Steven Hofmann, Dominik Hübner, Jeongmin Lee

|

Projekt: Webseite für eine Autovermietung

Gruppe 4

**Inhalt**

[**1 Beschreibung** 1](#_Toc158240401)

[**2 Anforderungen** 1](#_Toc158240402)

[**3 Verwendete Frameworks und Technologien** 2](#_Toc158240403)

[**4 Architekturmodell** 4](#_Toc158240404)

[**5 Datenbankstruktur** 5](#_Toc158240405)

[**6 Funktionsweise** 5](#_Toc158240406)

[**6.1 Konfiguration und Datenbankverbindung** 5](#_Toc158240407)

[**6.2 Server-Logik** 6](#_Toc158240408)

[**6.3 Frontend und Benutzerinteraktion** 8](#_Toc158240409)

## **1 Beschreibung**

Das Hauptziel dieses Projekts ist die Entwicklung einer umfassenden Webanwendung für eine Autovermietung, die eine benutzerfreundliche und nahtlose Plattform für Kunden und Mitarbeiter bereitstellt und eine effiziente und transparente Interaktion mit dem Autovermietungsservice ermöglicht.

Für Kunden soll die Webanwendung einen einfachen Reservierungsprozess bieten. Dies beinhaltet eine intuitive Fahrzeugsuche, klare Informationen zu Verfügbarkeit und Kosten, sowie die Möglichkeit, Reservierungen einfach über ein Kundenkonto zu verwalten.

Für Mitarbeiter stellt die Webanwendung ein Instrument zur Verfügung, um Reservierungen effektiv zu verwalten. Das schließt ein Dashboard für die Reservierungsübersicht ein, die Möglichkeit zur Bearbeitung von Reservierungen sowie die Verwaltung von Fahrzeugen, Stellplätzen und Rechnungen.

## **2 Anforderungen**

Kunden

* Registrierung
* Login
* Verfügbare Fahrzeuge, pro Standort, Zeitraum, Preis, Fahrzeugklasse filtern
* Reservieren
* Reservierung anschauen & ändern & löschen
* Rechnung einsehen
* Persönliche Daten ändern & anschauen & löschen

Mitarbeiter

* Login
* Fahrzeuge hinzufügen & entfernen & bearbeiten
* Reservierung anschauen & bestätigen & stornieren
* Rechnung einsehen & erstellen & ändern
* Mitarbeiter erstellen & entfernen & bearbeiten
* Persönliche Daten von Kunden anschauen & löschen
* Standorte anlegen & löschen & bearbeiten (Parkplätze)
* Wartung veranlassen (Kunden Rechnung erstellen)

## **3 Verwendete Frameworks und Technologien**

**Python:**

Als Programmiersprache wurde grundlegend Python gewählt, da die nötige Einarbeitung in diese Sprache gegeben ist und dadurch ohne größere Verzögerung mit der Programmierung angefangen werden konnte.

**Flask:**

Es wurde sich für das Flask-Framework für Python entschieden, da es eine schlanke und flexible Lösung für die Entwicklung der gewünschten Webanwendung bietet. Die minimalistische Architektur von Flask ermöglicht es, eine leichte und effiziente Anwendung zu erstellen, die genau auf die spezifischen Anforderungen zugeschnitten ist. Im Vergleich zu umfangreicheren Frameworks wie **Django**, bietet Flask die nötige Agilität in der Entwicklung, ohne dabei Kompromisse bei der Qualität einzugehen.

**Vorteile**

* Umfang: Flask kann sehr schnell installiert und dann direkt genutzt werden.
* Flexibilität: Auch die Flexibilität, die Flask mit sich bringt, ist sehr gut. Es gibt die Möglichkeit, jedes Problem individuell zu lösen und es kann genau jene Bibliotheken implementieren, die wirklich benötigt werden, wodurch sich Projekte individuell angehen lassen.
* Lernkurve: Das Framework ist einfach aufgebaut, erlaubt aber auch die Arbeit an anspruchsvollen Projekten.
* Open Source: Flask ist kostenlos und Open Source.

