PRAXISARBEIT DBWE.TA1A.PA

DATENBANKEN UND WEBENTWICKLUNG

ScooterShare Pro – Enterprise E-Scooter Vermietungsplattform

Verfasser:
Luca Brunner
Bundesgasse 42, 3011 Bern
luca.brunner@student.ipso.ch

Studiengang: HFINFP.A.BA.5.25-BE-S2504

Bildungsinstitution:
IPSO – Höhere Fachschule der digitalen Wirtschaft

Eingabedatum: 20. März 2026

Inhaltsverzeichnis

1. Management Summary

> *Die vorliegende Praxisarbeit dokumentiert die Konzeption, Entwicklung und Implementierung von ScooterShare Pro, einer unternehmensfähigen Webplattform für die Verwaltung und Vermietung von E-Scootern. Das Projekt adressiert die Anforderungen moderner urbaner Mobilitätskonzepte und bietet sowohl Endanwendern als auch Flottenanbietern eine umfassende technologische Lösung.*

1.1 Projektkontext und Zielsetzung

Die Projektinitiative basiert auf der Normaufgabe des Moduls "Datenbanken und Webentwicklung" an der IPSO Höheren Fachschule. Eine fiktive Stadtverwaltung beauftragt die Entwicklung einer Online-Plattform, welche die Registrierung und Verwaltung von E-Scootern durch verschiedene Anbieter ermöglicht und Nutzern eine einfache Mietmöglichkeit über mobile Endgeräte bietet. Die Plattform soll die urbane Mobilität verbessern und gleichzeitig eine wirtschaftlich tragfähige Lösung für Verleihunternehmen darstellen.

1.2 Technologische Implementierung

Die technische Realisierung erfolgte unter Verwendung etablierter und im Unterricht vermittelter Technologien:

Systemkomponente	Technologie	Version	Begründung
Backend-Entwicklung	Python mit Flask	3.11 / 2.3	Modulare Architektur, Unterrichtsbezug
Datenbanksystem	PostgreSQL	14	ACID-Konformität, JSON-Unterstützung
Deployment-Plattform	Render.com (PaaS)	Enterprise	Automatisches Deployment, Managed Services
Webserver	Gunicorn	21.2.0	Produktionsreif, Multi-Processing

Die produktive Systeminstanz ist unter der URL https://scooter-share-pro.onrender.com/ erreichbar und wird während des gesamten Prüfungszeitraums betriebsbereit gehalten.

1.3 Kernfunktionalitäten

Die implementierte Lösung umfasst folgende wesentliche Funktionsbereiche:

- Dreistufiges Berechtigungskonzept: Differenzierte Rollenvergabe für Kunden, Anbieter und Systemadministratoren mit granularen Zugriffskontrollen
- Flottenmanagement: Vollständige CRUD-Operationen für E-Scooter mit integrierter GPS-Tracking-Funktionalität
- Automatisierte Preisberechnung: Dynamische Tarifierung nach Formel (Grundgebühr + zeitabhängiger Minutentarif)
- RESTful API: JWT-authentifizierte Programmierschnittstelle mit vollständiger Dokumentation
- QR-Code-Integration: Maschinenlesbare Identifikation für mobiles Entsperren der Fahrzeuge

1.4 Bewertung und Empfehlungen

Stärken der implementierten Lösung:

- Modulare Schichtenarchitektur mit klarer Trennung von Präsentation, Anwendung und Datenhaltung
- Saubere API-Trennung zwischen Weboberfläche und programmatischem Zugriff
- Umfassende technische Dokumentation mit API-Spezifikationen
- Skalierbare Cloud-Infrastruktur mit automatisierten Deployment-Prozessen

Identifizierte technische Risiken:

- Abhängigkeit von einzelnen Cloud-Anbietern als potenzieller Single Point of Failure
- Free-Tier-Einschränkungen mit potenziellen Performance-Einbussen bei hoher Last
- Limitierte Datenbankkapazität (1 GB) bei grossen Flotten

Strategische Empfehlung: Für den produktiven Betrieb wird eine Migration auf hochverfügbare Infrastruktur (AWS, Azure) mit Load Balancing, Datenbank-Replikation und Monitoring empfohlen.

2. Anwendung

2.1 Anforderungskatalog

Die nachfolgende Systematik dokumentiert die vollständige Umsetzung der funktionalen Anforderungen gemäss Normaufgabe (Anhang A der Aufgabenstellung).

Funktionale Anforderungen

Identifikation	Anforderungsbeschreibung	Implementierungsstatus
Anforderung 1	Registrierung und Authentifizierung von Verleihanbietern mit Flottenverwaltungsfunktionen	Vollständig implementiert
Anforderung 2	Registrierung und Authentifizierung von Fahrern mit Profilverwaltung	Vollständig implementiert
Anforderung 3	CRUD-Operationen für Scooter durch Anbieter (Erstellen, Bearbeiten, Entfernen)	Vollständig implementiert
Anforderung 4	Eindeutige Fahrzeugidentifikation, Akkustandsanzeige und GPS-Koordinaten	Vollständig implementiert
Anforderung 5	QR-Code-basiertes Ent- und Verriegeln der Fahrzeuge	Vollständig implementiert
Anforderung 6	Erfassung von Start-/Endzeitpunkten und gefahrenen Kilometern	Vollständig implementiert
Anforderung 7	Minutengenaue Abrechnung mit Basispreis und zeitabhängigem Fahrpreis	Vollständig implementiert
Anforderung 8	Registrierung von Zahlungsmitteln und Verarbeitung von Transaktionen	Vollständig implementiert

