nach der Browser-Konfiguration des Nutzers eine passende Sprache ausgewählt und der Webshop in dieser angezeigt werden. Die gewählte Sprache wird im Account gespeichert, hat jedoch keine weitere Auswirkung auf die Logik. • Import/Erstellen einer geeigneten i18n-Lösung für React • Möglichkeit, die Lokalisierung im Quellcode anzupassen • Wählen geeigneter Sprachen (Deutsch und Englisch) • Integration in allen Komponenten durch einen t(locKey)-Aufruf			
Ergebnis des AP 9: Ein Grundgerüst für die Lokalisierung in den Sprachen Englisch und Deutsch. Alle bis Projektende bestehenden Zeichenketten existieren in jeder Lokalisierung und werden korrekt angezeigt.			

# 10a	**Backend: Admin-Accounts und dazugehörige Endpunkte**	Start-Datum 05.06.2025	End-Datum 12.06.2025
Hauptverantwortlich: Tim Wall			
Weitere Beteiligte: -			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Dieses AP dient der Entwicklung einer Möglichkeit zwischen Administratoren und normalen Kunden des Webshops zu unterscheiden. Außerdem ist diese Implementation zu verwenden, um Statistiken über Kaufverhalten der Nutzer an den Endpunkten abzufragen. • Erweiterung des Datenbankschemas, sodass Accounts als Administratoren zu erkennen sind • Herausfinden, welche Statistiken von Interesse und nicht zu kleinschneidig sind (Verkaufs-zahlen pro Kategorie, Umsatz pro Kategorie, Bestellungszustand, Lagerbestand in %) • Backend-Endpunkt erstellen, welcher ermöglicht nach Adminrechten zu überprüfen • Backend-Endpunkt zum Abfragen von Statistiken erstellen, mit gleichen Parametern wie zuvor, außerdem Export der Daten in passendem Format • Backend-Endpunkte, welche von normalen Nutzern nicht erreicht werden dürfen, sind gleichermaßen abzusichern			
Ergebnis des AP 10a: Ein Admin-System, welches die in der Inhaltsbeschreibung genannten Punkte erfüllt.			

# 10b	**Frontend: Admin-Page Statistik**	Start-Datum 16.06.2025	End-Datum 18.06.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Entwicklung einer Benutzeroberfläche des Webshops, welche es dem Administrator ermöglicht eine Visualisierung des Kaufverhaltens der Nutzer anzuzeigen. • Import einer geeigneten Statistikvisualisierung für React (MUI-X) • Erstellung einer anschaulichen Page für Statistiken • Integration der REST-API durch useQuerys • Anpassung der Statistik-Gestaltung zum visuellen Unterscheiden der verschiedenen Typen (Umsatz, Lager, Bestellungen) • Sicherstellen, dass die Seite nur als Administrator verfügbar ist			
Ergebnis des AP 10b: Eine vollständige Oberfläche mit aus dem Backend abgerufenen Statistiken, welche nur mit korrekten Berechtigungen aufrufbar ist.			

# 10c	**Frontend: Admin-Page Items & Accounts**	Start-Datum 13.06.2025	End-Datum 18.06.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Entwicklung einer Benutzeroberfläche des Webshops, welche es dem Administrator ermöglicht eine Visualisierung der Account- sowie Artikeldaten anzeigen zu lassen. • Import einer geeigneten Statistikvisualisierung für React (MUI-X) • Erstellung einer jeweils anschaulichen Page für Artikel und Accounts • Integration der REST-API durch useQuerys • Anpassung der Gestaltung zur visuellen Unterstützung (Soll- vs. Ist-Lagerfüllstand, Bewertung, Preis nach Rabattberechnung, Administrator-Zustand) • Dem Administrator nur sinnvolle Eingaben erlauben (kein Ändern der Kunden-E-Mail, kein Ändern der Artikelbewertung) • Sicherstellen, dass die Seite nur als Administrator verfügbar ist			
Ergebnis des AP 10c: Eine vollständige Oberfläche mit aus dem Backend abgerufenen Daten, welche nur mit korrekten Berechtigungen aufrufbar ist und die oben genannten Kriterien erfüllt.			

