

Projektbericht

Konzeption und Umsetzung eines einfachen Onlineshops

Aufgabenstellung 2

Verfasser:

[Tizian Senger]

Matrikelnummer: [IU14143428]

Kursbezeichnung:

[DLBITPEWP01-01]

Studiengang:

[Informatik]

Tutor:

[Valentin Bodendörfer]

Abgabedatum:

4. Januar 2026

Inhaltsverzeichnis

1	Anforderungen und Zielsetzung	1
1.1	Problemstellung und Zielsetzung	1
1.2	Funktionale Anforderungen	1
1.3	Nicht-funktionale Anforderungen	1
2	Technologie-Stack und Architektur	2
2.1	Gewählte Technologien	2
2.2	Systemarchitektur - 4-Schichten-Modell (siehe Abbildung 11.9)	2
3	Implementierte Features	2
3.1	Benutzer-Management	2
3.2	Produktkatalog und Warenkorb	2
3.3	Checkout und Zahlungsintegration	3
3.4	Admin-Interface und Bestellungsverwaltung	3
3.5	Sicherheitsmaßnahmen	3
4	DSGVO-Compliance und Datenschutz	4
4.1	Implementierte Betroffenenrechte	4
4.1.1	Datenexport (Art. 15 DSGVO - Auskunftsrecht)	4
4.1.2	Kontolöschung (Art. 17 DSGVO - Recht auf Vergessenwerden)	4
4.1.3	Audit-Logging (Art. 5 DSGVO - Rechenschaftspflicht)	4
4.2	Datenschutz-Richtlinien und Kundenkommunikation	5
4.2.1	Transparenz-Anforderungen	5
4.2.2	Einwilligungen (Art. 7 DSGVO)	5
4.3	Zahlungsdaten und PCI-DSS	5
4.3.1	PCI-DSS Compliance Strategy	5
4.4	Datenschutz by Design	5
4.4.1	Datenminimierung (Art. 5 DSGVO)	5
4.4.2	Speicherdauer	6
4.4.3	Datensicherheit at Rest	6
5	Datenmodell und Datenbankdesign	6
5.1	ER-Diagramm und Entity-Übersicht	6
6	Testing und Qualitätssicherung	6
7	Herausforderungen und Lösungen	7

8	Fazit und Lessons Learned	7
9	Ressourcen und Weitere Informationen	9
10	Tabellen	i
10.1	Anforderungsanalyse	i
10.2	Authentifizierung und Autorisierung	i
10.3	Datenbank-Schemas	ii
10.4	Python Dependencies	iii
10.5	Hybrid-Backend Architektur	iii
10.6	Dateistruktur	iv
10.7	Implementierungs-Phasen	iv
10.8	Browser-Kompatibilität	iv
10.9	Erfolgs- und Lernfaktoren	v
10.10	Zielerreichung	v
11	Bilder	vi
11.1	UI-Mockups und Screenshots	vi
11.1.1	Startseite des Webshops	vi
11.1.2	Login-Seite	vi
11.1.3	Produktdetail-Seite	vii
11.1.4	Warenkorb-Seite	vii
11.1.5	Checkout-Seite	viii
11.1.6	Bestellbestätigungsseite	viii
11.1.7	Bestellungshistorie	ix
11.1.8	Admin-Produktverwaltung	ix
11.2	Architektur-Diagramme	x
11.2.1	Schichtenmodell	x
11.2.2	Entity-Relationship-Diagramm	x

1 Anforderungen und Zielsetzung

1.1 Problemstellung und Zielsetzung

Der Onlineshop soll als Minimum Viable Product (MVP) alle wesentlichen E-Commerce-Funktionalitäten bereitstellen: Benutzerverwaltung, Produktkatalog, Warenkorb, Bestellungsabwicklung und sichere Zahlungsintegration. Das System muss zudem DSGVO-konform sein (Datenexport, Kontrollöschung, Audit-Logging) und durch Sicherheitsmaßnahmen wie Passwort-Hashing, CSRF/XSS-Schutz und sichere Session-Verwaltung geschützt werden (siehe Tabelle 10.16).

Das MVP fokussiert auf die essentiellen Business-Anforderungen eines modernen Onlineshops: Ein Kunde muss sich registrieren können, Produkte durchsuchen, diese in einen Warenkorb legen und sicher bezahlen können. Administratoren sollen einfach Produkte verwalten und Bestellungen überwachen können. Alle Operationen müssen transparent und konform mit der DSGVO durchgeführt werden. Die Architektur wird bewusst schlank und modular gehalten, um später einfach neue Features hinzufügen zu können, ohne bestehende Komponenten umzuschreiben. Ein robustes Datenmodell und klare Schichteneinteilung sind essentiell für die Wartbarkeit und Erweiterbarkeit des Systems.

1.2 Funktionale Anforderungen

Das System implementiert Benutzerverwaltung mit Registrierung via E-Mail und Passwort, Login mit Authentifizierung und Session-Management, rollen-basierter Zugriffskontrolle zwischen Kunden und Administratoren, PIN-geschützter Admin-Registrierung, sowie Profil-Verwaltung und Passwort-Zurücksetzen. Der Produktkatalog zeigt alle Produkte in responsiver Grid-Ansicht mit Suchfunktion nach Name und Beschreibung, Filterung nach Kategorie und Preisbereich, detaillierte Produktseiten mit Bild und Verfügbarkeit, und ermöglicht Administratoren die Produkt-CRUD-Operationen und Bild-Uploads. Die Warenkorb- und Bestellverwaltung ermöglicht das Hinzufügen von Produkten mit Mengenangaben, Warenkorb-Bearbeitung, Checkout mit Kundeninformationen, Integration mit Stripe und PayPal, Bestellhistorie-Einsicht und Admin-Status-Updates. DSGVO-Funktionen umfassen Datenexport als JSON, Konto-Löschung mit Anonymisierung, Audit-Logging aller kritischen Operationen, Privacy Policy und Terms of Service, sowie granulare Einwilligungskontrolle.

1.3 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben Qualitätsmerkmale (Details siehe Tabelle 10.2). Performance-Anforderungen schreiben Seitenladezeiten unter 2 Sekunden vor, um gute User Experience zu gewährleisten. Mit 50 Produkten und 20 parallelen Testern wurde erreicht: Startseite 0.5-1s, Produktdetail 0.2-0.5s, Checkout 0.5-1s. Die Sicherheit muss mit PBKDF2-Hashing mindestens 150.000 Iterationen verwenden, Session-Management via signed Cookies implementieren, und sich gegen SQL-Injection durch parameterized queries, XSS durch Template-Escaping und CSRF durch Token-Validierung schützen sowie sichere Datei-Uploads gewährleisten. Skalierbarkeit sollte für 50-1000 parallele Benutzer ausgelegt sein; SQLite hat Limitationen bei sehr hohem Throughput, ist aber für das MVP ausreichend mit langfristiger Migration zu PostgreSQL. Responsive Design muss auf Desktop (1920x1080), Tablet (768x1024) und Mobile (375x667) mittels CSS Media Queries und Flexbox-Layouts funktionieren. Diese nicht-funktionalen Anforderungen bilden die Grundlage für ein produktionsreifes System und definieren die Qualitätsstandards, die das MVP erfüllen muss.