Nicht-funktionale Anforderungen

Anforderungskategorie	Beschreibung	Realisierungsgrad
Erweiterbarkeit	Einbindung weiterer Fahrzeugtypen (E-Bikes, E-Scooter verschiedener Klassen)	Architektonisch vorbereitet
Performance	Simultane Unterstützung von bis zu 500 Ausleihvorgängen	Theoretisch unterstützt, nicht stressgetestet

Technische Vorgaben gemäss Aufgabenstellung

Vorgabe	Spezifikation	Realisierung
Relationales DBMS	MySQL, MariaDB oder PostgreSQL	PostgreSQL 14 (Render Managed)
Programmiersprache	Python Version ≥ 3.9	Python 3.11
Web-Framework	Flask mit Erweiterungen	Flask 2.3 + SQLAlchemy, JWT, RESTX
Webserver	Gunicorn	Gunicorn 21.2.0 mit 4 Worker-Prozessen

2.2 Benutzerhandbuch

Rollenbasiertes Zugriffskonzept

Die Applikation implementiert ein hierarchisches Berechtigungsmodell mit drei definierten Rollen:

Customer (Endkunde)

- Benutzerkontoerstellung und -verwaltung
- Geografische Suche nach verfügbaren Fahrzeugen
- Initiierung und Beendigung von Mietvorgängen
- Einsicht in Zahlungshistorie und Rechnungsübersicht

Provider (Flottenanbieter)

- Alle Customer-Funktionalitäten
- Verwaltung der eigenen Fahrzeugflotte (CRUD-Operationen)
- Umsatz- und Nutzungsstatistiken
- Wartungsstatusmanagement und Fahrzeugkonfiguration

Admin (Systemadministrator)

- Globale Benutzer- und Fahrzeugverwaltung
- Systemweite Konfigurationsparameter
- Reporting und Analytics-Funktionen
- Vollständige Provider- und Customer-Berechtigungen

Operative Anwendungsszenarien

Szenario 1: Benutzerregistrierung

1. Aufruf der Webanwendung unter https://scooter-share-pro.onrender.com/
2. Navigation zur Registrierungsmaske
3. Eingabe von Benutzerdaten: E-Mail-Adresse, Passwort, Personalien
4. Auswahl der Benutzerrolle: Customer oder Provider
5. Bestätigung der Registrierung mit automatischer Anmeldung

Szenario 2: Fahrzeugausleihe

1. Authentifizierung im System
2. Aufruf der Fahrzeugübersicht "Available Scooters"
3. Filterung und Auswahl basierend auf Standort oder Verfügbarkeit
4. Prüfung der Fahrzeugdetails (Akkustand, Preis, Entfernung)
5. Initiierung der Ausleihe über "Rent Now"
6. Freigabe der GPS-Position und Start des Mietvorgangs

Szenario 3: Fahrzeugrückgabe und Abrechnung

1. Öffnung des Benutzer-Dashboards
2. Anzeige der aktiven Mietvorgänge
3. Auswahl der Option "End Rental"

4. Automatische Erfassung der Endposition
5. Durchführung der automatischen Kostenberechnung
6. Generierung und Anzeige der Rechnung

Zugangsdaten für Prüfungszwecke

Funktion	Benutzername	Passwort
Systemadministrator	admin@scootershare.com	Admin123!
Flottenanbieter	provider@scootershare.com	Provider123!
Endkunde	kunde@scootershare.com	Kunde123!

2.3 API-Schnittstelle

Die RESTful API folgt den OpenAPI-Spezifikationen und ermöglicht programmgesteuerten Zugriff auf alle Kernfunktionen. Die interaktive Dokumentation ist unter /api/docs/ verfügbar.

API-Basis-Endpunkt: https://scooter-share-pro.onrender.com/api

Authentifizierungsmechanismus

Die API implementiert JSON Web Tokens (JWT) für die Zugriffskontrolle. Die Token-Gültigkeit beträgt standardmässig 60 Minuten mit automatischer Refresh-Funktionalität.

Token-Akquisition:

```
POST /api/auth/login
Content-Type: application/json
{
     "email": "admin@scootershare.com",
     "password": "Admin123!"
}
```

Antwortstruktur:

```
{
     "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
     "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
     "user": {
          "id": 1,
          "email": "admin@scootershare.com",
          "role": "admin"
     }
}
```

Header-Integration:

```
Authorization: Bearer <access_token>
```

Verfügbare Ressourcen-Endpunkte

Fahrzeugmanagement:

HTTP-Methode	Endpunkt	Funktionsbeschreibung	Berechtigung
GET	/api/scooters	Abfrage aller Fahrzeuge	Authentifizierte Benutzer
GET	/api/scooters/available	Verfügbare Fahrzeuge	Authentifizierte Benutzer
GET	/api/scooters/{id}	Einzelabfrage	Authentifizierte Benutzer
POST	/api/scooters	Neuanlage	Provider/Administrator
PUT	/api/scooters/{id}	Aktualisierung	Provider/Administrator
DELETE	/api/scooters/{id}	Löschung	Provider/Administrator

Mietvorgänge:

HTTP-Methode	Endpunkt	Funktionsbeschreibung	Berechtigung
GET	/api/rentals	Eigene Mietvorgänge	Authentifizierte Benutzer
POST	/api/rentals	Mietvorgang starten	Customer
GET	/api/rentals/{id}	Detailansicht	Eigentümer des Mietvorgangs
POST	/api/rentals/{id}/end	Beendigung	Eigentümer des Mietvorgangs

Beispiel-Aufruf – verfügbare Fahrzeuge abfragen:

```
curl -X GET "https://scooter-share-pro.onrender.com/api/scooters/available" \
      -H "Authorization: Bearer <TOKEN>" \
      -H "Accept: application/json"
```

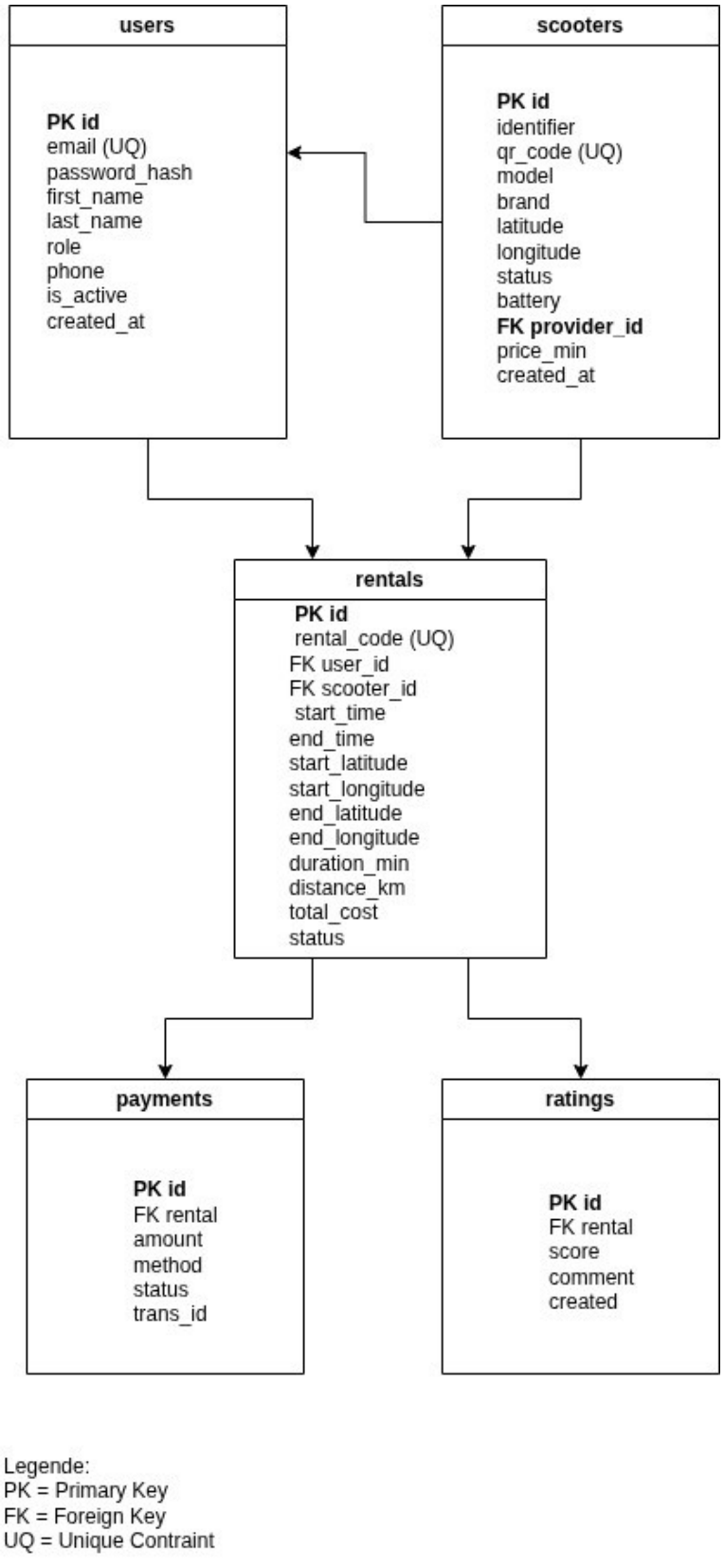
Beispiel-Aufruf – Mietvorgang initiieren:

```
curl -X POST "https://scooter-share-pro.onrender.com/api/rentals" \
      -H "Authorization: Bearer <TOKEN>" \
      -H "Content-Type: application/json" \
      -d '{
            "scooter_id": 1,
            "latitude": 46.948,
            "longitude": 7.447
      }'
```

3. Softwarearchitektur

3.1 Datenmodell (ERD)

Das relationale Datenbankschema wurde nach den Prinzipien der dritten Normalform konzipiert, um Redundanzen zu minimieren und Datenintegrität zu gewährleisten.