# 10d	**Frontend: Admin-Page Order**	Start-Datum 12.06.2025	End-Datum 22.06.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Tim Wall			
Must- oder May-Have? May-Have			
Inhaltsbeschreibung gesamt: Entwicklung einer Benutzeroberfläche des Webshops, welche es dem Administrator ermöglicht eine Visualisierung der Bestellungen sowie deren Verwaltung anzeigen zu lassen. • Erstellung einer anschaulichen Page für Bestellungen (Design ähnlich einem Kanban-Board) • Integration der REST-API durch useQuerys • Anpassung der Gestaltung zur visuellen Unterstützung (Kanban-Board-Design, Drag & Drop Funktionalität der Orders, Öffnen der Bestellungen, Fehleranzeige) • Dem Administrator die Möglichkeit geben, Bestellungen zu verwalten (*Bestellt, Storniert, in Versand, Zugestellt, Lieferschwierigkeiten*) • Sicherstellen, dass die Seite nur als Administrator verfügbar ist			
Ergebnis des AP 10c: Eine vollständige Oberfläche mit aus dem Backend abgerufenen Daten, welche nur mit korrekten Berechtigungen aufrufbar ist und die oben genannten Kriterien erfüllt.			

# 11	**Testen**	Start-Datum 18.06.2025	End-Datum 23.06.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Laura Dolibois, Mathusan Saravarapavan, Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Überprüfung aller in den Arbeitspaketen genannten Inhalte und Ergebnisse nach ihrer Richtigkeit und Fehlerlosigkeit.			
Ergebnis des AP 11: Eine fehlerfreie Software, bestehend aus Backend und Frontend, welche den zuvor erstellten Plan abbildet.			

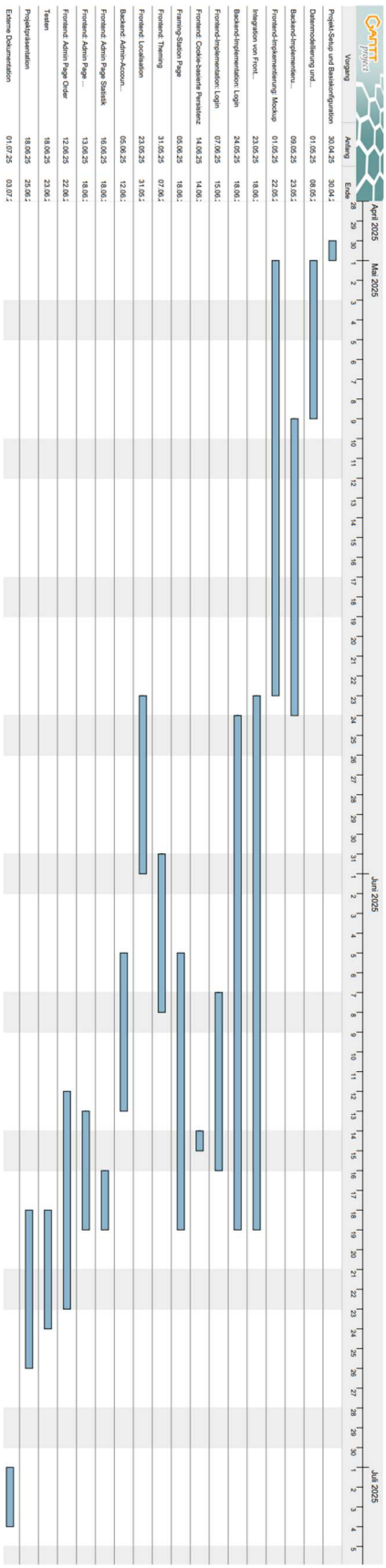
# 12	**Projektpräsentation**	Start-Datum: 18.06.2025	End-Datum: 25.06.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Laura Dolibois, Mathusan Saravarapavan, Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: • Erstellung einer PowerPoint-Präsentation, welche Motivation, alle Arbeitspakete sowie technische Dokumentation enthalten und innerhalb der DPS-Vorlesung vorgetragen wird • Zuweisung der Verantwortlichkeit bezüglich Themen zu allen Beteiligten • Festlegung eines Datums, an welchem der Vortrag gehalten wird.			
Ergebnis des AP 12: Eine PowerPoint-Präsentation, bestehend aus den oben genannten Inhalten sowie eine Präsentationsplanung, welche jedem Beteiligten einen Redeteil zuweist.			

# 13	**Externe Dokumentation**	Start-Datum 01.07.2025	End-Datum 03.07.2025
Hauptverantwortlich: Florian Speicher			
Weitere Beteiligte: Laura Dolibois, Mathusan Saravarapavan, Tim Wall			
Must- oder May-Have? Must-Have			
Inhaltsbeschreibung gesamt: Erstellung eines Dokumentes, welches folgende Punkte beinhaltet und ausführt: • Projektlogo, Projekt-Titel • Projektpartner • Vorhaben (2.5 - 6 Seiten) • Anforderungen • Projektplan • Umsetzung (5 – 13 Seiten) • Reflektion • Architektur und Technologie-Stack • Features • Dokumentation			
Ergebnis des AP 13: Dieses Dokument, bestehend aus den oben genannten Inhalten.			

