**Deskripsi Proyek**

API ini dirancang untuk mengelola proses penyewaan kendaraan, termasuk operasi CRUD (Create, Read, Update, Delete) untuk entitas seperti kendaraan, pelanggan, dan transaksi penyewaan. Proyek ini menggunakan Java dengan framework Spring Boot dan manajemen dependensi menggunakan Maven.

**Struktur Proyek**

Repositori ini memiliki struktur direktori sebagai berikut:

Rental\_api/

├── .mvn/ # Wrapper Maven

├── db/ # Skrip dan konfigurasi basis data

├── src/ # Kode sumber aplikasi

│ └── main/ # Kode sumber utama

├── target-dist/ # Hasil build distribusi

├── .gitattributes # Konfigurasi atribut Git

├── .gitignore # Daftar file yang diabaikan oleh Git

├── README.md # Dokumentasi proyek

├── mvnw # Skrip Maven wrapper untuk Linux/Mac

├── mvnw.cmd # Skrip Maven wrapper untuk Windows

└── pom.xml # File konfigurasi Maven

**Database**

**1. Tabel data\_mobil**

| **Kolom** | **Tipe Data** | **Keterangan** |
| --- | --- | --- |
| id\_mobil | int, AUTO\_INCREMENT, PRIMARY KEY | ID unik untuk setiap mobil |
| merk | varchar(255) | Merek mobil (contoh: Toyota, Honda) |
| model | varchar(255) | Model mobil |
| tahun | int | Tahun pembuatan mobil |
| plat\_nomor | varchar(255) | Nomor plat mobil (unik) |
| harga\_sewa\_per\_hari | double | Harga sewa mobil per hari |
| status | enum('tersedia','disewa') | Status mobil, default 'tersedia' |

**2. Tabel data\_pelanggan**

| **Kolom** | **Tipe Data** | **Keterangan** |
| --- | --- | --- |
| id\_pelanggan | int, AUTO\_INCREMENT, PRIMARY KEY | ID unik pelanggan |
| nama | varchar(255) | Nama pelanggan |
| alamat | varchar(255), NULLABLE | Alamat pelanggan |
| no\_telepon | varchar(255), NULLABLE | Nomor telepon pelanggan |
| email | varchar(255), NULLABLE, UNIQUE | Email pelanggan (unik) |

**3. Tabel data\_rental**

| **Kolom** | **Tipe Data** | **Keterangan** |
| --- | --- | --- |
| id\_rental | int, AUTO\_INCREMENT, PRIMARY KEY | ID unik transaksi rental |
| id\_transaksi | int | Foreign key ke data\_transaksi.id\_transaksi |
| id\_mobil | int | Foreign key ke data\_mobil.id\_mobil |
| id\_pelanggan | int | Foreign key ke data\_pelanggan.id\_pelanggan |
| tanggal\_mulai | date | Tanggal mulai rental |
| tanggal\_batas | date | Batas tanggal rental |

**4. Tabel data\_transaksi**

| **Kolom** | **Tipe Data** | **Keterangan** |
| --- | --- | --- |
| id\_transaksi | int, AUTO\_INCREMENT, PRIMARY KEY | ID unik transaksi |
| id\_pelanggan | int | Foreign key ke data\_pelanggan.id\_pelanggan |
| id\_mobil | int | Foreign key ke data\_mobil.id\_mobil |
| tanggal\_mulai | date | Tanggal mulai transaksi |
| tanggal\_selesai | date | Tanggal selesai transaksi |
| total\_harga | string | Total harga |
| denda | string, nullable | Denda keterlambatan |
| status | enum('berlangsung','selesai'), default 'berlangsung' | Status transaksi |

**5. Tabel user**

| **Kolom** | **Tipe Data** | **Keterangan** |
| --- | --- | --- |
| id\_user | int, AUTO\_INCREMENT, PRIMARY KEY | ID unik user |
| username | varchar(255), UNIQUE | Username login |
| password | varchar(255) | Password |
| name | varchar(255) | Nama user |

**Menjalankan Proyek**

**Prasyarat**

* Java Development Kit (JDK) versi 11 atau lebih tinggi
* Apache Maven versi 3.6 atau lebih tinggi
* Database MySQL
* Framework Spring Boot

**Langkah-langkah**

1. **Clone repositori:**

git clone https://github.com/Budhycc/Rental\_api.git

cd Rental\_api

1. **Konfigurasi basis data:**

* Import Database ke MySQL
* Sesuaikan pengaturan koneksi basis data di file src/main/resources/application.properties

1. **Jalankan** **aplikasi**

Buka direktori proyek di terminal, lalu jalankan perintah ‘mvn spring-boot:run’

1. **Build aplikasi:**Buka terminal di direktori proyek dan jalankan perintah ‘mvn clean package’  
     
   Jika selesai, pergi ke folder target dan buka terminal dan jalankan perintah ‘java -jar api-0.0.1-SNAPSHOT.jar’
2. **Akses API:**

API dapat diakses melalui http://localhost:8080/api/{endpoint}.