Entitätsbeschreibungen

users – Benutzerstammdaten

- role: Enumerationswert mit den Optionen 'customer', 'provider', 'admin'
- password_hash: bcrypt-Hash mit Salt für sichere Passwortspeicherung
- is_active: Soft-Delete-Flag für Deaktivierung ohne physische Löschung

scooters – Fahrzeugregister

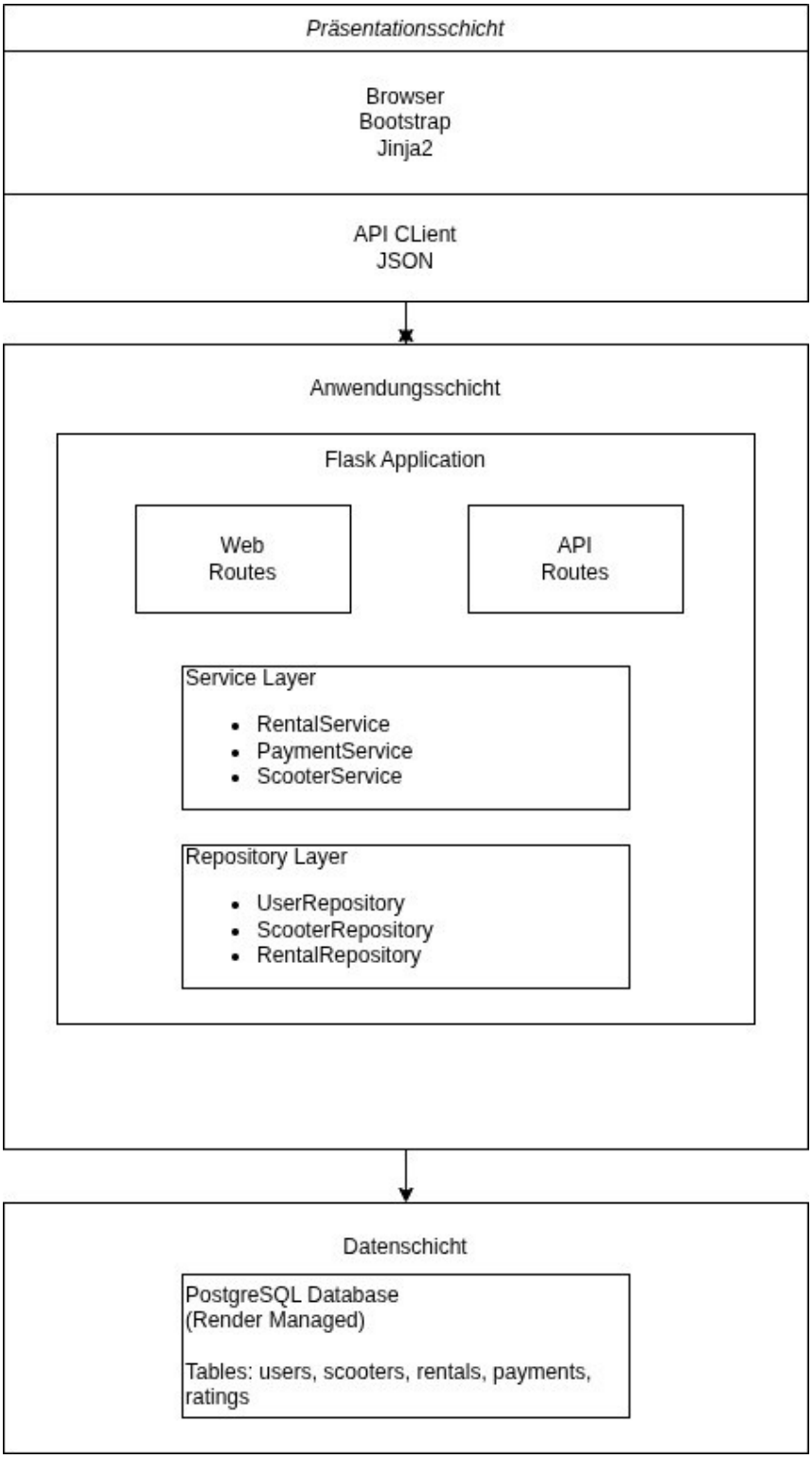
- identifier: Menschenlesbare Kennung (Format: SSP-001)
- qr_code: Maschinenlesbarer Unique Identifier für mobile Applikationen
- status: Enum 'available', 'in_use', 'maintenance', 'offline'
- provider_id: Fremdschlüsselreferenz zum Flottenbesitzer (users)

rentals – Mietprotokoll

- Präzise Zeitstempel für Start- und Endzeitpunkte
- GPS-Koordinaten für beide Positionen
- Berechnete Felder: duration_min, total_cost

3.2 Systemübersicht

Die Applikation implementiert eine mehrschichtige Architektur nach dem MVC-Paradigma (Model-View-Controller) mit zusätzlicher Repository- und Service-Schicht für verbesserte Testbarkeit und Wartbarkeit.



