SHOPLET

Software/Anwendungprojekt

Sommersemester 2025

MI7 / MMB

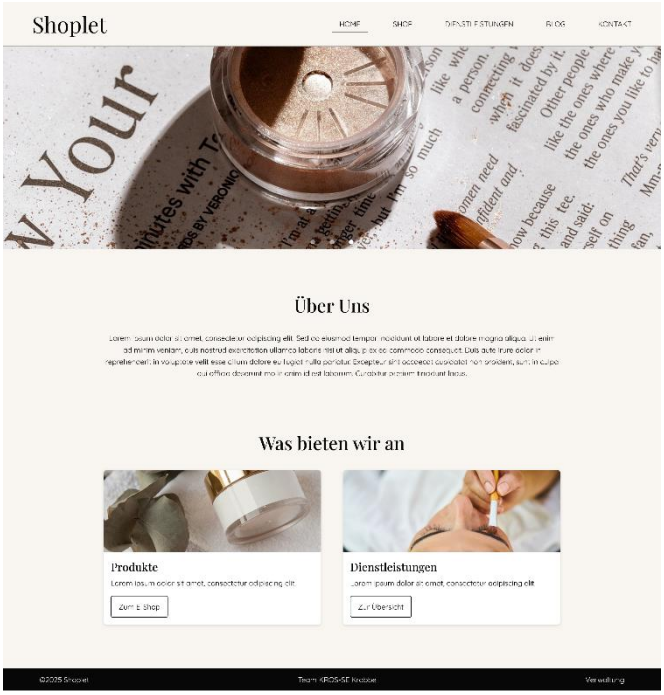
Stepan Vyroubal, sv059@hdm-stuttgart.de

Oliver Seher, os052@hdm-stuttgart.de

Katja Gehringer, kg074@hdm-stuttgart.de

Rahel Hafner, rh092@hdm-stuttgart.de

https://gitlab.mi.hdm-stuttgart.de/shoplet



1. Kurzbeschreibung

Shoplet ermöglicht kleinen Unternehmen, schnell und unkompliziert einen eigenen Webshop aufzubauen. Produkte und Dienstleistungen lassen sich professionell präsentieren und direkt online verkaufen – ganz ohne technisches Vorwissen.
Es gibt eine benutzerfreundliche Ansicht für Kund:innen, über die sie bequem einkaufen können, sowie einen Administrationsbereich für Betreiber:innen. Dort können Inhalte verwaltet und Produkte hinzugefügt oder entfernt werden.

2. Startklasse

Das Projekt muss im Backend über Docker compose up –Build gestartet werden.

Im Frontend auch über Docker compose up –Build.

(Mehr Info im Readme)

3. Besonderheiten

Wir haben zum Planen Figma genutzt hier ist auch unsere Architektur zu finden.
https://www.figma.com/board/OqZ6T1dQuWjPIomwOGa3Sm/Ideen-Board?node-id=0-1&t=jkbYcAvyV7YYsoaq-1

Es gibt ein Repo für Frontend und eines für Backend

https://gitlab.mi.hdm-stuttgart.de/shoplet/frontend

https://gitlab.mi.hdm-stuttgart.de/shoplet/backend

Die wichtigsten und externen Sachen sind im Backend abgelegt wie zum Beispiel die Dokumentation.

Alles ist in der Adminansicht anpassbar, allerdings muss man im Backend einmalig die Daten vom Bank Account, … dort hinterlegen.
Diese ist unter resources -> application.yml zu finden. Dort muss man dann nur den Inhalt der vorhandenen Felder ändern.

Wir haben mit Dockern gearbeitet. Man kann diesen Docker compose up –Build starten

Und mit Docker compose down beenden.

Admin, Home und Kontakt werden über einen Initializer erstellt, wenn es keines der 3 Objekte gibt. (Damit beim ersten Start bestimmte Objekte initialisiert werden und diese dann als singleton verwendet werden)

Man kann ein Kontaktformular ausfüllen und abschicken, welches dann an den Betreiber per Mail gesendet wird. Das kann genutzt werden um einen Termin für Dienstleistungen zu vereinbaren.