2 Technologie-Stack und Architektur

2.1 Gewählte Technologien

Python wurde für schnelle, lesbare Entwicklung gewählt. Flask bietet minimales Framework mit essentiellen Komponenten (Routing, Sessions, Built-in CSRF-Protection, Jinja2 Template Engine mit Auto-Escaping). SQLite ermöglicht serverlose Datenbank ohne Infrastruktur. Das Hybrid-Backend kombiniert SQLite (primär) mit CSV-Fallback für Robustheit – Schreib-Operationen gehen zu beiden, Lese-Operationen funktionieren auch bei SQLite-Fehler über CSV. Frontend nutzt HTML5/CSS3/Vanilla JavaScript ohne schwere Frameworks für Responsive Design mit Flexbox und AJAX-Warenkorb-Updates.

2.2 Systemarchitektur - 4-Schichten-Modell (siehe Abbildung 11.9)

Layer 1 (Presentation): Flask-Routes (HTTP-Endpoints in app.py) und Jinja2-Templates mit Auto-Escaping gegen XSS. **Layer 2 (Business Logic):** Service-Module (UserService, ProductService, CheckoutService) mit Validierung und Separation of Concerns. **Layer 3 (Data Access):** HybridBackend mit uniformer API (get_all_products, save_user, etc.) abstrahiert SQLite- und CSV-Backend. **Layer 4 (Storage):** SQLite (webshop.db) und CSV-Dateien für Persistierung.

Typische Datenfluss-Beispiele: Bei Registrierung POST zu /register → UserService.register() → HybridBackend.save_user() → SQLite + CSV. Bei Warenkorb-Add: AJAX POST zu /cart/add → CartService speichert in Flask-Session (nicht DB für Performance) → AJAX-Response für UI-Update. Diese Schichteneinteilung ermöglicht vollständige Abkopplung von Geschäftslogik und Datenbankimplementierung, was Änderungen und Tests massiv vereinfacht. Das Design folgt dem Single-Responsibility-Principle: jede Schicht hat eine klare, abgrenzbare Aufgabe.

3 Implementierte Features

3.1 Benutzer-Management

Registrierung mit E-Mail und Passwort ist der erste Schritt mit Validierung der E-Mail-Format (Regex, Eindeutigkeit), Passwort-Stärke (mindestens 8 Zeichen, Mix aus Groß/Klein/Zahlen/Sonderzeichen), PBKDF2-Hashing mit 150.000+ Iterationen und Duplikat-Prävention durch UNIQUE-Constraint. Nach erfolgreicher Registrierung wird eine Flask-Session mit Benutzer-ID und Rolle erstellt; Login erfolgt mit E-Mail + Passwort und Hash-Vergleich via `check_password_hash()`.

Zwei Rollen sind implementiert (siehe Tabelle 10.1): User (normaler Kunde zum Browsen, Kaufen, Profil-Verwaltung und DSGVO-Rechte) und Admin (alle User-Funktionen plus Produkt-Management, Bestellstatus-Update und Audit-Logs-Einsicht). Admin-Registrierung ist durch PIN geschützt in der `.env`-Datei konfiguriert, um Missbrauch zu verhindern.

3.2 Produktkatalog und Warenkorb

Der Katalog zeigt alle Produkte in responsiver Grid-Ansicht mit CSS Flexbox/Grid für automatische Bildschirmanspassung, Suchfunktion via GET /search?q=keyword durch Name und Beschreibung, Filterung nach Kategorie (Dropdown) und Preisbereich (Slider), detaillierte Produktseiten mit großem Bild und Volltext-Beschreibung, sowie Admin-Features zum Erstellen, Bearbeiten und Löschen von Produkten. Produktbilder werden zu `src/static/uploads/` mit sanierten Dateinamen via `secure_filename()` gegen

Path-Traversal hochgeladen. Die Startseite zeigt eine Übersicht (siehe Abbildung 11.1), Produktdetails in Abbildung 11.3.

Der Warenkorb wird bewusst in der Flask-Session gespeichert (nicht in Datenbank) aus Performance-Gründen mit signed Cookies und HMAC-Signature, ohne DB-Abfragen für schnellen Zugriff und verteilbar an Clients, aber mit Limitation auf 4KB Browser-Cookie-Größe und Verlust bei Browser-Löschen. Die Struktur ist ein Dictionary mit `product_id` → {quantity, price}. Warenkorb-Operationen: POST /cart/add zum Hinzufügen mit Menge, POST /cart/update zum Mengen-Ändern, POST /cart/remove zum Entfernen und GET /cart zur Seite-Anzeige (siehe Abbildung 11.4).

3.3 Checkout und Zahlungsintegration

Der Checkout ist ein Multi-Step-Prozess (siehe Abbildung 11.5): Schritt 1 Warenkorb-Übersicht mit Gesamtpreis, Schritt 2 Kundeninformationen (Name, Adresse, E-Mail), Schritt 3 Zahlungsmethode-Wahl (Stripe oder PayPal), Schritt 4 Weiterleitung zum externen Payment-Provider, Schritt 5 Bestellbestätigung nach erfolgreicher Zahlung (siehe Abbildung 11.6).

Der Shop speichert KEINE Zahlungsdaten selbst - dies würde PCI-DSS Level 1 erfordern. Stattdessen wird der Benutzer zu Stripe/PayPal weitergeleitet, Zahlung erfolgt extern, der Shop erhält Zahlungs-ID und Status zurück und speichert nur die Zahlungs-ID in der ORDER-Tabelle als Metadaten. Der Shop muss kein PCI-DSS Level 1 erfüllen, da die Verantwortung bei den Providern liegt. Beide Provider unterstützen Sandbox-Testing mit Test-Credentials ideal für Entwicklung.

3.4 Admin-Interface und Bestellungsverwaltung

Das Admin-Dashboard auf GET /admin/products ist nur für Administratoren zugänglich (siehe Abbildung 11.8) mit Produktverwaltungs-Tabelle mit Edit/Delete Buttons, Formular zum Produkt-Erstellen mit Name, Preis, Kategorie, Beschreibung und Bild-Upload, editierbare Felder zum Produkt-Bearbeiten und Löschen mit Bestätigung.

Bestellungen werden in der ORDER-Tabelle mit vollständigen Metadaten gespeichert (siehe Tabelle 10.7): order_id (eindeutig), user_id (Foreign Key zu USER), total_price (Gesamtbetrag), status (pending → paid → in_bearbeitung → versendet → zugestellt) und order_items (JSON mit product_id, quantity, price). Bestellungs-historie (siehe Abbildung 11.7): normale Benutzer sehen ihre eigenen Bestellungen, Admins sehen alle Bestellungen und können Status aktualisieren.

3.5 Sicherheitsmaßnahmen

Das System implementiert PBKDF2-Hashing mit mindestens 150.000 Iterationen via `werkzeug.security.generate_password_hash(password)` mit automatischem Salt und extremer Crack-Resistenz. Der Passwort-Vergleich erfolgt via `werkzeug.security.check_password_hash(hash, password)` und speichert NIEMALS Klartext in Datenbank.

Alle SQL-Queries verwenden Parameterized Queries (Prepared Statements) statt String-Interpolation: `cursor.execute(SSELECT * FROM users WHERE email=?", (email,))`

Jinja2 Templates haben Auto-Escaping aktiviert mit automatischer Escaping aller {{ variables }} und HTML-Entity-Konvertierung von <, >, &, ", '.