1.2.1 Meilensteine

- Initiales Backend mit Datenbankdesign und CRUD-Operationen
- Initiales Frontend mit Dummy-Daten
- Erstes Walking Skeleton: Frontend mit implementierter Backend-Anbindung um Artikel via Rest
- Funktionierendes Register-/Login-/Session-System
- Funktionaler ThemeProvider
- Vollständiges Admin-Panel mit Admin-Account
- Fertige Dokumentation und Testphase

1.2.2 GANTT-Chart



2 Umsetzung

2.1 Reflexion

Diese Reflexion bewertet den Projektverlauf anhand der definierten Meilensteine und analysiert, inwieweit die Anforderungen erfüllt wurden, wie die Teamarbeit funktionierte und wo es zu Abweichungen vom ursprünglichen Plan kam.

Das Projekt war in klar strukturierte Arbeitspakete (APs) gegliedert, deren Verantwortlichkeiten eine effiziente Aufgabenverteilung ermöglichten. Die festgelegten Meilensteine dienten dabei als Orientierungsrahmen und wurden systematisch abgearbeitet.

2.1.1 Umsetzung der Meilensteine

I. **Initiales Backend mit Datenbankdesign und CRUD-Funktionalität:**
Dieser Meilenstein wurde erfolgreich und planmäßig umgesetzt. Auf Basis des dedizierten Arbeitspakets für das Datenmodell wurden JPA-Entitäten definiert und die Beziehungen zwischen ihnen modelliert. Durch die Nutzung von Spring Data JPA konnten wir effizient Repositories erstellen, die grundlegenden CRUD-Funktionalitäten out-of-the-box bereitstellten. Die REST-Controller wurden darauf aufbauend implementiert und boten eine simple und testbare API für alle Kernobjekte. Die Entscheidung für eine dateibasierte SQLite-Datenbank erwies sich für die Entwicklungsphase als äußerst vorteilhaft, da sie das Setup und dadurch die Reproduzierbarkeit auf anderen Geräten erheblich vereinfachte.

II. **Initiales Frontend mit Dummy-Daten:**
Parallel zum Backend wurde das Frontend entwickelt. Wie im AP für die Frontend-Implementierung vorgesehen, wurden die grundlegenden UI-Komponenten und das Layout der Homepage sowie Artikelseite erstellt. Um die Entwicklung zu entkoppeln und nicht auf das fertige Backend warten zu müssen, wurde zunächst mit statischen Dummy-Daten gearbeitet. Die genutzte MUI Library hat dabei ein schnelles Prototyping und dadurch frühes UI/UX-Feedback ermöglicht.

III. **Erstes Walking Skeleton:**
Dies war der kritischste und zugleich erfolgreichste Meilenstein. Der im AP zur Integration im Voraus kommunizierte definierte Datenfluss ermöglichte eine reibungslose Verbindung. Die Dummy-Daten wurden durch API-Aufrufe mittels React-Query (useQuery) ersetzt. Das Ergebnis war ein "lebendiges" System, in dem das Frontend dynamisch Produktdaten vom Backend anzeigte. Dieser Erfolg bestätigte die korrekte Auswahl unserer Architektur.

IV. **Funktionierendes Register-/Login-/Session-System:**
Die Umsetzung dieses Meilensteins erfolgte in zwei spezialisierten APs. Im Backend wurde Spring Security integriert, um die API-Endpunkte abzusichern und ein Registrierungssystem zu implementieren, das Passwörter sicher mittels BCrypt speichert. Im Frontend wurde die Anbindung durch useQuery für die Registrierung und zum Abrufen von Account-Daten realisiert. Die Session-Persistenz wurde wie im AP vorgesehen durch den AccountProvider

und den nachträglichen Einsatz von Cookies sichergestellt, was ein nahtloses Benutzererlebnis über Seiten-Reloads hinweg ermöglichte. Die Verwaltung von Cookies innerhalb von React war uns im bisherigen Verlauf unbekannt, weswegen wir eine lange Rechercheeinheit einlegen mussten, um dieses Thema zufriedenstellend zu realisieren.

V. **Funktionaler ThemeProvider:**
Dieser Meilenstein zur Verbesserung der Nutzererfahrung wurde exakt nach AP umgesetzt. Der realisierte ThemeProvider erlaubt den Wechsel zwischen Dark- und Light-Mode. Durch die Implementierung in jedem React-Baustein wird die Benutzerauswahl konsequent ersichtlich. Dies war ein schnell implementiertes, aber wirkungsvolles Feature, besonders bei nächtlichem Webseitengebrauch.