**Endpoint API**

API ini menyediakan berbagai endpoint untuk mengelola data

1. Endpoint mobil

* **GET** /api/mobil: Mengambil daftar semua kendaraan.
* **GET** /api/mobil/{id}: Mengambil detail kendaraan berdasarkan ID.
* **POST** /api/mobil: Menambahkan kendaraan baru.
* **PUT** /api/mobil/{id}: Memperbarui informasi kendaraan berdasarkan ID.
* **DELETE** /api/mobil/{id}: Menghapus kendaraan berdasarkan ID.

1. Endpoint pelanggan

* **GET** /api/pelanggan: Mengambil daftar semua pelanggan.
* **GET** /api/pelanggan/{id}: Mengambil detail pelanggan berdasarkan ID.
* **POST** /api/pelanggan: Menambahkan pelanggan baru.
* **PUT** /api/pelanggan/{id}: Memperbarui informasi pelanggan berdasarkan ID.
* **DELETE** /api/pelanggan/{id}: Menghapus pelanggan berdasarkan ID.

1. Endpoint transaksi

* **GET** /api/transaksi: Mengambil daftar semua transaksi.
* **GET** /api/transaksi/{id}: Mengambil detail transaksi berdasarkan ID.
* **POST** /api/transaksi: Menambahkan transaksi baru.
* **PUT** /api/transaksi/{id}: Memperbarui informasi transaksi berdasarkan ID.
* **DELETE** /api/transaksi/{id}: Menghapus transaksi berdasarkan ID.

1. Endpoint user

* **GET** /api/user: Mengambil daftar semua user.
* **GET** /api/user/{id}: Mengambil detail user berdasarkan ID.
* **POST** /api/user: Menambahkan user baru.
* **PUT** /api/user/{id}: Memperbarui informasi user berdasarkan ID.
* **DELETE** /api/user/{id}: Menghapus user berdasarkan ID.

### Penjelasan fungsi kode di src/main/java (struktur Spring Boot):

#### 1. **ApiApplication.java**

**Lokasi**

src/main/java/com/rental/api/ApiApplication.java

**Fungsi Utama**

File ini adalah **titik awal (entry point)** dari aplikasi Spring Boot. Saat aplikasi dijalankan, file inilah yang pertama kali dieksekusi.

**Penjelasan Kode**

package com.rental.api;

* Menentukan **package** tempat file ini berada.
* Ini membantu dalam pengorganisasian kode dan dependency injection Spring.

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

* Mengimpor class dari Spring Boot:
  + SpringApplication digunakan untuk menjalankan aplikasi Spring.
  + @SpringBootApplication adalah anotasi utama untuk mengaktifkan fitur Spring Boot (component scanning, auto-configuration, dan configuration).

@SpringBootApplication

public class ApiApplication {

* @SpringBootApplication merupakan shortcut untuk tiga anotasi:
  + @Configuration: Menyatakan bahwa class ini mengandung konfigurasi Spring.
  + @EnableAutoConfiguration: Mengaktifkan konfigurasi otomatis berdasarkan dependensi yang ada.
  + @ComponentScan: Memberitahu Spring untuk memindai package ini dan sub-package-nya untuk komponen (controller, service, dll).

public static void main(String[] args) {

SpringApplication.run(ApiApplication.class, args);

System.out.println("SERVER RUNNING");

}

* Ini adalah metode main, titik awal eksekusi Java standar.
* SpringApplication.run() akan:
  + Menginisialisasi konteks Spring.
  + Memulai embedded server (seperti Tomcat).
  + Menjalankan seluruh aplikasi.
* System.out.println("SERVER RUNNING"); hanya mencetak pesan ke konsol saat server berhasil dijalankan.

**Output Saat Dijalankan**

Jika berhasil, di terminal/konsol akan muncul pesan seperti:

SERVER RUNNING

Menandakan bahwa server Spring Boot sudah aktif dan siap menerima request.

#### 2. **Model entity class (**src/main/java/com/rental/api/model**)**

**a. mobil.java**

**Lokasi**

src/main/java/com/rental/api/model/mobil.java

**Fungsi Utama**

File ini adalah **model/entity class** yang merepresentasikan tabel data\_mobil di database. Spring Boot (dengan JPA) menggunakan class ini untuk melakukan pemetaan data dari tabel ke objek Java dan sebaliknya.

