**Architekturspezifikation**

**Architekturüberblick:**

Server kann Linux oder Windows Server als Betriebssystem verwenden.

Server kann redundant konfiguriert werden. Der Backup-Server muss über Eigenschaften verfügen, die denen des Master-Servers entsprechen. Auf diese Weise kann bei Ausfall des Hauptservers der Backup-Server einspringen und den unterbrechungsfreien Betrieb der Website sicherstellen.

Server werden von einer Serververwaltungssoftware verwaltet und fernüberwacht.

**Definiere in Controller den API:**

Der Controller kann Anfragen zum Reservieren von Fahrzeugen entgegennehmen und validieren, ob das angefragte Fahrzeug verfügbar ist und ob der Benutzer über ausreichende Rechte verfügt, um das Fahrzeug zu reservieren.

Der Controller kann Abfragen zur Verfügbarkeit von Fahrzeugen entgegennehmen und die Ergebnisse in einer lesbaren Form zurückgeben.

Der Controller kann Anfragen zur Berechnung von Mietpreisen entgegennehmen und die Preise auf der Grundlage von Kriterien wie der Mietdauer, dem Abhol- und Rückgabeort und dem Fahrzeugtyp berechnen.

Der Controller kann Anfragen zur Verwaltung von Fahrzeugen entgegennehmen, wie beispielsweise zur Hinzufügung neuer Fahrzeuge, zur Aktualisierung von Fahrzeuginformationen oder zur Löschung von Fahrzeugen.

Der Controller kann Anfragen zur Verwaltung von Benutzerkonten entgegennehmen, wie beispielsweise zur Registrierung neuer Benutzer, zur Aktualisierung von Benutzerinformationen oder zur Löschung von Benutzerkonten.

**Definiere in service die geschäftslogik:**

Der Service kann eingehende Anfragen validieren, um sicherzustellen, dass die angegebenen Parameter gültig sind und dass der Benutzer über die erforderlichen Berechtigungen verfügt, um auf die gewünschten Ressourcen zuzugreifen. Die Verfügbarkeit von Fahrzeugen für die angeforderten Zeiträume überprüft, dass keine Überschneidungen zwischen den Reservierungen bestehen.

Der Service kann die Mietpreise auf der Grundlage von Kriterien wie der Mietdauer, dem Abhol- und Rückgabeort und dem Fahrzeugtyp berechnen.

Der Service kann die Datenbank aktualisieren, um Änderungen an Reservierungen, Fahrzeugen oder Benutzerkonten zu speichern, außerdem kann mit externen Diensten wie Zahlungs- oder Buchhaltungsdiensten integriert werden, um die Abwicklung von Transaktionen zu unterstützen.

**Versorge in Repository die Daten für die Geschäftslogik:**

Das Repository kann Abfragen zum Abrufen von Informationen über die verfügbaren Fahrzeuge entgegennehmen und die Ergebnisse in einer lesbaren Form zurückgeben, Reservierungen von Benutzern speichern und sicherstellen, dass die Daten konsistent und vollständig sind, außerdem Abfragen zum Abrufen von Informationen über Benutzerkonten entgegennehmen und die Ergebnisse in einer lesbaren Form zurückgeben.

Das Repository kann die Datenbank aktualisieren, um neue Fahrzeuge hinzuzufügen, bestehende Fahrzeuginformationen zu aktualisieren oder nicht mehr verfügbare Fahrzeuge zu löschen, die Datenbank aktualisieren, um neue Benutzerkonten hinzuzufügen, bestehende Benutzerinformationen zu aktualisieren oder nicht mehr benötigte Benutzerkonten zu löschen.