VI. **Vollständiges Admin-Panel mit Admin-Account:**
Das Admin-Panel ist eine Erweiterung der bestehenden Systeme. Im Backend wurden neue, durch einen Account-Flag geschützte Endpunkte geschaffen, die administrative Aktionen (z. B. Produkte hinzufügen oder bearbeiten) erlauben. Im Frontend wurde ein neuer, geschützter Bereich (/admin) implementiert, der auf diese Endpunkte zugreift. Die solide Basis des bestehenden Login-Systems erlaubte eine effiziente Umsetzung, jedoch wurde wegen eines State-Fehlers das Admin-Panel manchmal unerlaubterweise angezeigt. Nach einer kollaborativen Debugging-Session konnte dieser Fehler schnell behoben werden.

VII. **Fertige Dokumentation und Testphase:**
Schon zum Anfang wurde die API-Dokumentation mittels Swagger/OpenAPI automatisiert generiert. Die Testphase umfasste manuelle End-to-End-Tests der Kern-User-Flows (Registrierung, Produktkauf, Admin-Aktionen) mit hoher Coverage. In Retrospektive hätten sich auch automatisierte Unit-Tests angeboten und uns möglicherweise Zeit erspart, jedoch wurden diese wegen mangelhaftem Wissen über das Thema innerhalb der React-Welt nicht realisiert. Bezüglich des Backends haben die Programmierkapazitäten für ausgiebiges Testen gefehlt, weswegen sie auch dort vergeblich gesucht werden.

2.1.2 Aufgabenverteilung und Abweichungen in der Zusammenarbeit

Die Aufgabenverteilung war initial klar nach technologischen Domänen getrennt: Ein Teammitglied war primär für das Spring Boot Backend verantwortlich, die anderen für das React Frontend. Diese Aufteilung basierte auf den Hauptverantwortlichkeiten, die in den Arbeitspaketen definiert waren. Wenn es jedoch zu Staus oder ähnlichen Problemen kam, wurde diese Aufteilung auch aufgebrochen. Dies war besonders zum Ende hin notwendig, um in unserem strikt gesetzten Zeitplan das Admin-Panel so umzusetzen, wie es jetzt ist.

I. **Funktionierende Aspekte:**
Die klare Trennung funktionierte bei der Implementierung der reinen Backend-Logik (Meilenstein 1) und Frontend-Komponenten (Meilenstein 2) hervorragend. Die API-Schnittstelle wurde frühzeitig als Vertrag zwischen den beiden Domänen definiert, was eine parallele

und störungsfreie Entwicklung ermöglichte. Regelmäßige Meetings innerhalb der Universität oder im digitalen Raum über Discord sicherten den Informationsfluss und den stetigen Programmierfortschritt.

II. **Abweichungen und Lerneffekte:**

Die strikte Trennung der Verantwortlichkeiten stieß bei der Umsetzung von domänenübergreifenden Features wie dem Login- und Session-System (Meilenstein 4) an ihre Grenzen. Die APs für die Admin-Page im Frontend erforderten tiefes Verständnis darüber, wie die Daten im Backend zusammengefasst und versendet wurden. Hier wich die Umsetzung von der Planung ab: Statt einer alleinigen Zuständigkeit des Frontend-Entwicklers wurde dieses AP in engem Peer-Programming gelöst.
Diese Abweichung war jedoch keine Schwäche: Für komplexe Full-Stack-Features ist eine kollaborative statt getrennter Arbeitsweise deutlich effektiver. Die initialen APs hätten hier eine geteilte Verantwortlichkeit vorsehen sollen. Diese Erfahrung wird in die Planung zukünftiger Projekte einfließen.

III. **Fazit:**

Das Projekt wurde erfolgreich abgeschlossen, und alle definierten Meilensteine wurden erreicht. Die gewählte Architektur hat sich als robust, flexibel und skalierbar erwiesen. Die Teamkollaboration war trotz kleiner Abweichungen von der initialen Planung sehr produktiv und hat zu wichtigen Lerneffekten für zukünftige Projekte geführt.