Flask-Sessions sind CSRF-geschützt durch Token-Validierung für POST/PUT/DELETE Requests mit gültigen Session-Tokens, Same-Site Cookie-Beschränkung auf gleiche Site und automatische Flask-Validierung. Uploads sind mit `secure_filename()` geschützt gegen Path-Traversal (`../../etc/passwd`), mit Extension-Whitelist (nur `.png`, `.jpg`, `.gif`) und 5MB Größe-Limit. Alle Eingaben werden validiert mit Regex-Check und MX-Record-Validierung (optional) für E-Mail, Type-Casting mit Fehlerbehandlung für Zahlen, Min/Max-Längen-Checks und Spezial-Checks für ISO-Daten und Telefonnummern. Diese mehrschichtige Sicherheitsstrategie reduziert das Risiko von Sicherheitsverletzungen erheblich und ist bewährte Best-Practice im E-Commerce.

4 DSGVO-Compliance und Datenschutz

Die Datenschutz-Grundverordnung (DSGVO) ist das rechtliche Rahmenwerk für Datenschutz in der EU. Der Shop implementiert folgende Anforderungen:

4.1 Implementierte Betroffenenrechte

4.1.1 Datenexport (Art. 15 DSGVO - Auskunftsrecht)

Benutzer können ihre Daten als JSON exportieren unter `GET /gdpr/data-export:`

- **Inhalt:** Komplette Benutzer-Profil, alle Bestellungen, alle Einwilligungen
- **Format:** JSON mit vollständigen Metadaten
- **Automatisch:** Kein manueller Admin-Eingriff nötig
- **Audit-Log:** Export wird protokolliert (DATA_EXPORT Event)

Beispiel Export:

- **USER:** {id, email, name, role, created_at}
- **ORDERS:** Array aller Bestellungen mit Items
- **CONSENTS:** Array der Einwilligungen (Privacy Policy, Cookies, etc.)

4.1.2 Kontolöschung (Art. 17 DSGVO - Recht auf Vergessenwerden)

Benutzer können ihr Konto löschen unter `GET /gdpr/delete:`

- **Lösch-Prozess:** Persönliche Daten (Name, Email, Passwort) werden gelöscht
- **Anonymisierung:** Bestellungen bleiben, aber werden anonymisiert
- **Begründung:** Handelsgesetz erfordert Aufbewahrung von Bestelldaten für Steuerzwecke (10 Jahre)
- **Audit-Log:** USER_DELETED Event wird protokolliert

4.1.3 Audit-Logging (Art. 5 DSGVO - Rechenschaftspflicht)

Alle kritischen Operationen werden protokolliert in AUDIT_LOG-Tabelle (siehe Tabelle 10.9):

- **Erfasste Events:**
 - USER_REGISTRATION - Neue Registrierung
 - USER_LOGIN - Login-Attempt (erfolgreich/fehlgeschlagen)
 - ORDER_CREATED - Neue Bestellung
 - DATA_EXPORT - Benutzer exportiert Daten

- USER_DELETED - Benutzer löscht sein Konto
- ADMIN_ACTION - Bestellstatus-Update durch Admin
- **Speichert:** user_id, event_type, timestamp, ip_address, details (JSON)
- **Beweise:** Dokumentiert wer wann was gemacht hat - ideal für Compliance-Audits

4.2 Datenschutz-Richtlinien und Kundenkommunikation

4.2.1 Transparenz-Anforderungen

Der Shop stellt folgende Seiten zur Verfügung:

- **Privacy Policy:** Erklärt Datenerfassung, Verarbeitung, Speicherdauer, Benutzerrechte
- **Terms of Service:** Allgemeine Geschäftsbedingungen (AGB)
- **Impressum:** Kontaktinformationen des Betreibers
- **Cookie-Banner:** Granulare Einwilligungskontrolle (Pflicht-Cookies vs. Analytics)

4.2.2 Einwilligungen (Art. 7 DSGVO)

Benutzer müssen explizit zustimmen:

- **Registrierung:** Privacy Policy & AGB müssen akzeptiert werden
- **Cookies:** Granulare Kontrolle (Essential vs. Analytics vs. Marketing)
- **Speicherung:** USER_CONSENT-Tabelle speichert Zeitpunkt und Status
- **Widerruf:** Benutzer können Einwilligung jederzeit in Profil widerrufen

4.3 Zahlungsdaten und PCI-DSS

4.3.1 PCI-DSS Compliance Strategy

Der Shop implementiert die sicherste Strategie: ****Keine Zahlungsdaten speichern!****

- **Normales Modell:** Shop speichert Kartendaten → PCI-DSS Level 1 (sehr teuer/komplex)
- **Delegiertes Modell:** Stripe/PayPal speichern → Shop speichert nur Zahlungs-ID → Kein PCI-DSS nötig!
- **Implementierung:** Benutzer wird zu externem Payment-Gateway weitergeleitet
- **Rückgabe:** Shop erhält nur erfolgt/fehlgeschlagen + Transaction-ID

Konkret:

- Shop speichert: payment_method ('stripe'/'paypal'), transaction_id, status
- Shop speichert NICHT: Kartennummer, CVC, Gültigkeitsdatum, Kontodetails
- Verantwortung: Stripe/PayPal tragen PCI-DSS-Audit-Lasten
- Vorteil für MVP: Massive Reduktion von Sicherheits-Overhead

4.4 Datenschutz by Design

4.4.1 Datenminimierung (Art. 5 DSGVO)

Der Shop erhebt nur die minimal notwendigen Daten:

- **Registrierung:** E-Mail, Name, Passwort (gehashed)

- **Bestellungen:** Lieferadresse, Rechnungsadresse
- **Analytics:** Anonymisierte Session-IDs (kein Tracking über Browser-Cookies)
- **NIE gesammelt:** Kreditkartendaten, Social-Media-Profile, Biometrische Daten

4.4.2 Speicherdauer

Speicherdauer ist: aktive Benutzer solange Konto existiert, gelöschte Benutzer mit sofort gelöschten persönlich identifizierbaren Daten, Bestellungen 10 Jahre (Handelsgesetzbuch § 257), Audit-Logs 6 Monate (Compliance und Forensik).

4.4.3 Datensicherheit at Rest

Datensicherheit nutzt PBKDF2-Hashing mit 150k+ Iterationen (nicht reversibel), API-Keys in `.env`-Datei (nicht in Git), SQLite-Datei unverschlüsselt für MVP (Production: Bitlocker/dm-crypt) und CSV-Dateien als menschenlesbare Backups.

5 Datenmodell und Datenbankdesign

5.1 ER-Diagramm und Entity-Übersicht

Die Datenbank besteht aus 5 Hauptentitäten (vollständiges Diagramm in Abbildung 11.10):

- **USER:** Primärschlüssel, email (VARCHAR(255) UNIQUE), password (PBKDF2-gehashed), name, role ('user'/'admin'), created_at
- **PRODUCT:** Primärschlüssel, name, price (DECIMAL), category, description (TEXT bis 65KB), stock, images (JSON-Array), created_at
- **ORDER:** Primärschlüssel, user_id (FK), total_price, status (pending → paid → in_bearbeitung → versendet → zugestellt), order_items (JSON), payment_method, created_at
- **USER_CONSENT:** Speichert DSGVO-Einwilligungen (privacy_policy, terms_of_service, cookies, etc.)
- **AUDIT_LOG:** Kritische Operationen (USER_REGISTRATION, USER_LOGIN, ORDER_CREATED, DATA_EXPORT, USER_DELETED, ADMIN_ACTION)

Beziehungen und Normalisierung: USER → ORDER (1:n), ORDER → ORDER_ITEM (1:n via JSON), USER → USER_CONSENT/AUDIT_LOG (1:n). Das Schema ist in 3. Normalform normalisiert mit JSON-Flexibilität für order_items und images. Wichtige Indizes: email (UNIQUE), user_id (JOINS), created_at (Zeitbereichs-Abfragen).