**Nachteile**

* Umfang: Je nach Einsatzzweck kann der minimalistische Umfang auch ein Nachteil sein. Es muss alles zusätzlich implementiert werden, während andere Frameworks deutlich mehr Funktionen anbieten.
* Abhängigkeit von Drittanbietern: Die Verwendung externer Bibliotheken ist immer auch eine mögliche Problemquelle, diese sind bei Flask aber unabdingbar.

**Ajax**

Ausgeschrieben: **„**Asynchronous JavaScript and XML“ ist eine Schlüsseltechnologie in der modernen Webentwicklung, die es ermöglicht, Webseiten dynamisch durch asynchronen Datenaustausch mit dem Server zu aktualisieren, ohne dass ein Neuladen der gesamten Seite erforderlich ist. Diese Methode führt zu einer verbesserten Benutzererfahrung, da Inhalte nahtlos und ohne Unterbrechungen aktualisiert werden können. Ajax kann mit verschiedenen serverseitigen Technologien verwendet werden, einschließlich Flask, weshalb es als Ergänzung für die Verbindung von Front- und Backend gewählt wurde.

Die Integration von Ajax in eine Flask-basierte Anwendung ermöglicht eine effiziente Bearbeitung von Client-Anfragen aus dem Frontend durch das Backend. Der Prozess beginnt typischerweise auf der Client-Seite, wo eine JavaScript-basierte Ajax-Anfrage durch Benutzerinteraktionen wie das Klicken auf einen Button oder das Einreichen eines Formulars ausgelöst wird. Diese Anfrage wird dann an das Flask-Backend gesendet, das für die Verarbeitung spezifische Routen und Funktionen definiert hat.

Im Flask-Backend wird für jede Ajax-Anfrage eine Route eingerichtet, die mit einer entsprechenden Funktion verknüpft ist. Diese Funktionen sind verantwortlich für die Verarbeitung der eingehenden Daten, die Ausführung von Geschäftslogiken, den Austausch mit Datenbanken und schließlich die Erstellung einer Antwort, die oft im JSON-Format erfolgt, um die einfache Handhabung und Integration in die Client-Seite zu gewährleisten.

Sobald die Antwort vom Flask-Backend generiert und zurück an den Client gesendet wurde, wird die empfangene Information verwendet, um die Webseite dynamisch zu aktualisieren. Diese Aktualisierungen können das Hinzufügen von Inhalten, die Modifikation bestehender Elemente oder die Anzeige von Benachrichtigungen umfassen, alles ohne die Notwendigkeit, die Seite neu zu laden.

Die Nutzung von Ajax in Verbindung mit Flask ermöglicht die Entwicklung reaktiver Webanwendungen, die eine schnelle und flüssige Benutzererfahrung bieten.

**CX\_Oracle:**

Eine Python-Bibliothek, die es ermöglicht, mittels SQL-Befehlen auf Oracle-Datenbanken zuzugreifen und Daten zu lesen, zu schreiben, Transaktionen zu verwalten und Oracle-Datenbankfunktionen aufzurufen.

cx\_Oracle ist Teil des Oracle Call Interface (OCI), eine umfassende Schnittstelle (API) für Oracle Database. cx\_Oracle nutzt diese, um eine Verbindung zur Datenbank herzustellen.

## **4 Architekturmodell**

## **Ein Bild, das Text, Diagramm, Plan, Reihe enthält. Automatisch generierte Beschreibung5 Datenbankstruktur**

## **6 Funktionsweise**

## **6.1 Konfiguration und Datenbankverbindung**

Das Modul database.py verwendet die in **config.json** definierten Konfigurationsdaten, um eine Verbindung zur Datenbank herzustellen. Es liest die Konfigurationsdaten, etabliert eine Datenbankverbindung und hält diese für zukünftige Abfragen vor. Die **config.json**-Datei enthält essentielle Verbindungsparameter wie Benutzernamen, Passwort, Hostadresse, Port und den Servicenamen der Datenbank. Diese Parameter sind kritisch für die Etablierung einer sicheren und effektiven Verbindung zur Oracle-Datenbank, die als Backend für die Autovermietungs-Webseite dient und die relevanten Daten verwaltet.