2.2 Architektur und Technologie-Stack

Unser Produkt besteht aus einer klassischen Client-Server-Architektur. Dabei stellt unser Backend den Server und das Frontend den Client dar, der später beim Kunden läuft. Die Hauptrechenleistung sollte bei uns deswegen im Backend liegen und das Frontend nur zur einfachen Darstellung dienen, damit die Nutzerexperience gesteigert wird und lange Ladezeiten vermieden werden. Die Schnittstelle zwischen den beiden getrennten Systemen wird über REST bereitgestellt. Wir haben uns bewusst für diese Architektur entschieden, da diese der Standard für solche Anwendungen ist und auch leicht in eine Cloud für einen hochskalierbaren Shop geändert werden kann.

2.2.1 Backend

Für unser Backend verwenden wir eine gängige Sprache für Webserver-Anwendungen. Da alle Gruppenmitglieder im Studium schon auf Java gestoßen sind, war dies unserer Meinung nach der beste Weg, einen direkten und einfacheren Projektstart ohne lange Einarbeitung zu ermöglichen. Da Java von allein keine direkte Anbindung einer REST-Schnittstelle ermöglicht, haben wir Spring Boot als Framework gewählt. Dieses ist das am meisten verwendete Java-Framework und bietet eine einfache und effiziente REST-Anbindung, aber auch mit Hilfe von Spring Data JPA und Hibernate eine gute Anbindung zu allen gängigen Datenbanksystemen. Unser

Backend wurde nach dem in Spring weit verbreitenten Layer-Modell aufgebaut, sodass eine REST-Anfrage auf der obersten Schicht in den Layer-Baum eintritt und dann bis zur untersten Schicht wandert, um die Persistenz-Schicht aufzurufen. Danach wandert die Anfrage noch einmal in entgegengesetzteer Richtung zum User zurück. Unsere Layer beginnen mit den Controllern, welche unsere REST-Schnittstellen bereitstellen und die nächstuntere Ebene die Services aufrufen. Diese Services sind für die Verarbeitung der Daten zuständig und beinhalten unsere Geschäftslogik. Die davon bei Bedarf aufgerufene Persistenz-Schicht zur Datenbank besteht nun aus mehreren Elementen, die zusammenspielen. Einmal haben wir Repositories, die über Spring Data JPA und Hibernate Datenbankabfragen ausführen können und Entitäten, die ein Abbild unserer Datenbank-Entität im Code bilden um dadurch eine höhere Sicherheit und besseren Schutz vor Angriffen wie SQL-Injections bieten.

In unserem Backend haben wir stets Java-Coding-Guidelines sowie Best-Practices, wie die SOLID-Prinzipien, und weitere Werte der Softwaretechnik eingehalten, sodass hohe Wartbarkeit, Erweiterbarkeit und Modularität gewährleistet sind.

2.2.2 Frontend

Als Frontend nutzen wir TypeScript in React. Da sich einige in unserer Gruppe schon mit React auskannten, war dies auch eine Wahl basierend auf den vorhandenen Fähigkeiten. Jedoch muss man sagen, dass React generell eine der besten Wahlen ist, da es ein sehr weit verbreitetes JavaScript Framework mit viel Dokumentation und vielen Features ist. TypeScript wurde verwendet, um Verwirrung zu vermeiden, die durch typlose Sprachen entstehen kann. Um gutaussehende Komponenten zu gewährleisten, nutzten wir Material UI als Komponenten-Bibliothek, da diese auf dem weitbekannten Google Material Design aufbauen und leicht in React einzubinden sind. Außerdem gibt es eine gute Dokumentation über jede einzelne Komponente, die auch konkrete Anwendungsfälle zeigt und somit gerade für im Frontend unerfahrene Personen eine gute Basis zum Lernen bildet.

2.2.3 Datenbank

Als Datenbank benutzen wir SQLite, da wir alle im Kontext unserer Datenbankvorlesung damit in Berührung gekommen sind und somit jeder über alle Features des Systems informiert ist. SQLite ist eine sehr leichtgewichtige Datenbank, die für unseren Kontext und die Last, die das System in der Testung und Kontext der Arbeit ausgesetzt ist, auf jeden Fall ausreicht. Für den Fall, dass der Throughput zur Datenbank höher wird und das Projekt live genommen wird, kann die Datenbank auch problemlos auf ein schwergewichtigeres System wie PostgreSQL umgestellt werden, da alle Features des leichtgewichtigen SQLite auch in komplexeren Systemen verwendet werden können. Somit ist die Datenbank leicht austauschbar und kann mit nur wenigen Änderungen der Treiber und Adresse im Code angepasst werden.