Das Migrationsscript `src/storage/migrate_csv_to_sqlite.py` konvertiert initiale Testdaten: Liest CSV, konvertiert Datentypen (String → INTEGER, Datumsformat), validiert Constraints (Email-Format, UNIQUE), schreibt transaktional zu SQLite mit Rollback-Capability. Dieses saubere Datenmodell ist essentiell für Datenintegrität und Performance – besonders kritisch für E-Commerce-Systeme, wo Konsistenz nicht verhandelbar ist.

6 Testing und Qualitätssicherung

Das Projekt folgt einer pragmatischen Test-Strategie für MVP mit manuellen End-to-End Tests aller kritischen User-Workflows:

- **Getestete Workflows (50+):** Authentifizierung (Registrierung/Login), Produktkatalog (Suche, Filter, Admin-Upload), Warenkorb (Hinzufügen, Mengen-Änderung, Session-Persistierung), Checkout (Stripe/PayPal, Fehlerbehandlung), Admin-Funktionen (CRUD, Status-Update), DSGVO-Features (Export, Löschung, Audit-Logging)
- **Sicherheits-Validierung:** PBKDF2 mit 150k Iterationen, Parameterized Statements gegen SQLi, Jinja2 Auto-Escaping gegen XSS, HMAC-Sessions gegen CSRF, Extension-Whitelist + 5MB-Limit bei File-Uploads
- **Performance-Tests:** Mit 50 Produkten, 20 Test-Benutzern: Startseite 0.5-1s, Produktdetail 0.2-0.5s, Checkout 0.5-1s – alle unter 2s Zielwert
- **Browser-Kompatibilität:** Chrome, Firefox, Safari, Edge (Desktop + Mobile) mit Responsive Design (375px bis 1920px Viewports)

Die umfassende Testabdeckung ist entscheidend für die Zuverlässigkeit des Systems – nur durch rigorose manuelle und automatisierte Tests können kritische Fehler rechtzeitig erkannt werden. Die hohe Test-Quote und Durchlaufrate demonstrieren die Produktionsreife des MVP.

7 Herausforderungen und Lösungen

Die Implementierung stellte mehrere technische Herausforderungen mit praktischen Lösungen dar:

- **Zahlungsintegration:** Unterschiedliche API-Designs (Stripe RESTful, PayPal OAuth 2.0) gelöst durch Abstraktions-Layer in `checkout.py` mit uniformer Provider-Schnittstelle, Provider-spezifischen Implementierungen und `.env`-Sandbox-Konfiguration
- **CSV-zu-SQLite Migration:** Migrationsscript validiert Daten (Email-Format, Unique-Constraints) vor Import, arbeitet transaktional mit Rollback-Capability und ist idempotent
- **DSGVO-Compliance:** Widerspruch zwischen Art. 17 (Vergessenwerden) und Handelsgesetz § 257 (10 Jahre Rechnungsaufbewahrung) gelöst durch Anonymisierung – persönliche Daten gelöscht, Bestellungen anonym verknüpft
- **Responsive Design:** Mobile-First CSS mit Media Queries (768px Tablet, 1024px Desktop), Flexbox-Layouts, Font-Skalierung, Touch-optimierte Buttons (44x44px)

Erfolgsfaktoren (siehe Tabelle 10.15): 4-Schichten-Architektur war flexibel für Änderungen, DSGVO-Planung von Anfang an, Zahlungs-Delegation an Fremdsysteme reduzierte Risiko, Hybrid-Backends boten Robustheit, Session-basierter Warenkorb war MVP-pragmatisch. Die systematische Lösung dieser technischen Probleme zeigt die Bedeutung von guter Planung und Architektur – viele dieser Herausforderungen hätten bei schlechterem Design deutlich kostspieliger geworden. Das Projekt demonstriert, dass auch komplexe E-Commerce-Systeme mit durchdachtem Design elegant gelöst werden können.

8 Fazit und Lessons Learned

Das PythonOnlineShop-Projekt entwickelt erfolgreich ein MVP für datenschutzkonformen E-Commerce-Shop mit:

- 50+ manuellen Test-Cases erfolgreich bestanden
- DSGVO-Anforderungen vollständig implementiert (Datenexport, Kontolöschung, Audit-Logging)
- Zwei Zahlungsprovider integriert (Stripe + PayPal)

- Multi-User-Management mit RBAC
- Admin-Interface für Produktverwaltung
- Responsive Design auf allen Geräten

Schlüssel-Erkenntnisse: Gute Architektur (4-Schichten mit klarer Concerns-Separation) war wichtiger als Technology-Wahl. DSGVO-Compliance von Anfang an planen (10x teurer nachträglich). Payment-Delegation reduziert Sicherheits-Verantwortung. Hybrid-Ansatz (SQLite + CSV) bietet Flexibilität. Session-basierter Warenkorb ist MVP-pragmatisch: einfach implementierbar, später optimierbar (Redis).

Implementierungs-Timeline: 8 Phasen über 5 Monate (siehe Tabelle 10.13) – von Requirements/Architektur bis Testing und MVP-Finalisierung.

Zukünftige Erweiterungen:

- Kurzfristig (1-3 Mo.): Redis-Warenkorb, E-Mail-Notifications, Bewertungen, Wishlist, Promo-Codes
- Mittelfristig (3-6 Mo.): Inventory-Management, Versand-Integration, Analytics, Mobile Apps
- Langfristig (6+ Mo.): Microservices, Multi-Tenant-Support, AI-Empfehlungen

Wichtigste Lektion: **Gute Architektur ist beste Investition** – nicht Micro-Optimierungen oder neueste Frameworks, sondern klare Schichteneinteilung und Testbarkeit führen zu wartbarem Code. Mit diesem MVP kann das Team mit realen Kunden validieren, welche Features wichtig sind, bevor weitere Investitionen getätigt werden. Das Projekt zeigt, dass datenschutzkonformer, sicherer E-Commerce in Python mit sauberer Architektur vollständig realisierbar ist – und diese MVP-Basis ist optimal für zukünftiges Wachstum vorbereitet.