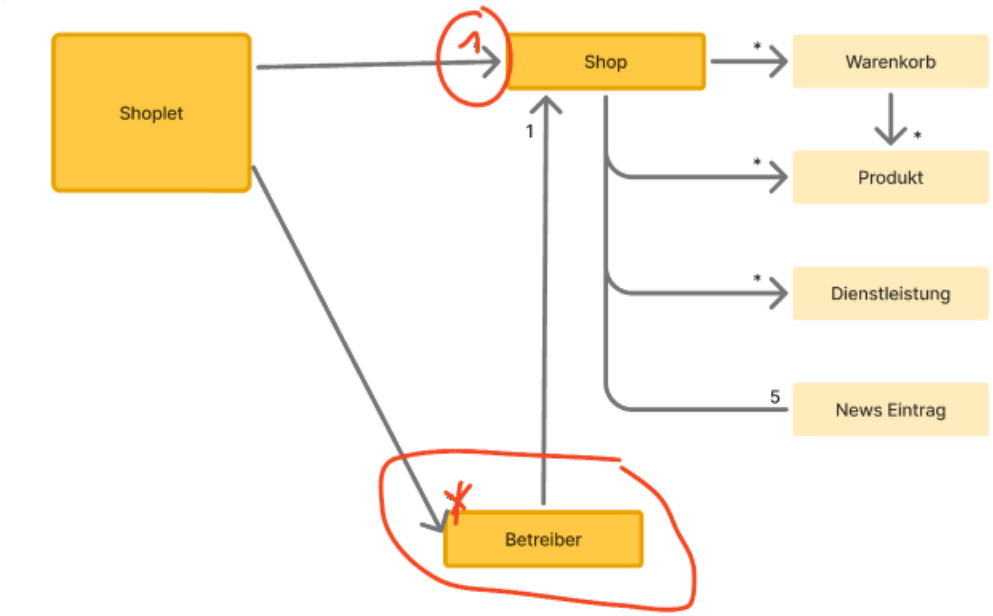
Es ist möglich Produkte im Shop kaufen und dann auf Rechnung bezahlen. Wir haben uns gegen Paypal entschieden, da es für kleine Unternehmen meist zu teuer ist und es so gar keine Verwendung finden würde.

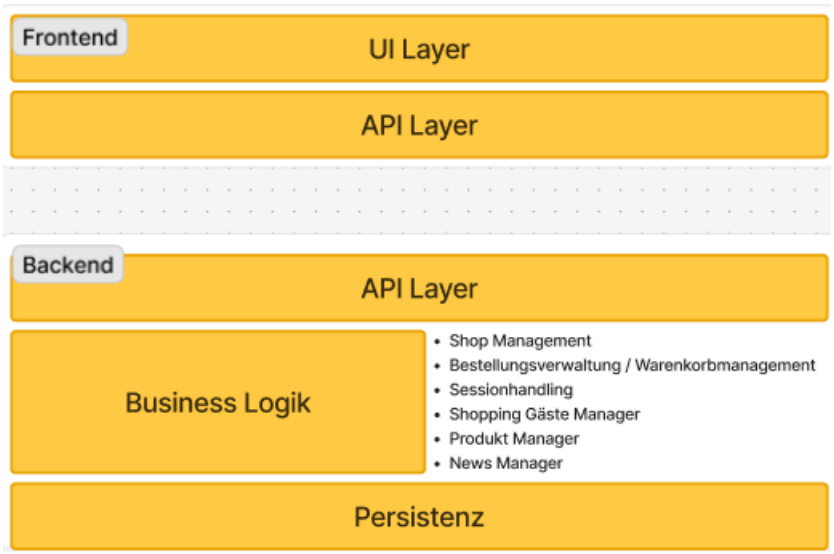
Der Admin hat eine komplett eigene Ansicht in der er die Website nach seinen Wünschen gestalten kann. Er muss sich ausloggen um die Ansicht des Nutzers zu haben und um die Seite wie dieser nutzen zu können.

Wir arbeiten mit Session Handling in Bezug auf den Warenkorb. Es gibt keine Anmeldung für den User, allerdings wird der Warenkorb pro Session gespeichert, sodass der Nutzer immer seinen Warenkorb behält, solange er in der gleichen Session ist.