2.3 Features

Im Folgenden werden Features in die jeweiligen Geltungsbereiche unterteilt:

- **Generelle Features:**
  - **Produktübersicht:** Sobald die App aufgerufen wird, ist eine Liste aller Produkte zu sehen. Diese werden dynamisch aus der Datenbank im Backend geladen und im Frontend optisch aufbereitet. Man kann den Preis, die Gesamtbewertung, das Produktbild und den Namen sehen. Ein Produkt kann mit Hilfe eines Buttons auf der Produktkarte direkt in den Warenkorb gelegt werden.
  - **Einzelansicht:** Hier werden Details zu dem jeweiligen Produkt angezeigt. Eine kurze Beschreibung sowie die Verfügbarkeit werden als Zahlen angegeben. Außerdem kann man eine größere Anzahl an Produkten gleichzeitig in den Warenkorb hinzufügen.
  - **Bewertungen:** Bewertungen werden auf der Produktseite angezeigt, mit Bewertungstext und einem Rating von null bis fünf Sternen. Auf derselben Seite besteht auch die Möglichkeit, selbst Rezensionen für ein Produkt abzugeben. Dazu muss die Bewertung in Sternen ausgewählt und optional ein Text hinzugefügt werden. Durch einen Klick auf den Submit-Button wird die Bewertung veröffentlicht.
  - **Suche:** In der App-Bar gibt es eine Suchleiste. Mittels Auto-Complete werden vorhandene Produktnamen als Vorschläge angezeigt, um dem Nutzer das Suchen zu erleichtern, wenn er die genaue Bezeichnung nicht kennt.
  - **Filter:** Wir haben drei Filter unserem Shop hinzugefügt. Diese können bei Bedarf schnell und einfach erweitert werden, da ein Filter-Template vorhanden ist. Derzeit existiert ein Preisfilter in Form eines Sliders, ein Bewertungsfilter in Form von Sternen und einen Kategorienfilter, da alle unsere Produkte in Kategorien unterteilt sind. Dadurch können beispielsweise technische und pflanzliche Produkte unterschieden werden. Außerdem haben wir Standardkategorien hinzugefügt, die wir als sinnvoll erachtet haben. Im Nachhinein können beleibeig viele Kategorien ergänzt werden.
  - **Datenschutzerklärung:** Aus rechtlichen Gründen muss ein Impressum und eine Datenschutzerklärung auf einer Verkaufswebseite im geschäftlichen Kontext vorhanden sein. Diese ist wie gefordert von jeder Seite des Shops über die App-Bar am oberen Bildschirmrand erreichbar. Impressum und Datenschutzerklärung befinden sich auf derselben Seite. Bisher ist lediglich eine Dummy-Datenschutzerklärung eingefügt, diese muss vor einer Inbetriebnahme noch rechtlich überprüft werden.
  - **Theming:** Auf unserer Webseite sind verschiedene Themes implementiert. Die Farbe der Seitenelemente sind als Variable beschrieben. Durch das Wechseln zwischen Light- und Darkmode über den Button in der App-Bar wird diese Variable umgeschrieben und die Elemente nehmen die Farbe des jeweiligen Themes an.

Das Hinzufügen von weiteren Themes ist möglich, auch wenn die beiden implementierten Themes im normalen Betrieb ausreichen.

- **Registrierung:** Um ein Benutzerkonto zu erstellen muss der Nutzer auf den Avatar in der App-Bar klicken und danach im Menü “Einloggen” wählen. In dem nun aufkommenden Fenster kann “Registrieren” angeklickt werden, wodurch nach Eingabe der erforderlichen Daten und dem Bestätigen ein neues Nutzerkonto angelegt wird. Die angegebenen Passwörter werden im Backend verschlüsselt und mit einem Salt in der Datenbank abgelegt, wodurch eine hohe Sicherheit gegen Angriffe gewährleistet ist.
- **Login:** Nach Anlegen eines Nutzerkontos kann man sich über das gleiche Fenster anmelden. Nach Anmeldung sind die zusätzlichen Pages “Order” und “Account” verfügbar, da diese auf Nutzerdaten zugreifen und dementsprechend erst nach dem Login verfügbar sind. Außerdem ist bei entsprechender Berechtigung der Reiter “Admin” in der App-Bar verfügbar sein. Die Login-Informationen werden mit Hilfe eines Cookies für 7 Tage gespeichert. Somit bleibt man nach dem Schließen der Webseite weiterhin eingeloggt.
- **Sprachen:** Wir haben unsere App mit Hilfe von i18n internationalisiert. Es werden die Sprachen Englisch und Deutsch unterstützt. Durch die Abfrage der Browsersprache kann direkt beim App-Start gewehrleistet werden, dass die richtige Sprache für den jeweilige Benutzer eingerichtet ist. Wenn die Sprache nicht in der Liste der verfügbaren Sprachen liegt, wird auf die Fallbacksprache Englisch gewechselt. Es können weitere Sprachen mit Hilfe einer Übersetzungsdatei im JSON-Format und dem Einfügen des Sprachcodes in der i18n-Konfiguration hinzugefügt werden.
- **Personendaten:** Nachdem der User eingeloggt ist, wird diese Seite freigeschaltet. Wenn der Nutzer sich auf dieser Seite ausloggt, wird auf die Homepage zurücknavigiert. Personendaten stellen bei uns ausschließlich die Adresse und den Namen des Nutzers dar. Dies ist das benötigte Minimum, um eine Bestellung an den Nutzer versenden zu können.
- **Farming-Station:** Für eine bessere Nutzererfahrung haben wir eine Visualisierung der Farming-Station in unser Projekt integriert, in Form einer Seite, die eine Übersichtsgrafik enthält. Neben dieser Grafik erfolgt eine Auflistung aller benötigten Produkte, teils als Sets gegliedert (z. B. Schlauch-Set). Wenn der Nutzer die Maus über die Produkte bewegt, erscheint auf dem Übersichtsbild der Farming-Station eine Markierung an der entsprechenden Stelle. Somit erhält der Nutzer Unterstützung bei Aufbau der Station. Das Ganze wird technisch realisiert, indem das Bild der Farming-Station durch ein dem Produkt zugeordnetes Bild ersetzt wird. Das neue Bild unterscheidet sich von dem Default-Bild nur durch die Markierung.