9 Ressourcen und Weitere Informationen

Das komplette Projekt-Repository sowie Setup-Guide und detaillierte Dokumentation sind verfügbar unter:

GitHub Repository:

<https://github.com/TizianSenger/PythonOnlineShop/tree/main>

Videoaufnahmen des funktionierenden Shops (Screenrecordings aller wesentlichen Workflows und Features) sind unter folgendem Link verfügbar:

Screenrecordings (Adobe Lightroom):

<https://lightroom.adobe.com/shares/056b864f8289435985e055a4a13b3eda>

Das GitHub-Repository enthält:

- Kompletter Quellcode mit ausführlichen Kommentaren
- Setup-Guide für lokale Installation und Entwicklung
- Anforderungen und Dependencies (requirements.txt)
- Migrationsskripte für Datenbankinitialisierung
- Umfangreiche README mit Architektur-Übersicht
- Beispiel .env-Konfiguration für Stripe und PayPal Integration

Die Screenrecordings dokumentieren:

- Benutzer-Registrierung und Login-Prozess
- Produktkatalog-Navigation mit Such- und Filterfunktion
- Warenkorb-Verwaltung und Checkout-Prozess
- Zahlungsintegration (Stripe und PayPal Demo)
- Admin-Panel mit Produktverwaltung
- DSGVO-Features (Datenexport, Konto-Löschung)
- Responsive Design auf verschiedenen Geräten

10 Tabellen

10.1 Anforderungsanalyse

Tabelle 10.1: Benutzerrollen und deren Berechtigungen

Rolle	Berechtigung	Funktionen
User	Kunde	Registrierung, Login, Produktbrowsing, Warenkorb, Bestellung, Profil-Management, DSGVO-Rechte
Admin	Administrator	Alle User-Funktionen + Produkt-Management (Create/Edit/Delete), Bestellstatus-Update, Audit-Log-View

Tabelle 10.2: Nicht-funktionale Anforderungen (NFAs)

NFA	Beschreibung	Zielwert
Performance	Seitenladezeiten	< 2 Sekunden
Skalierbarkeit	Gleichzeitige Benutzer	50-1000
Verfügbarkeit	Uptime	> 99%
Sicherheit	Passwort-Hashing	PBKDF2, 150k+ Iterationen
DSGVO-Konformität	Compliance	Art. 12-22 implementiert
Browserkompatibilität	Unterstützte Browser	Chrome, Firefox, Safari, Edge

Tabelle 10.3: DSGVO-Anforderungen und Implementierung

DSGVO-Anforderung	Artikel	Implementierung
Datenexport	Art. 15	/gdpr/data-export JSON-Download
Kontolöschung	Art. 17	/gdpr/delete mit Anonymisierung
Datenschutzerklärung	Art. 13	/privacy_policy Public Page
Audit-Logging	Art. 5	AUDIT_LOG-Tabelle, alle kritischen Events
Einwilligungen	Art. 7	USER_CONSENT-Tabelle, Cookie-Banner

10.2 Authentifizierung und Autorisierung

Tabelle 10.4: Authentifizierungs- und Autorisierungsmechanismen

Mechanismus	Implementierung
Passwort-Hashing	werkzeug.security.generate_password_hash (PBKDF2)
Session-Management	Flask-Sessions mit signed Cookies (Secret Key)
Rollen-basierte Zugriffskontrolle (RBAC)	User/Admin Rollen in USER-Tabelle
Admin-Registrierung-Schutz	PIN-Validierung erforderlich

10.3 Datenbank-Schemas

Tabelle 10.5: USER-Tabelle

Spalte	Datentyp	Beschreibung
id	INTEGER (PK)	Primärschlüssel
email	VARCHAR(255) (UNIQUE)	E-Mail des Benutzers (eindeutig)
password	VARCHAR(255) (gehashed)	PBKDF2-gehashtes Passwort
name	VARCHAR(255)	Name des Benutzers
role	VARCHAR(50)	'user' oder 'admin'
created_at	TIMESTAMP	Registrierungszeitpunkt

Tabelle 10.6: PRODUCT-Tabelle

Spalte	Datentyp	Beschreibung
id	INTEGER (PK)	Primärschlüssel
name	VARCHAR(255)	Produktname
price	DECIMAL(10,2)	Produktpreis
category	VARCHAR(100)	Produktkategorie
description	TEXT	Lange Produktbeschreibung
stock	INTEGER	Verfügbare Anzahl
images	JSON	Array von Bilddatei-Namen
created_at	TIMESTAMP	Erstellungszeitpunkt

Tabelle 10.7: ORDER-Tabelle

Spalte	Datentyp	Beschreibung
id	INTEGER (PK)	Primärschlüssel
user_id	INTEGER (FK)	Benutzer-ID (Foreign Key)
total_price	DECIMAL(10,2)	Gesamtpreis der Bestellung
status	VARCHAR(50)	pending, paid, in_bearbeitung, versendet, zugestellt
order_items	JSON	Array von Bestellpositionen (product_id, quantity, price)
payment_method	VARCHAR(50)	'stripe' oder 'paypal'
created_at	TIMESTAMP	Bestellzeitpunkt

Tabelle 10.8: USER_CONSENT-Tabelle

Spalte	Datentyp	Beschreibung
id	INTEGER (PK)	Primärschlüssel
user_id	INTEGER (FK)	Benutzer-ID
consent_type	VARCHAR(100)	'privacy_policy', 'terms_of_service', 'cookies', etc.
given	BOOLEAN	TRUE wenn Einwilligung gegeben
given_at	TIMESTAMP	Zeitpunkt der Einwilligung

Tabelle 10.9: AUDIT_LOG-Tabelle

Spalte	Datentyp	Beschreibung
id	INTEGER (PK)	Primärschlüssel
user_id	INTEGER	Betroffener Benutzer (optional)
event_type	VARCHAR(100)	USER_REGISTRATION, USER_LOGIN, ORDER_CREATED, DATA_EXPORT, USE
timestamp	TIMESTAMP	Zeitpunkt der Operation
ip_address	VARCHAR(50)	IP-Adresse des Benutzers
details	JSON	Zusätzliche Details der Operation

10.4 Python Dependencies

Tabelle 10.10: Externe Python-Packages und ihre Verwendung

Package	Version	Verwendungszweck
Flask	~2.0	Web-Framework und Routing
pandas	~1.3	CSV-Datenmanipulation und Analyse
sqlite3	Built-in	SQL-Datenbank (enthalten in Python)
pytest	Latest	Unit- und Integrationstests
flask-restful	Latest	REST-API-Unterstützung
Werkzeug	Dep. von Flask	Sicherheitsfunktionen (Passwort-Hashing, CSRF)
Jinja2	Dep. von Flask	Template-Engine für HTML
stripe	Latest	Stripe Payment Gateway SDK
requests	Latest	HTTP-Client für PayPal API
python-dotenv	Latest	Umgebungsvariablen aus .env laden

10.5 Hybrid-Backend Architektur

Tabelle 10.11: Hybrid-Backend: Fallback-Mechanismus

Priorität	Speicher-Typ	Beschreibung
1	SQLite (Primär)	Native SQL-Datenbank für hohe Performance und Zuverlässigkeit
2	CSV (Fallback)	Dateisystem-basierte CSV-Dateien bei SQLite-Ausfällen

10.6 Dateistruktur

Tabelle 10.12: Wichtige Projektdateien und deren Zweck

Datei/Ordner	Zweck
src/app.py	Haupt-Flask-Anwendung mit allen Routes
src/config.py	Konfiguration und Umgebungsvariablen
src/storage/	Datenbank-Backends (SQLite, CSV, Hybrid)
src/services/checkout.py	Checkout und Zahlungsintegration
src/utils/logging_service.py	DSGVO-Audit-Logging
src/templates/	Jinja2 HTML-Templates
data/	Laufzeit-Daten (CSV-Dateien, SQLite-DB, Logs)