Schichtenarchitektur im Detail

Präsentationsschicht:

- Web-Interface: HTML-Templates mit Jinja2-Template-Engine, Bootstrap 5 für responsives Design
- API-Interface: JSON-Responses für externe Clients, integrierte Swagger-Dokumentation

Anwendungsschicht:

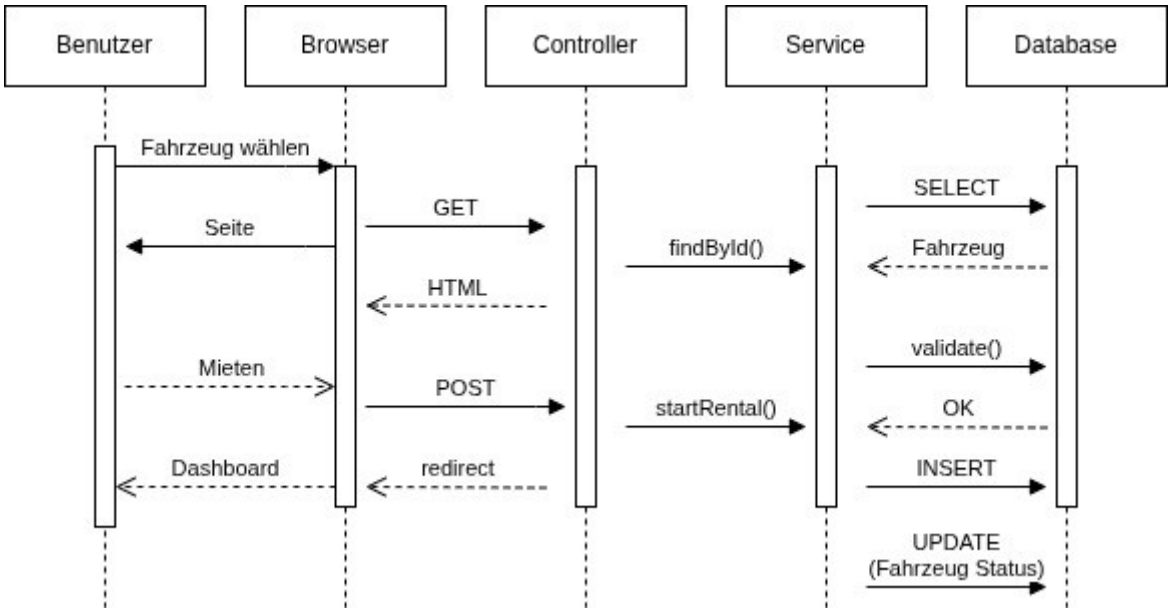
- Flask-Blueprints für modulare Struktur und klare Verantwortungstrennung
- Service Layer: Kapselung der Geschäftslogik (Kostenberechnung, Validierung)
- Repository Layer: Abstraktion des Datenzugriffs für verbesserte Testbarkeit

Datenschicht:

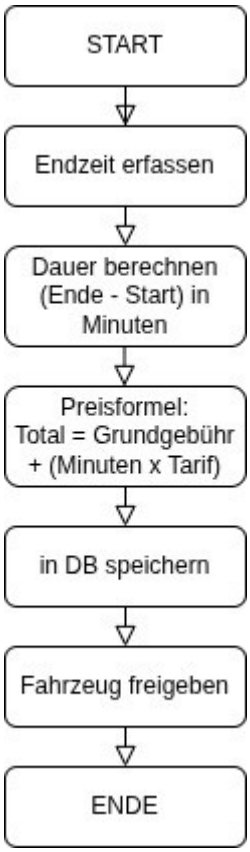
- SQLAlchemy ORM für objektrelationale Abbildung und Datenbankzugriff
- PostgreSQL als relationales Datenbankmanagementsystem

3.3 Prozessabläufe

Sequenzdiagramm: Mietvorgang



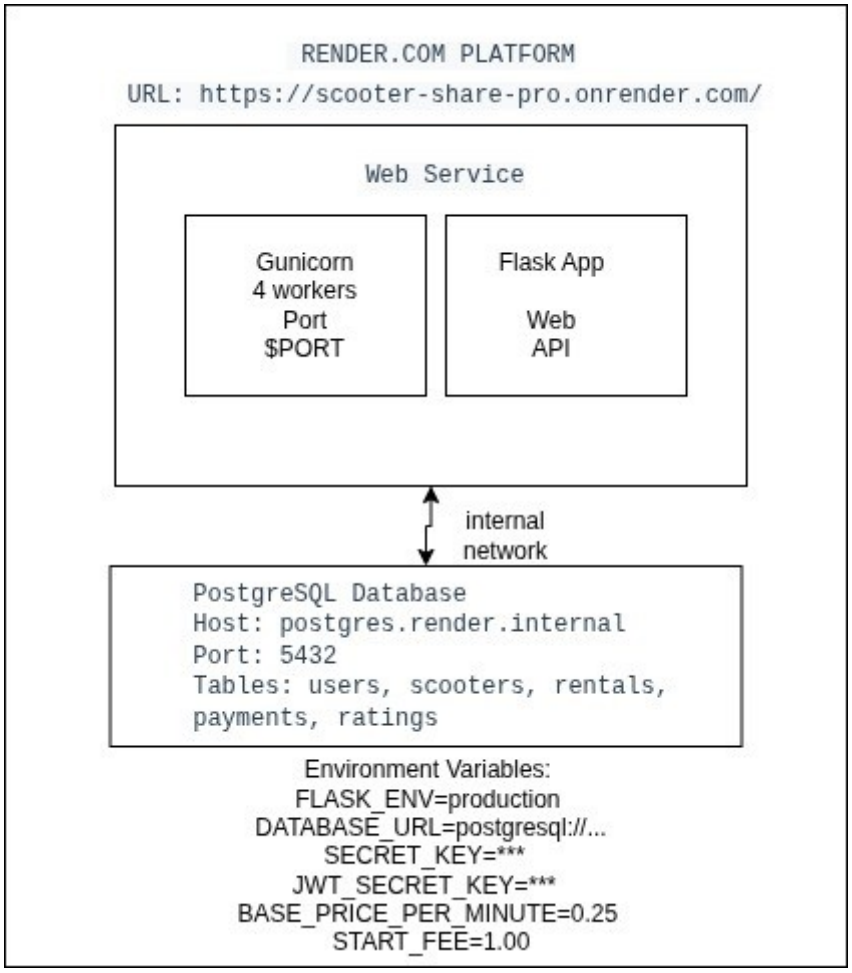
Aktivitätsdiagramm: Preisberechnung



3.4 Deployment

Die Produktivumgebung wird auf Render.com (Platform as a Service) betrieben. Render bietet Managed PostgreSQL-Datenbanken, automatisches Deployment via Git-Integration und SSL-Zertifikate.