Zur Herstellung einer Verbindung mit einer Oracle-Datenbank wird die Python-Bibliothek **cx\_Oracle** genutzt. Die Konfigurationsdaten werden aus **config.json** geladen, und auf Basis dieser Informationen wird die Datenbankverbindung konfiguriert. Diese Methode erlaubt eine zentrale Verwaltung der Verbindungsdaten und entkoppelt den verbindungsbezogenen Code von den spezifischen Konfigurationsdetails. Dies vereinfacht die Wartung und ermöglicht eine einfache Anpassung der Anwendung an verschiedene Umgebungen.

Die Funktion **setup\_database()** initialisiert die Datenbank durch das Erstellen von Tabellen und das Definieren von Primär- und Fremdschlüsselbeziehungen. Die notwendigen SQL-Anweisungen werden über separate Funktionen, **get\_create\_table\_statements()** und **get\_alter\_table\_statements()** abgerufen. **execute\_sql\_in\_transaction()** führt diese SQL-Anweisungen innerhalb einer Transaktion aus. Eine Transaktion wird als eine sequenzielle Gruppe von einer oder mehreren Datenbankoperationen betrachtet, die als eine einzige logische Einheit angesehen werden. Im Falle eines erfolgreichen Abschlusses werden alle Operationen durch einen commit bestätigt. Treten Fehler auf, wird ein rollback durchgeführt, um die Integrität der Datenbank zu wahren. Diese Vorgehensweise garantiert, dass alle Anweisungen zusammenhängend und ohne Unterbrechungen ausgeführt werden, was die Stabilität und Zuverlässigkeit der Datenbankoperationen sicherstellt.

Die **setup\_database()** kann dann später im Hauptmodul, der **app.py**, ausgeführt werden, um die Tabellen inital zu erstellen.

## **6.2 Server-Logik**

Die Server-Logik für die Autovermietungs-Webseite, beispielsweise wie sie in den Dateien für die Verwaltung von Fahrzeugen **fahrzeuge.py** und **routes.py, add\_vehicle.html, update\_vehicle.html, delete\_vehicle.html** und **vehicles.html** implementiert ist, bildet das Rückgrat der Anwendung, indem sie die Anfragen vom Frontend verarbeitet, die erforderlichen Datenbankoperationen durchführt und die Ergebnisse an das Frontend zurückgibt. Hier wird die Funktionsweise dieser Komponenten ausführlicher beschrieben:

**Hauptlogik (app.py)**

Das Hauptmodul, worüber die Funktionen aufgerufen und die Flask-Anwendung initialisiert und gestartet wird. Der Zweck besteht in:

* Import aller Blueprints und Funktionen: Da die einzelnen Ressourcen logisch voneinander getrennt wurden, müssen sie in **der app.py** importiert werden, wie auch die Blueprints für die Flask-Routen.
* *Flask-Anwendung*: Die Datei app.py initialisiert eine Flask-Anwendung, die als Hauptpunkt für die Einrichtung der Webserver-Umgebung dient.
* *Registrierung der Blueprints mit den URL-Präfixen*: **app.py** definiert die Grundstruktur für die Routing-Logik, indem es URL-Endpunkte mit spezifischen Funktionen in beispielsweise **fahrzeuge.py** und anderen Modulen verknüpft. Diese Funktionen werden aufgerufen, wenn die entsprechenden URLs vom Client angefordert werden.
* *Aufrufen der Hauptfunktionen*: Die Funktionen zur Initialisierung der Datenbankstruktur und zum initialen Einfügen von Testdaten in die Datenbank werden aufgerufen.