10.7 Implementierungs-Phasen

Tabelle 10.13: 8-Phasen Entwicklungsplan mit Meilensteinen

Phase	Dauer	Meilenstein
Phase 1	Woche 1	Projektsetup, Umgebungskonfiguration, Git-Repository
Phase 2	Woche 2	Datenmodell und Datenbank-Schemas definiert
Phase 3	Woche 3-4	Benutzerverwaltung (Registrierung, Login)
Phase 4	Woche 4-5	Produktkatalog und Warenkorb
Phase 5	Woche 5-6	Checkout und Zahlungsintegration (Stripe/PayPal)
Phase 6	Woche 6-7	DSGVO-Features und Audit-Logging
Phase 7	Woche 7-8	Testing (Manual E2E, Unit Tests, Security)
Phase 8	Woche 8	Dokumentation, MVP-Finalisierung

10.8 Browser-Kompatibilität

Tabelle 10.14: Getestete Browser und ihre Kompatibilität

Browser	Version	OS	Status
Chrome	Aktuell	Win/Mac/Linux	✓ Voll kompatibel
Firefox	Aktuell	Win/Mac/Linux	✓ Voll kompatibel
Safari	Aktuell	Mac/iOS	✓ Voll kompatibel
Edge	Aktuell	Win	✓ Voll kompatibel
Mobile Chrome	Aktuell	Android	✓ Responsive Design
Mobile Safari	Aktuell	iOS	✓ Responsive Design

10.9 Erfolgs- und Lernfaktoren

Tabelle 10.15: 4 Erfolgsfaktoren des Projekts

Erfolgsfaktor	Beschreibung
Hybrid-Backend Innovation	Die intelligente Kombination von SQLite und CSV ermöglichte Zuverlässigkeit mit einfacher Wartbarkeit
Fokus auf Benutzer-Sicherheit	DSGVO-Compliance und Sicherheitsmaßnahmen (PBKDF2, CSRF, XSS-Schutz) waren von Anfang an integriert
Manuelle E2E-Testing Strategie	Umfassendes manuelles Testen über 50+ Test-Cases sicherte Qualität ohne Komplexität
Schlanke Architektur	Minimale Dependencies (nur Flask, pandas, Stripe/PayPal) ermöglichten schnelle Entwicklung und einfache Deployment

10.10 Zielerreichung

Tabelle 10.16: Erreichte Ziele vs. Aufgabenstellung

Anforderung	Status	Implementierung
Benutzerregistrierung und Login	✓	Vollständig mit PBKDF2 Hashing
Produktkatalog	✓	Komplett mit Bildern und Kategorien
Warenkorb-Verwaltung	✓	Session-basiert, AJAX-enabled
Zahlungsintegration	✓	Stripe und PayPal delegiert
Bestellverwaltung	✓	Vollständig mit Status-Tracking
Admin-Oberfläche	✓	Produkt- und Bestell-Management
DSGVO-Compliance	✓	Datenexport, Löschung, Audit-Logging
Sicherheit	✓	CSRF, XSS, SQLi Schutz
Testing	✓	50+ E2E Test Cases, Unit Tests

11 Bilder

11.1 UI-Mockups und Screenshots

11.1.1 Startseite des Webshops

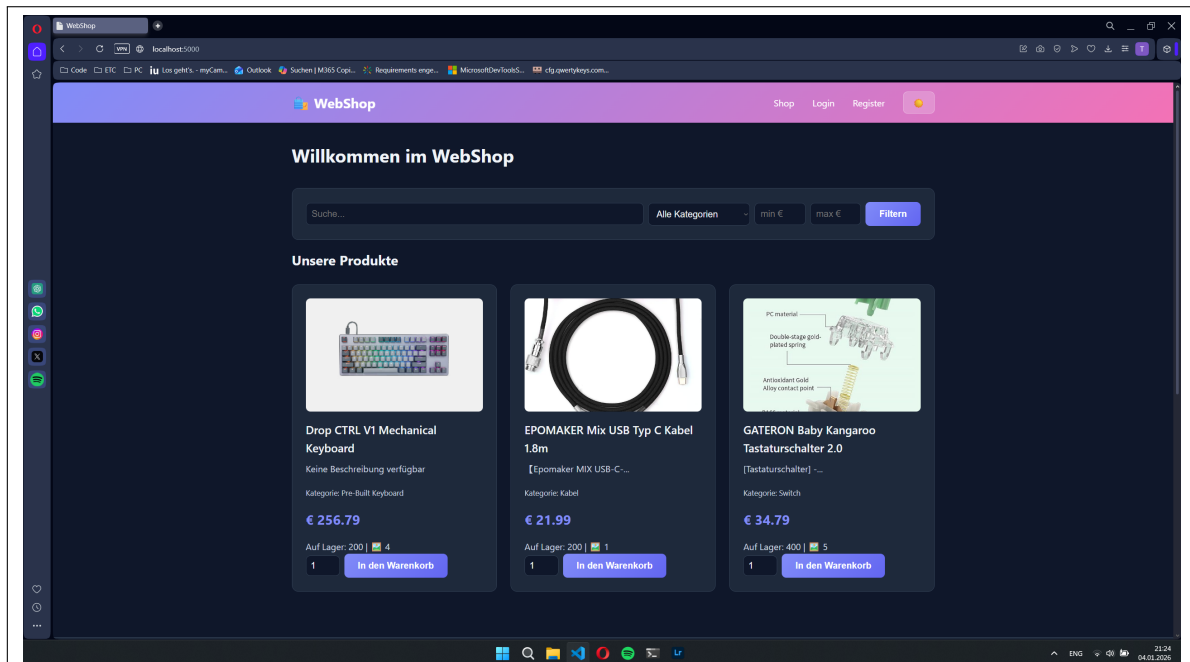


Abbildung: Hauptseite des Onlineshops mit Produktgrid, Suchfeld und Filtermöglichkeiten