**Penjelasan Kode**

**Deklarasi Package & Import**

package com.rental.api.model;

import jakarta.persistence.\*;

* Menentukan package dari class ini.
* Mengimpor anotasi dari Jakarta Persistence API (jakarta.persistence) seperti @Entity, @Table, @Id, dll., yang digunakan untuk pemetaan objek-ke-tabel (ORM - Object Relational Mapping).

**Anotasi Kelas**

@Entity

@Table(name = "data\_mobil")

public class mobil {

* @Entity: menandakan bahwa class ini adalah entity JPA (akan dipetakan ke tabel DB).
* @Table(name = "data\_mobil"): menandakan class ini dipetakan ke tabel data\_mobil.
* mobil: nama class-nya.

**Field (Kolom di DB)**

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int idMobil;

* @Id: primary key dari tabel.
* @GeneratedValue(...): nilai ID akan dibuat otomatis oleh database (auto increment).

@Column(nullable = false)

private String merk;

private String model;

private int tahun;

* Kolom wajib diisi (nullable = false).
* Mewakili merek mobil, model, dan tahun pembuatan.

@Column(nullable = false, unique = true)

private String platNomor;

* Plat nomor harus **unik** dan wajib diisi. Ini berguna untuk identifikasi mobil secara unik.

@Column(nullable = false)

private String hargaSewaPerHari;

* Harga sewa mobil per hari. Bertipe String, walau seharusnya bisa juga berupa BigDecimal jika ingin lebih presisi (untuk data keuangan).

@Enumerated(EnumType.STRING)

@Column(nullable = false)

private StatusMobil status = StatusMobil.tersedia;

* @Enumerated(EnumType.STRING): menyimpan enum sebagai **teks string** di database.
* Status default mobil adalah tersedia.

**Enum StatusMobil**

public enum StatusMobil {

tersedia,

disewa

}

* Enum yang mendefinisikan status mobil:
  + tersedia: mobil bisa disewa.
  + disewa: mobil sedang dipinjam.

**Getter & Setter**

// getIdMobil(), setMerk(), dst...

* Method untuk membaca (get) dan mengubah (set) nilai tiap properti.
* Spring/JPA menggunakan ini untuk mapping data dari/ke objek.

**b. Pelanggan.java**

**Lokasi**

src/main/java/com/rental/api/model/Pelanggan.java

**Fungsi Utama**

Class ini merepresentasikan entitas **pelanggan** pada sistem rental. Class ini akan dipetakan oleh JPA ke tabel data\_pelanggan di database.

**Penjelasan Kode**

**Deklarasi Package & Import**

package com.rental.api.model;

import jakarta.persistence.\*;

* package: menunjukkan bahwa file ini bagian dari model/data di aplikasi.
* jakarta.persistence.\*: digunakan untuk membuat entity JPA yang akan disimpan di database.

**Anotasi Kelas**

@Entity

@Table(name = "data\_pelanggan")

public class Pelanggan {

* @Entity: mendeklarasikan bahwa class ini adalah entity JPA.
* @Table(name = "data\_pelanggan"): nama tabel di database yang dipetakan ke class ini.

**Field (Properti) dan Anotasinya**

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int idPelanggan;

* @Id: menandakan bahwa idPelanggan adalah primary key.
* @GeneratedValue(...): nilainya akan digenerate otomatis oleh database (auto increment).

@Column(nullable = false)

private String nama;

* Kolom nama pelanggan, wajib diisi (nullable = false).

private String alamat;

* Kolom alamat, bersifat opsional karena tidak diberi nullable = false.

@Column(name = "no\_telepon")

private String noTelepon;

* Kolom nomor telepon, diberi nama kolom eksplisit no\_telepon di tabel.

@Column(unique = true)

private String email;

* Kolom email pelanggan, harus **unik** di database (tidak boleh ada duplikat).

**Getter dan Setter**

public String getNama() { return nama; }

public void setNama(String nama) { this.nama = nama; }

// dan seterusnya...

* Digunakan oleh Spring dan JPA untuk membaca dan mengubah nilai property.
* Membantu framework dalam melakukan serialisasi data ke JSON dan vice versa.

## c. transaksi.java

### Lokasi

src/main/java/com/rental/api/model/transaksi.java

## Fungsi Utama

Class ini adalah **entity JPA** yang memetakan data dari tabel data\_transaksi di database. Entitas ini menyimpan informasi tentang **transaksi penyewaan mobil** antara pelanggan dan sistem rental.

## Penjelasan Kode

### Package dan Import

package com.rental.api.model;

import jakarta.persistence.\*;

import java.util.Date;

* Mengatur package dan mengimpor anotasi untuk mapping JPA (@Entity, @ManyToOne, dll.).
* Date digunakan untuk tanggal mulai dan selesai sewa.