- **Userspezifische Features:**

- **Check-out:** Nach dem Hinzufügen eines Produktes in den Warenkorb kann die Check-out-Page aufgerufen werden. Hier bekommt der Nutzer eine Gesamtübersicht aller Produkte im Warenkorb mit Preis und Anzahl, sowie der Gesamtsumme. Beim Wechsel zur nächsten Seite erfolgt die Eingabe der Versanddaten. Wenn der Nutzer eingeloggt ist, werden die Daten mit den im Account gespeicherten Daten vorausgefüllt, können aber bei Bedarf geändert werden. Danach gelangt man zur Auswahl der Zahlungsart, was bei uns natürlich nicht implementiert ist. Wenn alle Schritte erfolgreich erledigt wurden, kann der Nutzer auf “Kostenpflichtig Bestellen” klicken, wodurch seine Bestellnummer angezeigt wird. Die Bestellung wird damit auch im Backend hinterlegt.
- **Warenkorb:** Der Warenkorb kann mit oder ohne Login genutzt werden. Alle Produkte können hier hinzugefügt oder wieder entfernt werden. Der Warenkorb wird mit Hilfe eines Cookies 7 Tage lang gespeichert, sodass dieser auch nach Schließen der Seite noch vorhanden ist.
- **Orders:** Der User kann hier alle seine getätigten Bestellungen sehen. Diese sind mit Hilfe von verschiedenen Tabs auf der Seite in “Laufende” und “Vergangene” unterteilt. Laufende Bestellungen können nach dem Anklicken in der Detailansicht mit Hilfe eines Buttons storniert werden. Sobald die Bestellung einen anderen Status bekommt, beispielsweise “Storniert”, wird dieser dem Nutzer angezeigt. Außerdem können die Produkte der vergangenen Bestellungen gesehen werden, damit man diese nachbestellen kann.

- **Adminspezifische Features:**
  - **Absicherung:** Alle Admin-Pages sind nur für den Admin-User erreichbar. Falls ein unautorisierter User trotzdem auf den Pfad gelangt, bekommt dieser nichts angezeigt, da die Daten vom Backend nur mit Angabe der Anmeldedaten herausgegeben werden. Wenn diese keinem Admin-User zugehörig sind, erfolgt keine Ausgabe, wodurch alle Admin-Abfragen extra abgesichert sind. Daher besteht für den normalen Nutzer keine Möglichkeit, in irgendeinem Wege an adminspezifische Daten zu gelangen.
  - **Statistiken:** Sobald der Nutzer mit Admin-Berechtigung die Admin-Page aufruft, kann er diverse Statistiken beobachten. Wir haben die Statistiken so gewählt, dass diese nützlich für Administration sind. Weitere Statistiken können mit Hilfe des Backends, welches diese Daten aufbereitet, nachträglich erstellt werden. Der genaue Aufbau der Statistik-Seite kann im Frontend live betrachtet werden und wird hier nicht weiter erläutert.
  - **Userdaten:** Der Admin hat die Möglichkeit Userdaten einzusehen. Diesem werden E-Mail und Adressdaten des Nutzers angezeigt, wobei die Adresse auch geändert werden kann. Außerdem ist es möglich ein Nutzerkonto zu löschen. Auch eine Beförderung vom Standarduser zum Admin ist durch eine Checkbox möglich.