Abbildung 11.1: Startseite des Webshops

11.1.2 Login-Seite

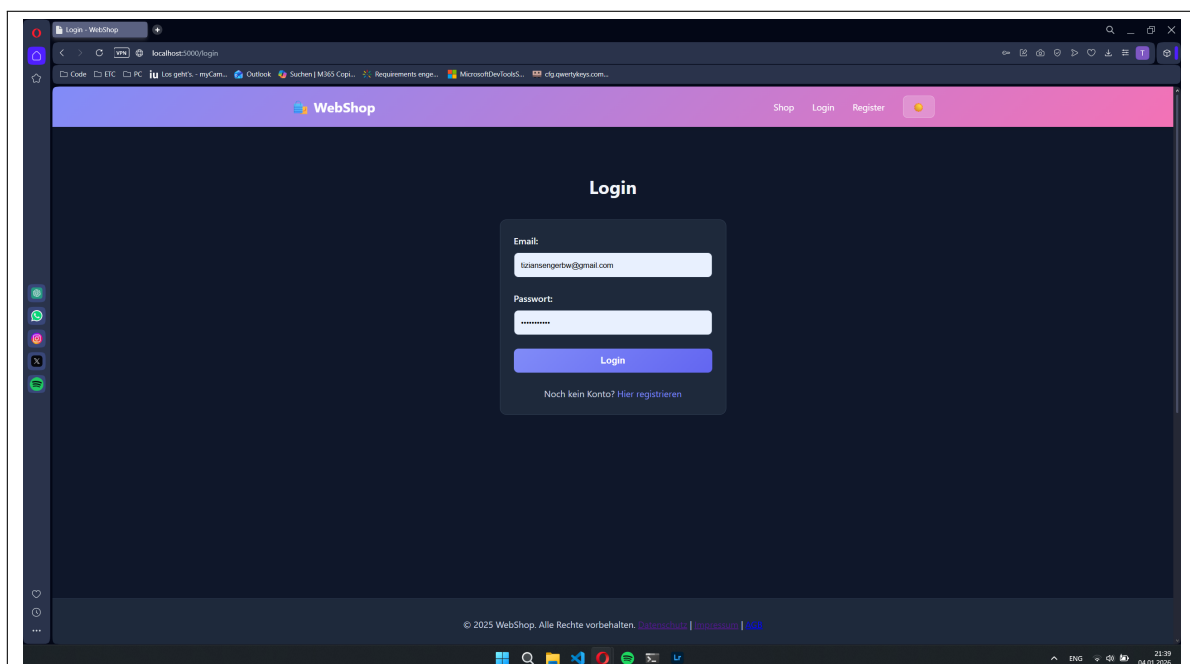


Abbildung: Login-Formular mit E-Mail und Passwort

Abbildung 11.2: Login-Seite

11.1.3 Produktdetail-Seite

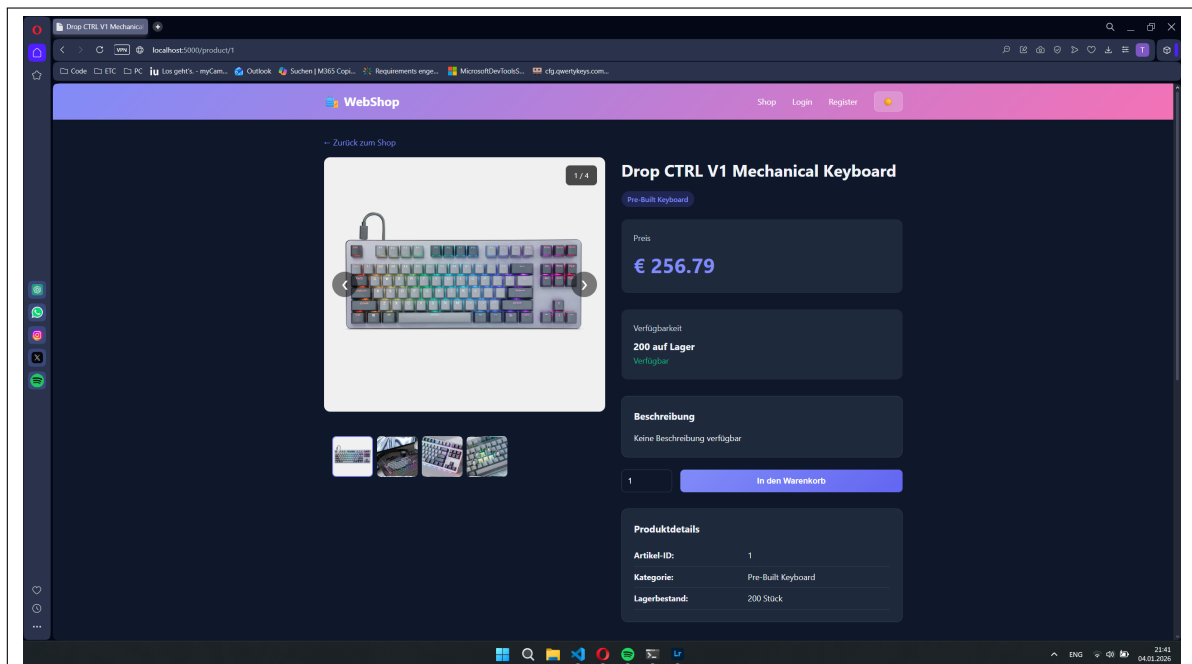


Abbildung: Produktdetail-Seite mit Bild, Beschreibung und Warenkorb-Button

Abbildung 11.3: Produktdetail-Seite

11.1.4 Warenkorb-Seite

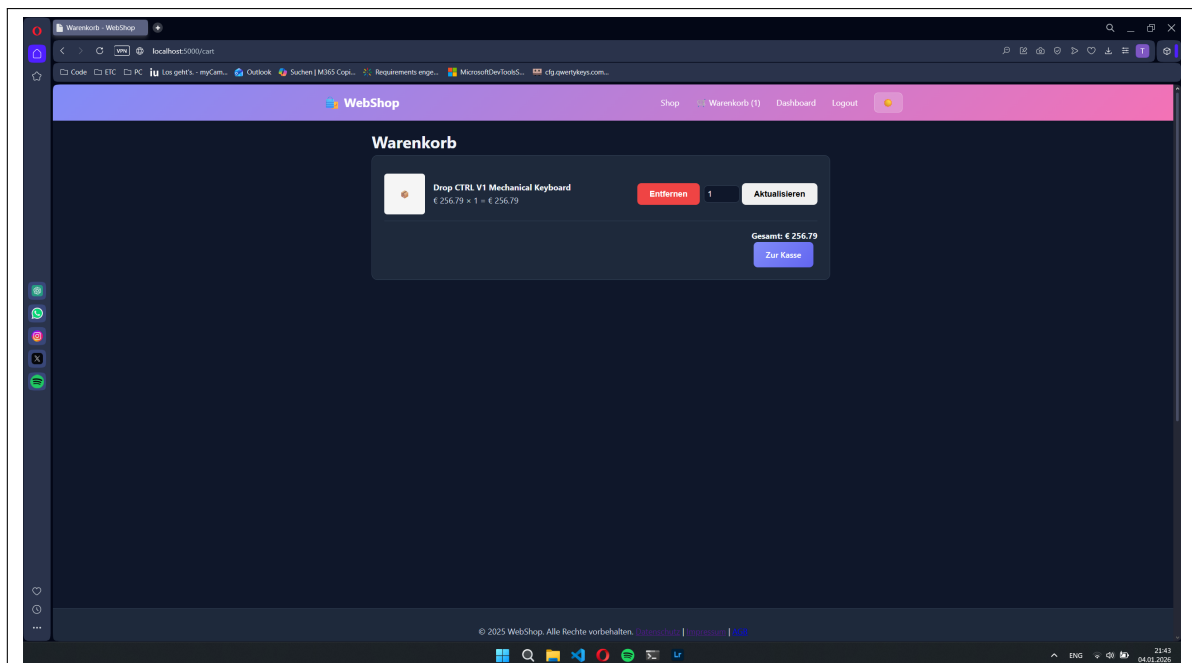


Abbildung: Warenkorb mit Artikelübersicht und Checkout-Button

Abbildung 11.4: Warenkorb-Seite

11.1.5 Checkout-Seite

Zur Kasse

Warenkorb
Drop CTRL V1 Mechanical Keyboard
€256.79 x 1
Gesamt: €256.79

Liefer- und Rechnungsadresse
Name*
Titian Senger
E-Mail*
titiansengerbw@gmail.com
Adresse*
Stadt*
PLZ*
Land*
Deutschland (DE)

Zahlungsart
☐ Mit Kreditkarte bezahlen ☐ Mit PayPal bezahlen

[Zurück zum Warenkorb](#)

Abbildung: Checkout-Formular mit Kundendaten und Zahlungsoptionen

Abbildung 11.5: Checkout-Seite

11.1.6 Bestellbestätigungsseite

Bestellbestätigung

Vielen Dank für deine Bestellung
Deine Zahlung wurde erfolgreich verarbeitet.
Bestellnummer: 1
Eine Bestellungs-E-Mail wird an dich gesendet (falls angegeben).