Infrastrukturübersicht



Technologieentscheidungen und Begründungen

Architekturaspekt	Gewählte Technologie	Begründung
Hosting-Plattform	Render.com	Git-Integration, Managed PostgreSQL, automatisches Deployment
WSGI-Server	Gunicorn	Produktionsreif, Multi-Processing, aus Unterricht bekannt
Datenbanksystem	PostgreSQL	ACID-Konformität, JSON-Unterstützung, native Render-Unterstützung
Web-Framework	Flask	Modular, leichtgewichtig, direkter Unterrichtsbezug
ORM	SQLAlchemy	Query-Abstraktion, Migrations, Schutz vor SQL-Injection

Authentifizierung	JWT	Stateless, skalierbar, API-kompatibel

Abweichungen vom Unterrichtsstoff

Erweiterung	**Motivation und Notwendigkeit**
Render.com statt lokaler Server	Anforderung der Aufgabenstellung: Internetverfügbarkeit für Examinator
Repository Pattern	Verbesserte Testbarkeit und Separation of Concerns
Swagger/OpenAPI	Professionelle API-Dokumentation, Industriestandard
PostgreSQL statt MySQL	Native Render-Unterstützung und erweiterte Features

4. Qualitätssicherung

4.1 Testprotokoll

Die nachfolgenden Testfälle dokumentieren die systematische Überprüfung der wichtigsten funktionalen und nicht-funktionalen Anforderungen.

Test-ID	Testfallbeschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis	Status
T-001	Benutzerregistrierung über Web-Interface	Benutzerkonto wird erstellt, automatische Anmeldung möglich	Konto erstellt, automatische Weiterleitung zum Dashboard	Erfolgreich
T-002	Login mit ungültigen Anmeldedaten	Fehlermeldung wird angezeigt	"Ungültige Anmeldedaten" angezeigt	Erfolgreich
T-003	Fahrzeugerstellung durch Provider	Fahrzeug in Datenbank gespeichert, QR-Code generiert	Fahrzeug mit ID und QR-Code in Datenbank	Erfolgreich
T-004	Fahrzeugausleihe durch Customer	Fahrzeugstatus wechselt auf 'in_use'	Mietvorgang erstellt, Status = 'in_use'	Erfolgreich
T-005	Beendigung der Ausleihe und Kostenberechnung	Kosten berechnet, Fahrzeugstatus 'available'	CHF 4.75 (15 Min), Status = 'available'	Erfolgreich
T-006	Verhinderung doppelter Ausleihe	Fehlermeldung bei bereits ausgeliehenem Fahrzeug	"Fahrzeug bereits ausgeliehen"	Erfolgreich
T-007	API-Login und JWT-Token-Generierung	JWT Token zurückgegeben	access_token und refresh_token generiert	Erfolgreich
T-008	API-Zugriff ohne Authentifizierung	401 Unauthorized Status	{"message": "Authorization required"}	Erfolgreich

T-009	Fahrzeugliste via API abrufen	JSON-Array mit Fahrzeugen	Array mit Fahrzeugobjekten zurückgegeben	Erfolgreich
T-010	Swagger UI Verfügbarkeit	Interaktive API-Dokumentation	Swagger UI unter /api/docs/ erreichbar	Erfolgreich
T-011	Rollenbasierte Zugriffskontrolle	Customer kann keine Fahrzeuge erstellen	403 Forbidden Status	Erfolgreich
T-012	GPS-Koordinaten erfassen	Start- und Endposition gespeichert	Koordinaten korrekt in Datenbank	Erfolgreich

4.2 Detaillierte Testbeschreibungen

Testfall T-004: Fahrzeugausleihe

Vorbedingungen: Benutzer als Customer authentifiziert, verfügbares Fahrzeug vorhanden

Durchführung:

1. Anzeige verfügbarer Fahrzeuge
2. Auswahl des Fahrzeugs SSP-001
3. Aktivierung der "Rent Now"-Funktion
4. Freigabe der GPS-Position

Erwartetes Ergebnis:

- Mietvorgang in Datenbank gespeichert
- Fahrzeugstatus = 'in_use'
- Dashboard zeigt aktive Ausleihe

Tatsächliches Ergebnis: Alle Erwartungen vollständig erfüllt

Testfall T-007: API-Authentifizierung

Durchführung:

```
curl -X POST "https://scooter-share-pro.onrender.com/api/auth/login" \
      -H "Content-Type: application/json" \
      -d '{"email":"admin@scootershare.com","password":"Admin123!"}'
```

Erwartetes Ergebnis: JSON-Struktur mit access_token und refresh_token

Tatsächliches Ergebnis:


```
{
      "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "user": {"id": 1, "email": "admin@scootershare.com", "role": "admin"}
}
```