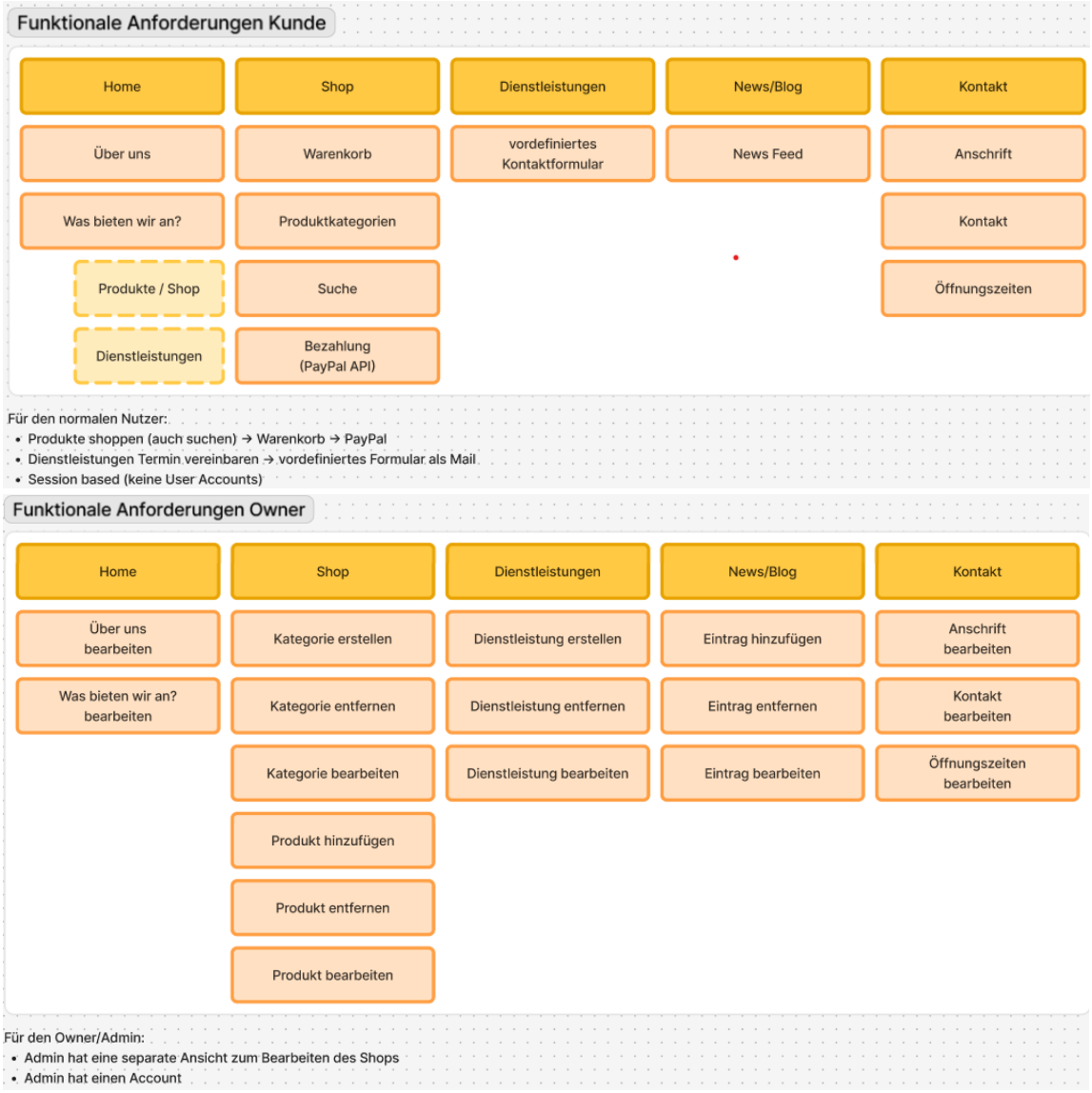
Außerdem Nutzen wir es zur Identifikation des Users / Admin um diese zu unterscheiden und die Ansicht und Rechte klarzustellen.

4. Datenmodell





5. Unsere Anforderungen



Während dem Projekt haben wir uns gegen PayPal und für Rechnungen entschieden, da uns auch durch den Austausch mit kleineren Unternehmern klar geworden ist, dass Paypal durch die hohen Kosten sehr unattraktiv für Kleinunternehmer ist.

6. Umsetzung der gegebenen Anforderungen

1. Anforderungsanalyse

1.1. Meilensteine

Die folgenden Meilensteine definieren wichtige Phasen und Aspekte der Projektentwicklung

Frontend:

Design: Festlegung des visuellen und interaktiven Konzepts der Anwendung.