- **Produktdaten:** Durch eine Produktdarstellung in Form einer Tabelle kann der Admin übersichtlich die Produkte des Shops verwalten. Es können alle Produktinformationen geändert werden (Änderungen der Bewertung ist nicht möglich, da diese nur durch die Nutzer geändert werden sollen). Die Bewertung und der aktuelle Füllstand zum angedachten Füllstand werden farblich bewertet, sodass Engpässe oder Qualitätsdefizite direkt ersichtlich sind
- **Orders:** Hier werden die Bestellungen aller Nutzer angezeigt. Diese sind in die jeweiligen Bestellkategorien wie “Storniert”, “Bestellt” oder “Versand” unterteilt. Alle Bestellungen liegen in einer Art Kanban-Board vor und können über Drag-and-Drop in eine andere Kategorie geschoben werden. Dies wird direkt im Backend gespeichert und ist sofort für den Nutzer, der die Bestellung aufgegeben hat, verfügbar.
- **Farming-Station-Verwaltung:** Die Verwaltung der Farming-Station-Page ist in der Admin-Produkt-Page integriert. Hier können Bilder zu den Produkten hochgeladen werden. Mit einem vorhandenen Bild wird das Produkt direkt auf der Farming-Station-Page dementsprechend aufgeführt.

3 Dokumentation

3.1 Standard-Account

Da der erste Admin-Account in der Datenbank hinzugefügt werden muss, haben wir diesen bereits angelegt. Alle weiteren Accounts können von diesem zum Admin befördert werden. Die Login-Daten sind:

- Benutzername: pascal.stoffels@htwsaar.de
- Password: ps287

3.2 Installieren der Abhängigkeiten

3.2.1 Backend

Zur Ausführung des Backend-Codes muss Java auf dem PC installiert sein. Unser Projekt verwendet Java 17. Es ist also ratsam diese Version zu verwenden und zusätzlich openjdk in Version 17 zu installieren. Des Weiteren muss Maven installiert sein, da dies als Building-Tool verwendet wird. Alternativ kann auch eine IDE wie IntelliJ verwendet werden, da hier die benötigten Abhängigkeiten vorab installiert sind und somit keine Probleme verursachen. Nach der Installation sollte ein Neustart des PCs erfolgen.

3.2.2 Frontend

NodeJs muss zwingend auf dem PC installiert sein, da dies als Building-Tool für das Frontend verwendet wird. Es soll also in der Konsole der Befehl ***npm*** verfügbar sein. Beim ersten Start oder Änderungen im Projekt sollte im Ordner des Frontends immer ein ***npm i*** ausgeführt werden, damit alle Dependencies installiert werden. Nach der Installation wird ein Neustart des PCs empfohlen.

3.3 Starten des Projekts

Das Starten des Projekts kann manuell oder mit Hilfe des von uns bereitgestellten Skripts erfolgen. Bei der manuellen Vorgehensweise muss der Code im Ordner ***/00-backend*** mit ***mvn package*** in eine jar-Datei umgewandelt werden. Danach kann das Backend mit ***java -jar ./target/webshop*.jar*** gestartet werden. In einer weiteren Konsole kann im Ordner ***/01-frontend*** mit Hilfe von ***npm start*** das Frontend gestartet werden. Wenn es zu Fehlern bezüglich Dependencies kommt, weil einige nicht installiert sind, sollte ein ***npm i*** Abhilfe schaffen.

Außerdem kann das bereitgestellte Skript verwendet werden, um das Projekt zu starten. Hierbei genügt es, die Datei ***start.sh*** unter Linux/Mac beziehungsweise ***start.ps1*** unter Windows mit entsprechenden Rechten auf dem PC auszuführen, wodurch die Applikation Back- sowie Frontend automatisch hochfährt.

Das Backend ist nun unter der URL localhost:8085 verfügbar. Das Frontend läuft auf localhost:5173.

3.4 Dokumentation des Codes

Der Code ist ausführlich an relevanten Stellen kommentiert, sodass Änderungen, Erweiterungen und Anpassungen auch ohne tiefgreifende Kenntnisse über den Aufbau der App durchgeführt werden können.

Insbesondere die Ports und verschiedenen Frontend-Pfade sind kommentiert und somit übersichtlich gestaltet, wodurch Änderungen schnell und einfach möglich sind.