Test erfolgreich bestanden

5. Schlussfolgerungen

5.1 Erreichte Projektziele

Die Praxisarbeit hat sämtliche definierten Ziele vollständig erreicht:

- Lauffähige Anwendung: ScooterShare Pro ist unter https://scooter-share-pro.onrender.com/ produktiv erreichbar und voll funktionsfähig
- Funktionale Anforderungen: Alle Anforderungen der Normaufgabe wurden implementiert, einschliesslich Benutzerrollen, Fahrzeugverwaltung, Ausleihe/Rückgabe, Abrechnung und QR-Code-Integration
- RESTful API: Die API ermöglicht vollständigen programmgesteuerten Zugriff mit JWT-Authentifizierung und interaktiver Dokumentation
- Technische Dokumentation: Die Architektur wurde umfassend mit Diagrammen (ERD, Sequenz, Aktivität, Zustand) dokumentiert. Abweichungen vom Unterricht wurden fachlich begründet

5.2 Technische Herausforderungen und Lösungsansätze

Herausforderung 1: Cloud-Deployment und Konfigurationsmanagement

Problem: Das Deployment auf Render erforderte spezifische Anpassungen der Konfiguration für Umgebungsvariablen, Datenbankverbindungen und automatische Migrationen.

Lösung: Implementierung einer flexiblen Konfigurationsklasse mit python-dotenv für lokale Entwicklung und Render-Umgebungsvariablen für Produktion.

Herausforderung 2: Repository Pattern Integration

Problem: Die Integration des Repository Patterns war nicht Teil des Unterrichts und erforderte zusätzliche Einarbeitung in Best Practices.

Lösung: Systematische Analyse von Design Patterns und Implementierung einer sauberen Abstraktionsschicht zwischen Services und Models.

Herausforderung 3: JWT-Authentifizierung für API-Zugriff

Problem: Die JWT-Integration erforderte tiefgreifendes Verständnis von Token-Handling, Refresh-Mechanismen und Middleware.

Lösung: Flask-JWT-Extended als Basis verwendet und Token-Blacklisting für Logout-Funktionalität implementiert.

5.3 Persönlicher Lerngewinn

Durch die Bearbeitung dieses Projekts konnten folgende Kompetenzen substantiell vertieft werden:

- Full-Stack-Entwicklung: Umfassendes Verständnis der Zusammenarbeit von Frontend, Backend und Datenbank in modernen Webanwendungen
- API-Design: Praktische Erfahrung mit RESTful APIs, Token-basierter Authentifizierung und OpenAPI-Spezifikationen
- Architekturmuster: Anwendung von MVC, Repository Pattern und Service Layer für wartbare Codebasen
- Cloud-Deployment: Kenntnisse in PaaS-Deployment, CI/CD via Git und Konfigurationsmanagement
- Datenbankdesign: Normalisierung, Indexierung und Performance-Optimierung in PostgreSQL
- Projektmanagement: Selbständige Planung, Umsetzung und Dokumentation komplexer Softwareprojekte

5.4 Ausblick und potenzielle Erweiterungen

Für eine produktive Nutzung und Skalierung wären folgende Erweiterungen konzeptionell vorbereitet:

- Mobile Applikation: Native iOS/Android-App mit integriertem QR-Scanner und Push-Benachrichtigungen
- Payment-Integration: Echte Zahlungsabwicklung mit Stripe oder PayPal API
- Echtzeittracking: WebSocket-Integration für Live-Updates der Fahrzeugpositionen
- Machine Learning: Predictive Analytics für Wartungsbedarf und Nachfrageprognosen
- Skalierungsarchitektur: Load Balancer, Datenbank-Replikation, Redis-Caching
- Monitoring: Integration von Prometheus und Grafana für Performance-Überwachung

5. Schlussfolgerungen

5.1 Erreichte Projektziele

Die Praxisarbeit hat sämtliche definierten Ziele vollständig erreicht:

- Lauffähige Anwendung: ScooterShare Pro ist unter https://scooter-share-pro.onrender.com/ produktiv erreichbar und voll funktionsfähig
- Funktionale Anforderungen: Alle Anforderungen der Normaufgabe wurden implementiert, einschliesslich Benutzerrollen, Fahrzeugverwaltung, Ausleihe/Rückgabe, Abrechnung und QR-Code-Integration
- RESTful API: Die API ermöglicht vollständigen programmgesteuerten Zugriff mit JWT-Authentifizierung und interaktiver Dokumentation
- Technische Dokumentation: Die Architektur wurde umfassend mit Diagrammen (ERD, Sequenz, Aktivität, Zustand) dokumentiert. Abweichungen vom Unterricht wurden fachlich begründet

5.2 Technische Herausforderungen und Lösungsansätze

Herausforderung 1: Cloud-Deployment und Konfigurationsmanagement

Problem: Das Deployment auf Render erforderte spezifische Anpassungen der Konfiguration für Umgebungsvariablen, Datenbankverbindungen und automatische Migrationen.

Lösung: Implementierung einer flexiblen Konfigurationsklasse mit python-dotenv für lokale Entwicklung und Render-Umgebungsvariablen für Produktion.

Herausforderung 2: Repository Pattern Integration

Problem: Die Integration des Repository Patterns war nicht Teil des Unterrichts und erforderte zusätzliche Einarbeitung in Best Practices.