Schnittstelle: Definition und Implementierung der Kommunikationspunkte zwischen Frontend und Backend.

Bewertungskriterien: Eine Auflistung der Kriterien zur Bewertung des Projektes, um eine bessere Übersicht zu gewährleisten.

Backend:

Base-Structure: Erstellen der Grundlegenden Struktur des Backends.

Session Handling: Umsetzung des Session Handlings.

Paypal/API: Einbinden von Paypal (haben wir uns im lauf des Projekts dagegen entschieden)

Frontend /Backend Paths: Umstellen des Backends damit es zum Frontend passt.

Configuration and Session Handling: Session Handling in Verbindung mit dem Frontend zum laufen bringen. Auch bezogen auf funktionierende Cookies,…

1.2. Labels

Zur Organisation und Nachverfolgung von Aufgaben und Problemen werden folgende Labels verwendet

Frontend:

bug: Zur Kennzeichnung und Behebung von Fehlern.

critical: Signalisiert Aufgaben oder Probleme, die sofortige Aufmerksamkeit erfordern.

enhancement: Dient zur Markierung von neuen Funktionen und Verbesserungen.

documentation: Wird für die Erstellung und Pflege von Dokumentation verwendet.

criteria: Zur Kennzeichnung spezifischer Kriterien für die Übersichtlichkeit.

Backend:

bug: zur Kennzeichnung von Fehlern.

documentation: zum Hinzufügen von Dokumentationen.

feature: wenn Dinge implementiert wurden die keine Voraussetzung waren.

fix: wenn Probleme behoben wurden.

progress: stellt Fortschritt im Projekt dar.

request: wenn etwas im Projekt noch umgesetzt werden muss/soll.

2. Softwarearchitektur & Clean Code

Die Architektur der Anwendung ist modular aufgebaut und folgt den Prinzipien des Clean Code, um Wartbarkeit und Skalierbarkeit zu gewährleisten.

Frontend:

Beinhaltet Konfigurations- und Ignorierdateien sowie die notwendigen Dependencies für die Client-Seite der Anwendung.

public: Dieser Ordner enthält statische Assets wie Beispielbilder, Fonts und Icons.

src: Hier befindet sich die index.html (sowie die zugehörigen .ts- und .css-Dateien), die die Grundlage der gesamten Anwendung bildet.

app: Der Hauptanwendungsordner ist weiter unterteilt in:

guards: Module, die URL-Pfade vor unbefugtem Zugriff autorisierter oder unautorisierter Benutzer schützen.

interceptors: Komponente, die jedem HTTP-Request die notwendige Konfiguration und Backend-URL automatisch anhängt.

pages: Enthält die Hauptseiten der Anwendung:

pages-admin: Hauptseiten, die ausschließlich für Administratoren zugänglich sind.

pages-normal: Hauptseiten für reguläre Benutzer.

pages-universal: Weitere allgemeine Seiten und wiederverwendbare Komponenten.

services: Module, die die Verbindung zum Backend herstellen und HTTP-Anfragen kapseln.

environments: Beinhaltet die spezifische URL zum Backend.

Backend:

Controller: Beinhalten die Controller, welche Endpunkte bereitstellen.

Models: Beinhalten Entities, die Objekte definieren.

Repo: Ein JPA Repository für die jeweilige Entity.

Security: Beinhaltet die Security Konfigurationen.

Service: Enthält die PDF-Generierung.

Session: Verwaltet das Session Handling.

2.1. Ablauf der Kommunikation

Die Kommunikation beginnt beim Nutzer und führt über die page.html zur page.ts und anschließend zur page.service.ts. Von dort wird ein HTTP-Request gesendet, der vom Interceptor modifiziert wird. Die modifizierte Anfrage geht an das Backend, dessen Antwort zurück an die page.service.ts und dann an die page.ts geleitet wird, bevor sie schließlich auf der page.html angezeigt wird.

3. Git

3.1. Vorgehen und Branching

Anfänglich versuchten wir im Frontend, den Gitflow mit einer Master- und einer Develop-Branch zu implementieren. Dieses Vorgehen erwies sich jedoch mitten im Projekt als zu aufwendig, da unser Projekt nicht die Größe aufweist, die einen solchen Workflow rechtfertigen würde. Daher entschieden wir uns, zum "normalen" Feature-Branch-Workflow zurückzukehren, welches auch im Backend genutzt wurde.