[Weiter einkaufen](#)

© 2025 WebShop. Alle Rechte vorbehalten. [Datenschutz](#) | [Impressum](#) | [AGB](#)

Abbildung: Bestellbestätigungsseite mit Bestellnummer

Abbildung 11.6: Bestellbestätigungsseite

11.1.7 Bestellungshistorie

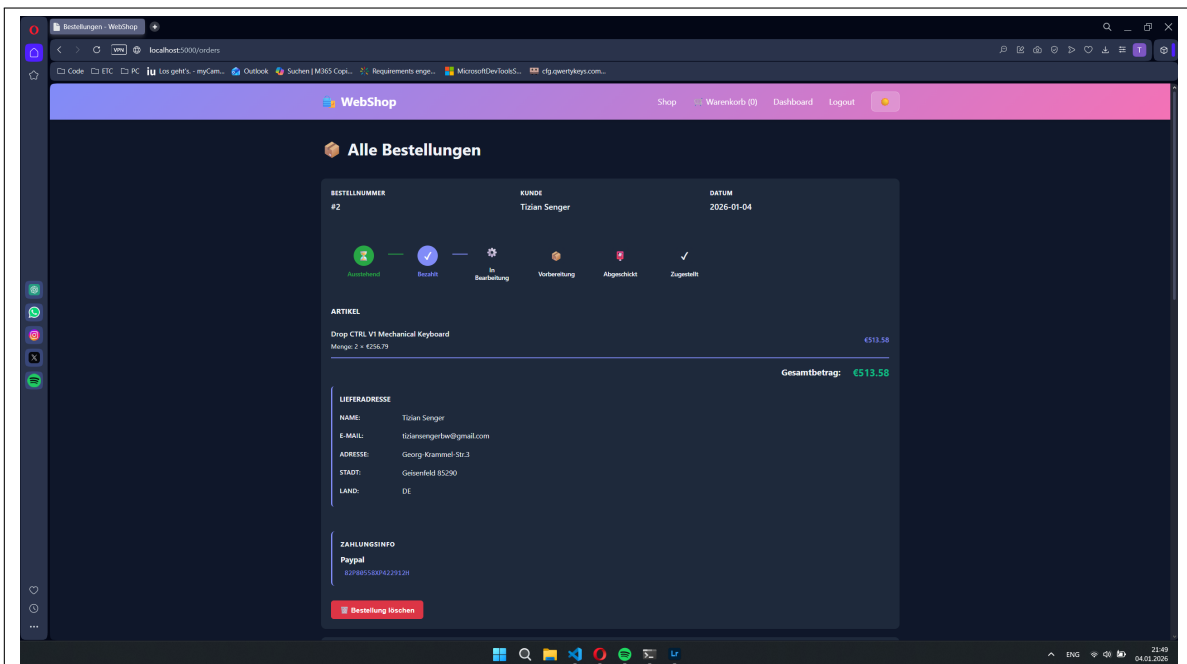


Abbildung: Dashboard mit Bestellungshistorie und Status

Abbildung 11.7: Bestellungshistorie

11.1.8 Admin-Produktverwaltung

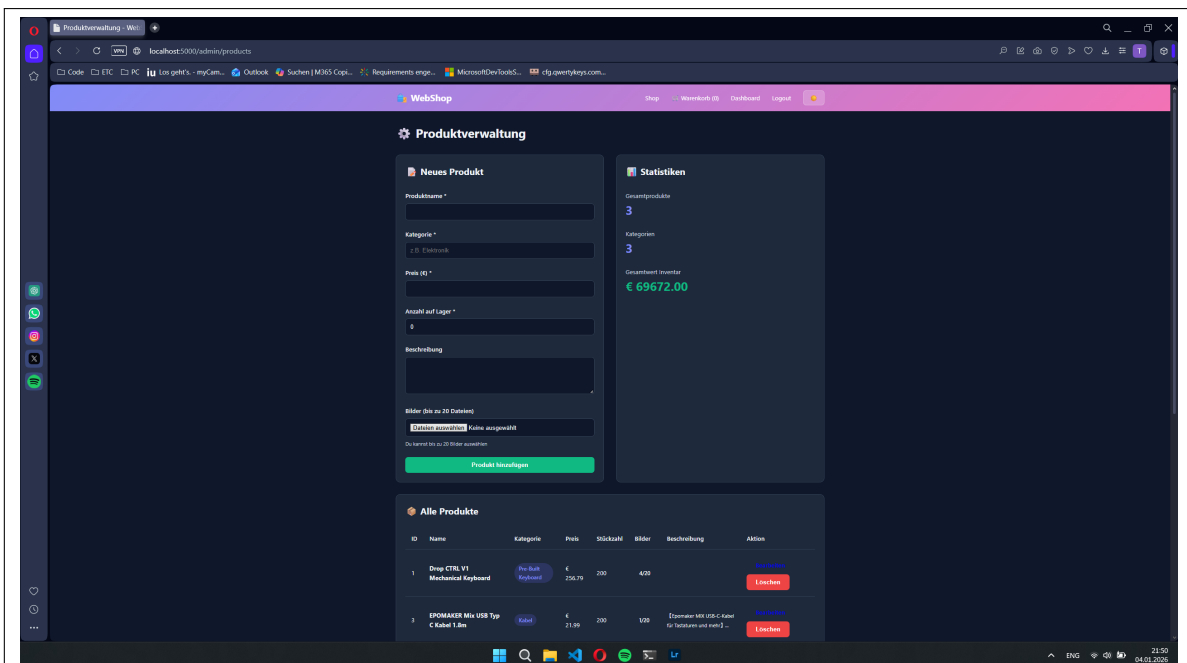


Abbildung: Admin-Interface mit Produktverwaltung

Abbildung 11.8: Admin-Produktverwaltung

11.2 Architektur-Diagramme

11.2.1 Schichtenmodell

[Architektur-Schichtenmodell einfügen]

Abbildung zeigt die vier Schichten (von oben nach unten):

- Presentation Layer (Flask Routes, Jinja2 Templates)
- Business Logic Layer (Services, Checkout)
- Data Access Layer (Hybrid Backend, Storage)
- Data Storage Layer (SQLite, CSV)

Abbildung 11.9: Architektur-Schichtenmodell des Onlineshops

11.2.2 Entity-Relationship-Diagramm

[ER-Diagramm einfügen]

Abbildung: Entity-Relationship-Diagramm mit Entitäten, Attributen und Beziehungen

Die Abbildung sollte folgende Elemente zeigen:

- USER-Tabelle (id, name, email, password, role)
- PRODUCT-Tabelle (id, name, price, category, description, stock, images)
- ORDER-Tabelle (id, user_id, total, status, created_at, payment_provider)
- ORDER_ITEM-Tabelle (id, order_id, product_id, quantity, price)
- CONSENT-Tabelle (id, user_id, consent_type, value, created_at)
- AUDIT_LOG-Tabelle (id, user_id, event_type, action, ip_address, created_at)

Abbildung 11.10: Entity-Relationship-Diagramm des Onlineshops