**Beschreibung der „Controller” Klassen**

**@RestController**

**@RequestMapping("/api/rentals")**

**public class RentalController {**

**@Autowired**

**private RentalService rentalService;**

**@PostMapping("/")**

**public ResponseEntity<?> createRental(@RequestBody Rental rental) {**

**try {**

**Rental createdRental = rentalService.createRental(rental);**

**return new ResponseEntity<>(createdRental, HttpStatus.CREATED);**

**} catch (Exception e) {**

**return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);**

**}**

**}**

**@GetMapping("/{id}")**

**public ResponseEntity<?> getRentalById(@PathVariable("id") Long id) {**

**try {**

**Rental rental = rentalService.getRentalById(id);**

**return new ResponseEntity<>(rental, HttpStatus.OK);**

**} catch (Exception e) {**

**return new ResponseEntity<>(e.getMessage(), HttpStatus.NOT\_FOUND);**

**}**

**}**

**@GetMapping("/")**

**public ResponseEntity<?> getAllRentals() {**

**try {**

**List<Rental> rentals = rentalService.getAllRentals();**

**return new ResponseEntity<>(rentals, HttpStatus.OK);**

**} catch (Exception e) {**

**return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);**

**}**

**}**

**@PutMapping("/{id}")**

**public ResponseEntity<?> updateRental(@PathVariable("id") Long id, @RequestBody Rental rental) {**

**try {**

**Rental updatedRental = rentalService.updateRental(id, rental);**

**return new ResponseEntity<>(updatedRental, HttpStatus.OK);**

**} catch (Exception e) {**

**return new ResponseEntity<>(e.getMessage(), HttpStatus.NOT\_FOUND);**

**}**

**}**

**@DeleteMapping("/{id}")**

**public ResponseEntity<?> deleteRental(@PathVariable("id") Long id) {**

**try {**

**rentalService.deleteRental(id);**

**return new ResponseEntity<>(HttpStatus.NO\_CONTENT);**

**} catch (Exception e) {**

**return new ResponseEntity<>(e.getMessage(), HttpStatus.NOT\_FOUND);**

**}**

**}**

**}**

**Zuerst wird die RentalController-Klasse mit den Annotationen @RestController und @RequestMapping definiert. Die Annotation @RestController gibt an, dass die Klasse ein RESTful-Webdienst ist. Die Annotation @RequestMapping bestimmt, auf welchem ​​Pfad diese Klasse ausgeführt wird. Hier wird ein Webdienst definiert, der auf dem Pfad /api/rentals ausgeführt wird.**

**Die Klasse wird von einer Dienstklasse namens RentalService verwendet. Daher wird die rentalService-Instanz mit der Annotation @Autowired eingefügt. Als Nächstes wird die createRental-Methode für die HTTP-POST-Methode definiert. Diese Methode verarbeitet die eingehende Anfrage und erstellt ein Mietobjekt durch RentalService und gibt das generierte Objekt als Antwort zurück.**

**Ebenso werden separate Methoden für andere HTTP-Methoden definiert. Mit der Annotation @PathVariable ist es möglich, in der URL angegebene Variablen als Parameter an die Methode zu übergeben. Schließlich wird für jede Methode eine entsprechende ResponseEntity zurückgegeben.**

**Rückverfolgbarkeit der Anforderungen:**

* **Der Benutzer möchte ein Auto mieten.**
* **Sie können einen neuen Mietvertrag mit der Methode POST /api/rentals erstellen.**
* **Der Benutzer möchte Mietwageninformationen anzeigen.**
* **Es kann eine bestimmte Miettransaktion mit der Methode GET /api/rentals/{id} anzeigen.**
* **Der Benutzer möchte alle Mietwageninformationen anzeigen.**
* **Es kann die Liste aller Miettransaktionen mit der Methode GET /api/rentals anzeigen.**
* **Der Benutzer möchte den Mietprozess aktualisieren.**
* **PUT kann eine bestimmte Lease mithilfe der /api/rentals/{id}-Methode aktualisieren.**
* **Der Benutzer möchte eine Mietwageninformation löschen.**
* **Es kann eine bestimmte Anmietung mit der Methode DELETE /api/rentals/{id} löschen.**

**Beschreibung der DB-Zugriffsschicht(Daten-Modelle):**

**Fahrzeugdaten: Zu diesen Daten gehören Informationen wie Fahrzeugmarke, Modell, Baujahr, Kraftstoffart, Gangart, Motorleistung, Kraftstoffverbrauch.**

**Kundendaten: Zu diesen Daten gehören persönliche Daten des Kunden wie Name, Nachname, Geburtsdatum, Adresse, Telefonnummer und E-Mail-Adresse.**

**Mietdaten: Diese Daten beziehen sich auf die Anmietung eines Autos durch einen Kunden und beinhalten Informationen über Anmietdatum, Rückgabedatum, Mietwagen und Kunde.**

**Zahlungsdaten: Diese Daten beziehen sich auf die Zahlung für ein Leasing und beinhalten Informationen wie Zahlungsdatum, Zahlungsbetrag, Zahlungsmethode und Zahlungsstatus.**

**RentalRepository Klass:**

**import org.springframework.data.jpa.repository.JpaRepository;**

**import java.util.Date;**

**import java.util.List;**

**public interface RentalRepository extends JpaRepository<Rental, Long> {**

**List<Rental> findByCustomerId(Long customerId);**

**List<Rental> findByCarId(Long carId);**

**List<Rental> findByStartDateBetween(Date startDate, Date endDate);**

**}**

**CarRepository Klass:**

**public interface CarRepository extends JpaRepository<Car, Long> {**

**List<Car> findByBrand(String brand);**

**List<Car> findByFuelType(String fuelType);**

**List<Car> findByProductionYearBetween(int startYear, int endYear);**

**}**

**CustomerRepository Klass:**

**public interface CustomerRepository extends JpaRepository<Customer, Long> {**

**Customer findByEmail(String email);**

**List<Customer> findByAddressContainingIgnoreCase(String keyword);**

**List<Customer> findByBirthDateBetween(LocalDate start, LocalDate end);**

**}**

**PaymentRepository Klass:**

**public interface PaymentRepository extends JpaRepository<Payment, Long> {**

**List<Payment> findByPaymentDateBetween(LocalDateTime start, LocalDateTime end);**

**List<Payment> findByPaymentMethod(String method);**

**List<Payment> findByPaymentStatus(PaymentStatus status);**

**}**