**Fahrzeugverwaltung (fahrzeuge.py)**

* *CRUD-Operationen*: Das Modul **fahrzeuge.py** implementiert die spezifischen Funktionen für CRUD-Operationen (Create, Read, Update, Delete) bezüglich der Fahrzeuge. Es enthält Funktionen zum Hinzufügen neuer Fahrzeuge, zum Abrufen von Fahrzeuginformationen, zum Aktualisieren bestehender Fahrzeugdatensätze und zum Löschen von Fahrzeugen aus der Datenbank.
* *Datenbankinteraktionen*: Für jede dieser Operationen werden SQL-Anweisungen vorbereitet und ausgeführt, indem die Funktionen aus **database.py** und SQL-Befehle aus **sql\_statements.py** verwendet werden. Dies umfasst das Aufbauen der Datenbankverbindung, das Senden von Anfragen an die Datenbank, das Verarbeiten der Ergebnisse und das Schließen der Verbindung.
* *Antwortgenerierung* *für Flask*: Nach der Ausführung der Datenbankoperationen generiert **fahrzeuge.py** Antworten, die typischerweise im JSON-Format sind, um die Ergebnisse oder Fehlermeldungen an das Frontend zurückzugeben. Diese Antworten werden später von Flask verwendet, um die HTTP-Antworten für die Client-Anfragen zu konstruieren.

**Routen und Endpunkte (routes.py)**

* *URL-Definitionen*: In **routes.py** werden die spezifischen URLs (oder "Routen") definiert, die von der Webanwendung bedient werden. Jede Route ist einer Funktion zugeordnet, die ausführt, was geschehen soll, wenn die Route angefragt wird. Zudem werden auch Blueprints der Routen festgelegt, die später von **app.py** aufgerufen werden.
* *Request-Handling*: Das Modul empfängt die HTTP-Anfragen (GET, POST, PUT, DELETE) vom Frontend, extrahiert erforderliche Informationen aus den Anfragen (z.B. Formulardaten oder URL-Parameter) und leitet diese Informationen an die entsprechenden Funktionen in **fahrzeuge.py** weiter.
* *Antwort-Routing*: Nachdem die Logik in **fahrzeuge.py** ausgeführt wurde und eine Antwort generiert hat, kümmert sich **routes.py** darum, diese Antwort in einem Format zurückzugeben, das vom Client verarbeitet werden kann, typischerweise als HTTP-Antwort mit einem entsprechenden Statuscode wie 200 (OK), 201 (Created) oder 500 (Internal Sever Error).

Zusammenfassend bildet die Server-Logik mit Flask das Herzstück der Anwendung, indem sie als Bindeglied zwischen dem Frontend und der Datenbank fungiert. Sie verarbeitet Benutzeranfragen, führt Geschäftslogiken und Datenbankoperationen aus und generiert Antworten, die den aktuellen Zustand der Anwendung für die Benutzer darstellen.

## **6.3 Frontend und Benutzerinteraktion**

Die Frontend-Komponenten und die Benutzerinteraktion der Autovermietungs-Webseite werden durch **HTML**-Seiten und **Ajax** dargestellt, um eine effiziente und interaktive Benutzerumgebung zu schaffen, indem Anfragen im Hintergrund verarbeitet werden, ohne dass sich bei Aktionen des Benutzers die Seite aktualisiert. Im Folgenden wird die Funktionsweise dieser Komponenten ausführlicher beschrieben, basierend auf den Dateien **add\_vehicle.html**, **delete\_vehicle.html**, **update\_vehicle.html**, und **vehicles.html**:

**Fahrzeug hinzufügen (add\_vehicle.html)**

* *Formulareingabe*: Diese Seite enthält ein Formular, über das Benutzer Daten für ein neues Fahrzeug eingeben können, einschließlich Standort-ID, Hersteller, Modell, Baujahr, Verfügbarkeit, Preis, Kilometerstand, Kennzeichen und Fahrzeugklasse.
* *Asynchrone Datenübermittlung*: Durch JavaScript wird ein Event Listener auf das Formular gelegt, der das Standard-Submit-Verhalten unterbindet und stattdessen eine asynchrone **POST**-Anfrage mit den Formulardaten im JSON-Format an den Server sendet. Das ermöglicht eine reibungslose Benutzererfahrung ohne Unterbrechungen durch das Neuladen der Seite.
* *Server-Interaktion*: Die Anfrage wird über die vorher festgelegte Flask-Route an den Endpunkt gesendet, wo sie durch die Funktion **add\_vehicle()** Server verarbeitet wird. Die Serverantwort (Erfolg oder Fehler) wird im Browser-Konsolenlog angezeigt.

**Fahrzeug löschen (delete\_vehicle.html)**

* *Interaktive Liste*: Auf dieser Seite wird eine Liste der Fahrzeuge mit einem "*Löschen*"-Button neben jedem Eintrag angezeigt. Benutzer können ein Fahrzeug zur Löschung auswählen, indem sie den entsprechenden Button drücken.
* *Löschbestätigung und Anfrage*: Ein JavaScript-Event-Handler fordert eine Bestätigung vom Benutzer an, bevor er über die festgelegte Flask-Route eine **DELETE**-Anfrage an /fahrzeuge/delete\_vehicle/{fahrzeugId} sendet, um das ausgewählte Fahrzeug zu löschen.
* *Dynamische Aktualisierung*: Nach einer erfolgreichen Löschoperation wird die Seite neu geladen, um die aktualisierte Fahrzeugliste anzuzeigen.

**Fahrzeug aktualisieren (update\_vehicle.html)**

* *Update-Formular*: Ähnlich wie beim Hinzufügen können Benutzer Fahrzeugdaten über ein Formular aktualisieren. Ein verstecktes Feld enthält die ID des zu aktualisierenden Fahrzeugs. Diese soll sich nicht bearbeiten lassen, da sie automatisch generiert wird.
* *PUT-Anfrage*: Beim Absenden des Formulars wird eine **PUT**-Anfrage mit den aktualisierten Fahrzeugdaten über die vorher festgelegte Flask-Route an /fahrzeuge/update\_vehicle/{fahrzeugId} gesendet. Die Datenübermittlung erfolgt asynchron, um das Neuladen der Seite zu vermeiden.
* *Feedback an Benutzer*: Nach der Verarbeitung der Anfrage durch den Server wird dem Benutzer eine Rückmeldung gegeben, die den Erfolg oder Misserfolg der Aktualisierung anzeigt.

**Fahrzeugliste anzeigen (vehicles.html)**

* *Asynchrones Laden*: Diese Seite verwendet **Ajax**, um die Fahrzeugliste asynchron vom Server zu laden. Beim Laden der Seite wird eine **GET**-Anfrage an /fahrzeuge/vehicles gesendet, um die aktuelle Liste der Fahrzeuge abzurufen.
* *Dynamische Inhaltsdarstellung*: Die empfangenen Fahrzeugdaten werden verwendet, um den Inhalt eines Div-Elements zu füllen, wobei jedes Fahrzeug in einem eigenen Element mit grundlegenden Informationen wie ID, Hersteller und Modell etc. angezeigt wird.
* *Fehlerbehandlung*: Im Falle eines Fehlers bei der Datenabfrage wird eine entsprechende Nachricht angezeigt, um den Benutzer über das Problem zu informieren.

Diese Frontend-Komponenten interagieren eng mit den Backend-Endpunkten. Durch die Verwendung von asynchronen Anfragen und JavaScript können Benutzer die Webseite bedienen, ohne dass vollständige Seitenneuladungen erforderlich sind.