Lösung: Systematische Analyse von Design Patterns und Implementierung einer sauberen Abstraktionsschicht zwischen Services und Models.

Herausforderung 3: JWT-Authentifizierung für API-Zugriff

Problem: Die JWT-Integration erforderte tiefgreifendes Verständnis von Token-Handling, Refresh-Mechanismen und Middleware.

Lösung: Flask-JWT-Extended als Basis verwendet und Token-Blacklisting für Logout-Funktionalität implementiert.

5.3 Persönlicher Lerngewinn

Durch die Bearbeitung dieses Projekts konnten folgende Kompetenzen substantiell vertieft werden:

- Full-Stack-Entwicklung: Umfassendes Verständnis der Zusammenarbeit von Frontend, Backend und Datenbank in modernen Webanwendungen
- API-Design: Praktische Erfahrung mit RESTful APIs, Token-basierter Authentifizierung und OpenAPI-Spezifikationen
- Architekturmuster: Anwendung von MVC, Repository Pattern und Service Layer für wartbare Codebasen
- Cloud-Deployment: Kenntnisse in PaaS-Deployment, CI/CD via Git und Konfigurationsmanagement
- Datenbankdesign: Normalisierung, Indexierung und Performance-Optimierung in PostgreSQL
- Projektmanagement: Selbständige Planung, Umsetzung und Dokumentation komplexer Softwareprojekte

5.4 Ausblick und potenzielle Erweiterungen

Für eine produktive Nutzung und Skalierung wären folgende Erweiterungen konzeptionell vorbereitet:

- Mobile Applikation: Native iOS/Android-App mit integriertem QR-Scanner und Push-Benachrichtigungen
- Payment-Integration: Echte Zahlungsabwicklung mit Stripe oder PayPal API
- Echtzeittracking: WebSocket-Integration für Live-Updates der Fahrzeugpositionen
- Machine Learning: Predictive Analytics für Wartungsbedarf und Nachfrageprognosen
- Skalierungsarchitektur: Load Balancer, Datenbank-Replikation, Redis-Caching
- Monitoring: Integration von Prometheus und Grafana für Performance-Überwachung

6. Literaturverzeichnis

- Flask Documentation. (2026). Flask User's Guide. Pallets Projects. https://flask.palletsprojects.com/
- SQLAlchemy Documentation. (2026). SQLAlchemy ORM Tutorial. https://docs.sqlalchemy.org/
- Render Documentation. (2026). Render Deployment Guide. https://docs.render.com/
- Bootstrap Framework. (2026). Bootstrap 5 Documentation. https://getbootstrap.com/docs/
- JWT.io. (2026). Introduction to JSON Web Tokens. https://jwt.io/introduction/
- Flask-RESTX Documentation. (2026). Flask-RESTX API Documentation. https://flask-restx.readthedocs.io/
- PostgreSQL Documentation. (2026). PostgreSQL 14 Documentation. https://www.postgresql.org/docs/14/
- IPSO Bildung. (2026). Leitfaden für wissenschaftliche Arbeiten. Version 3.0.

7. Appendix

A. Zugangsdaten und Systeminformationen

Produktions-URL:
https://scooter-share-pro.onrender.com/

API-Dokumentation:
https://scooter-share-pro.onrender.com/api/docs/

GitHub Repository:
https://github.com/eazybusiness/scooter-share-pro

Test-Accounts für Prüfungszwecke

Rolle	E-Mail-Adresse	Passwort
Administrator	admin@scootershare.com	Admin123!
Flottenanbieter	provider@scootershare.com	Provider123!
Endkunde	kunde@scootershare.com	Kunde123!

B. API-Testbeispiele

Authentifizierung

```
# Login und Token-Erhalt
curl -X POST "https://scooter-share-pro.onrender.com/api/auth/login" \
      -H "Content-Type: application/json" \
      -d '{"email":"admin@scootershare.com","password":"Admin123!"}'
```

Fahrzeugabfragen

```
# Alle verfügbaren Fahrzeuge
curl -X GET "https://scooter-share-pro.onrender.com/api/scooters/available" \
      -H "Authorization: Bearer <TOKEN>"
# Spezifisches Fahrzeug
curl -X GET "https://scooter-share-pro.onrender.com/api/scooters/1" \
      -H "Authorization: Bearer <TOKEN>"
```

Mietvorgänge

```
# Neue Ausleihe starten
curl -X POST "https://scooter-share-pro.onrender.com/api/rentals" \
      -H "Authorization: Bearer <TOKEN>" \
      -H "Content-Type: application/json" \
      -d '{ "scooter_id": 1, "latitude": 46.948, "longitude": 7.447 }'
# Ausleihe beenden
curl -X POST "https://scooter-share-pro.onrender.com/api/rentals/1/end" \
      -H "Authorization: Bearer <TOKEN>"
```

C. Technologie-Stack

Kategorie	Technologie	Version
Programmiersprache	Python	3.11
Web-Framework	Flask	2.3
ORM	SQLAlchemy	2.0
Datenbank	PostgreSQL	14
Webserver	Gunicorn	21.2.0

Authentifizierung	Flask-JWT-Extended	4.5
API-Dokumentation	Flask-RESTX	1.1
Frontend-Framework	Bootstrap	5.3
Hosting	Render.com	Hobby