Dokumentation CarRent.

# Aufgabenstellung:

Es soll ein neues Autovermietungssystem „CarRent“ erstellt werden. Das System soll aus einem Server-Teil und optional einen Web-Client bestehen.

- Die Daten sollen mittels Repository Pattern in eine Datenbank gespeichert werden können.

- Die Business Logik soll auf dem Backend laufen und eine REST APIs anbieten.

- Es soll zuerst ein Monolith erstellt werden und später auf eine Micro Service Architektur überführt werden.

Optional:

- Der Web-Client benutzt die REST API um die Funktionen auszuführen.

# Use-Cases:

**Kundenverwaltung:** Der Sachbearbeiter kann Kundeninformationen wie Namen, Adresse und Kundennummer im System verwalten. Er kann neue Kunden erfassen, bestehende Kundeninformationen bearbeiten oder Kunden aus dem System löschen. Zusätzlich kann der Sachbearbeiter nach einem Kunden suchen, indem er entweder den Namen oder die Kundennummer eingibt.

**Autoverwaltung und Suche:** Der Sachbearbeiter kann die Autos von CarRent verwalten. Jedes Auto kann einer bestimmten Klasse (Luxusklasse, Mittelklasse, Einfachklasse) zugeordnet werden und besitzt Informationen wie Marke und Typ sowie eine eindeutige Identifikationsnummer. Der Sachbearbeiter kann nach Autos suchen, indem er nach bestimmten Kriterien wie Marke oder Typ filtert.

**Klassen- und Tagesgebührenverwaltung:** Das System ermöglicht die Verwaltung von Fahrzeugklassen. Jede Klasse (Luxusklasse, Mittelklasse, Einfachklasse) hat eine zugehörige Tagesgebühr. Diese Gebühr wird verwendet, um die Kosten für die Anmietung eines Fahrzeugs aus einer bestimmten Klasse zu berechnen.

**Reservierungserstellung:** Ein Kunde kann eine neue Reservierung vornehmen. Dabei kann er ein Auto aus einer bestimmten Klasse auswählen. Der Kunde gibt die Anzahl der Tage an, für die er das Auto mieten möchte. Das System berechnet automatisch die Gesamtkosten basierend auf der ausgewählten Klasse und der Anzahl der Tage. Nachdem alle Informationen eingegeben wurden, wird die Reservierung im System gespeichert und mit einer eindeutigen Reservationsnummer versehen.

**Umwandlung in Mietvertrag:** Sobald der Kunde das reservierte Auto abholt, wird die Reservierung in einen Mietvertrag umgewandelt. Diese Umwandlung markiert den Beginn des Mietzeitraums. Der Mietvertrag enthält alle relevanten Informationen zur Reservierung sowie zusätzliche Angaben wie den Abholzeitpunkt und den Rückgabetermin.

# Qualitätsziele

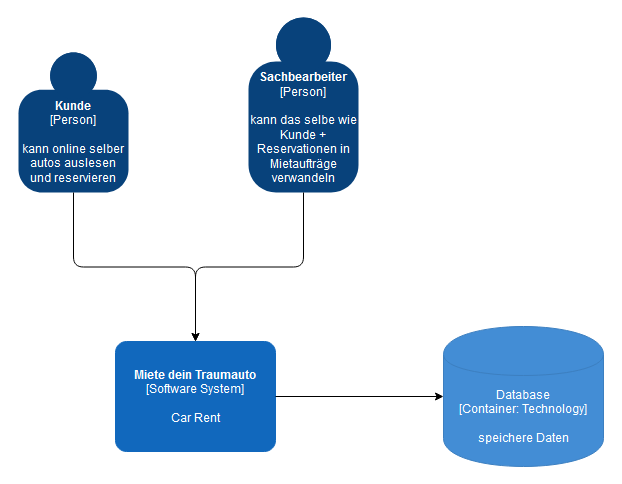
Eine saubere Architektur in einem Projekt zu erreichen, die einerseits alle Zuständigkeiten klärt und abgrenzt und zudem eine Verteilung auf verschiedene Layers und Tiers ermöglicht.

Wir wollen zudem die Möglichkeit haben alle internen und externen Abhängigkeiten zentral zu verwalten.

Eine funktionsfähige Website mit ASP.net erstellen. Mit dieser sollen Autos verwaltet, Kunden, Reservationen und Verträge angelegt werden.

# Big Picture:

## Context:



## Container:

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

## Component:

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

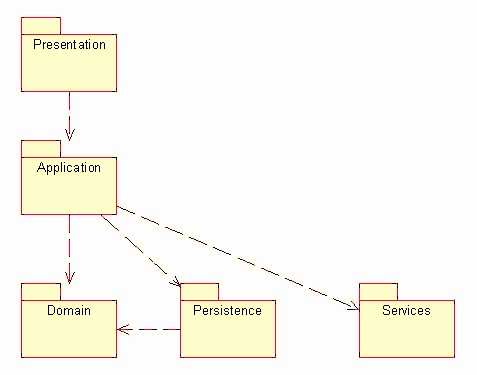
# Domainmodell:

Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Diagramm, Screenshot, Plan enthält.

Automatisch generierte BeschreibungDeployment View:

# Logical View:



# Teststrategie:

Unit Tests:

Testen jeder Methode in den Controllern, die mit der Businesslogic interagiert.

Testen der Repository-Klassen, um sicherzustellen, dass die Datenbankzugriffe korrekt funktionieren.

Integration Tests:

Testen der Interaktionen zwischen Controllern und Services. Testen ob Controller korrekte Daten über die REST API an den Web-Client weitergeben.

Mocking:

Verwendung von Mocking-Frameworks wie Moq, um Abhängigkeiten zu simulieren und isolierte Tests durchzuführen.

Automatisierte Builds und Tests:

Integrieren von Tests in eine Build-Pipeline, um sicherzustellen, dass Tests bei jedem Build automatisch ausgeführt werden.