Größere Features wurden im Frontend in separaten Branches entwickelt. Bugfixes oder Ähnliches wurden jedoch direkt auf der Master-Branch durchgeführt. Wir haben uns somit nicht auf einen festen Workflow festgelegt, sondern uns stattdessen intern abgestimmt, wer wann welche Aufgaben auf welcher Branch bearbeiten sollte.

4. Tests & Build Management

Wir haben im Backend Tests nach dem F.I.R.S.T Prinzip umgesetzt, für Entities und Controller. Integrationstests haben wir für die Repos (Schnittstelle zur Datenbank) geschrieben.

5. CI/CD

Wir haben eine CI/CD-Pipeline in GitLab implementiert. Es gibt insgesamt 3 stages: build, test und package. Hierbei werden nur die ersten beiden immer ausgeführt, der package Job muss manuell ausgeführt werden.
Damit werden Artefakte nur dann erstellt und bereitgestellt, wenn sie explizit benötigt werden und es werden Build- Speicherressourcen gespart.

6. Parallele Programmierung

Hat unser Projekt nicht, da wir uns dazu entschieden haben UI und Schnittstellen umzusetzen.

7. UI

Die Benutzeroberfläche wurde mit Angular v19 umgesetzt. Der von Angular bereitgestellte HttpClient verarbeitet HTTP-Anfragen standardmäßig asynchron.

8. Schnittstellen

Wir haben CRUD umgesetzt, die Security Sachen laufen durch das Spring Framework automatisch asynchron. Alles andere läuft synchron da wir keine Threads nutzen.

9. Persistenz

Wir haben eine Datenbank, welche Daten speichert und welche sich selbst neustartet falls Fehler auftreten.

10. Security-Maßnahmen (geplant und umgesetzt, projektabhängig)

Das Frontend läuft in einem Docker-Container auf Port 4200:80 (Details siehe docker-compose.yml) inklusive der zugehörigen Dockerfile. Die Nginx-Konfiguration ist ebenfalls auf Port 4200:80 eingestellt. Das Standard-Routing des Nginx-Servers wurde auf die Angular-Routen umgeleitet, um zu verhindern, dass Nginx selbst nach Ressourcen sucht und die korrekte Weiterleitung an die Angular-Anwendung sicherzustellen.

Das Backend nutzt verschiedene Sicherheitsmechanismen, um Anwendung und Daten zu schützen:

Authentifizierung und Autorisierung

Einsatz von Spring Security zur Benutzerverwaltung.

Passwörter werden sicher mit BCrypt verschlüsselt.

Rollenbasierter Zugriff: Bestimmte Bereiche wie /admin/** sind nur für Admins zugänglich.

Ein Custom UserDetailsService lädt Benutzerinformationen aus der Datenbank.

CSRF-Schutz

Schutz vor Cross-Site Request Forgery durch CSRF-Token via CookieCsrfTokenRepository.

Nach der Anmeldung wird automatisch ein neues Token erstellt.

Ein eigener Erfolgshandler sorgt für korrektes Setzen des Tokens beim Login.

Sitzungsverwaltung

Ein SessionListener überwacht Sitzungsstart und -ende.

Bei Abmeldung werden Sitzung und Cookies (z. B. JSESSIONID, XSRF-TOKEN) gelöscht.

Ressourcen wie Warenkörbe werden beim Sitzungsende automatisch bereinigt.

CORS-Konfiguration

Zugriff ist nur von vertrauenswürdigen Ursprüngen erlaubt (z. B. http://localhost:4200/).

Zulässige Methoden: GET, POST, PUT, DELETE, OPTIONS.

Logging

Sicherheitsrelevante Ereignisse (z. B. Login, Logout, Sitzungen) werden protokolliert.

Fehler werden erfasst, um mögliche Sicherheitsprobleme zu erkennen.

12. Reflexion

Früher besser im Team abstimmen um am Ende stress zu vermeiden.

7. Stellungnahme:

Wir haben uns letztendlich dazu entschieden Paypal nicht zu integrieren.
Unser Template soll für kleine Unternehmen oder sogar Einzelpersonen nutzbar sein. Gerade für Kleinunternehmer ist Paypal meistens zu teuer und rentiert sich nicht. Deshalb sind wir von Paypal auf Rechnung umgestiegen.