### Anotasi Class

@Entity

@Table(name = "data\_transaksi")

public class transaksi {

* @Entity: menunjukkan bahwa class ini adalah entity/tabel.
* @Table(name = "data\_transaksi"): mengaitkan entity ini dengan tabel data\_transaksi di database.

### Field & Mapping

#### Primary Key

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int idTransaksi;

* @Id: menandakan idTransaksi sebagai primary key.
* @GeneratedValue: auto increment di database.

#### Relasi ke Pelanggan

@ManyToOne

@JoinColumn(name = "id\_pelanggan", nullable = false)

private Pelanggan pelanggan;

* Banyak transaksi bisa dimiliki satu pelanggan.
* Relasi ManyToOne ke entity Pelanggan.

#### Relasi ke Mobil

@ManyToOne

@JoinColumn(name = "id\_mobil", nullable = false)

private mobil mobil;

* Banyak transaksi bisa melibatkan satu mobil.
* Relasi ManyToOne ke entity mobil.

#### Tanggal Transaksi

@Temporal(TemporalType.DATE)

@Column(nullable = false)

private Date tanggalMulai;

@Temporal(TemporalType.DATE)

@Column(nullable = false)

private Date tanggalSelesai;

* Menyimpan tanggal mulai dan selesai transaksi.
* @Temporal(TemporalType.DATE) memastikan hanya tanggal (tanpa jam) yang disimpan.

#### Harga & Denda

@Column(nullable = false)

private String totalHarga;

@Column

private String denda;

* totalHarga: harga total sewa mobil (wajib diisi).
* denda: opsional, untuk denda keterlambatan jika ada.

#### Status Transaksi

@Enumerated(EnumType.STRING)

@Column(nullable = false)

private StatusTransaksi status = StatusTransaksi.berlangsung;

* @Enumerated(EnumType.STRING): enum disimpan dalam bentuk string di database.
* Default status adalah berlangsung.

### Enum: StatusTransaksi

public enum StatusTransaksi {

berlangsung,

selesai

}

* Menyatakan status transaksi:
  + berlangsung: penyewaan sedang berjalan.
  + selesai: penyewaan sudah selesai.

### Getter dan Setter

// getIdTransaksi(), setMobil(), getTanggalMulai(), dll.

* Digunakan oleh JPA dan framework Spring untuk akses/modifikasi data.

**d. user.java**

**Lokasi**

src/main/java/com/rental/api/model/user.java

**Fungsi Utama**

Class ini merupakan entity JPA yang memetakan data ke tabel **user** dalam database. Entitas ini digunakan untuk menyimpan informasi pengguna sistem (misalnya admin atau user aplikasi rental mobil).

**Penjelasan Kode**

**Package dan Import**

package com.rental.api.model;

import jakarta.persistence.\*;

* Mendefinisikan package dan mengimpor anotasi JPA untuk pemetaan ke database.

**Anotasi Kelas**

@Entity

@Table(name = "user")

public class user {

* @Entity: Menandakan bahwa ini adalah kelas entity untuk database.
* @Table(name = "user"): Memetakan kelas ini ke tabel user di database.

**Field dan Anotasi**

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id\_user;

* @Id: Primary key dari tabel user.
* @GeneratedValue: Menggunakan auto-increment dari database.

@Column(nullable = false, unique = true)

private String username;

* Wajib diisi (nullable = false) dan tidak boleh duplikat (unique = true).

private String password;

private String name;

* Kolom password dan name tidak diberi batasan eksplisit.
* Secara keamanan.

**Getter dan Setter**

public int getIdUser() { return id\_user; }

public void setPassword(String password) { this.password = password; }

// dan seterusnya...

* Getter/setter digunakan oleh framework untuk manipulasi dan serialisasi data.

#### 3. **Repository interface (**src/main/java/com/rental/api/repository**)**

**a. mobilRepository.java**

**Lokasi**

src/main/java/com/rental/api/repository/mobilRepository.java

**Fungsi Utama**

Interface ini adalah **repository** yang digunakan untuk mengakses dan memanipulasi data entitas mobil di database.  
mobilRepository memanfaatkan Spring Data JPA agar developer tidak perlu menulis query SQL manual.

**Penjelasan Kode**

package com.rental.api.repository;

import com.rental.api.model.mobil;

import java.util.Optional;

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

**Penjelasan:**

* mobil: Entitas mobil yang akan diakses oleh repository.
* JpaRepository: Interface dari Spring Data JPA yang menyediakan fungsi CRUD secara otomatis.
  + Generic: <mobil, Integer> berarti entitas mobil dengan primary key bertipe Integer.

**Interface Repository**

public interface mobilRepository extends JpaRepository<mobil, Integer> {

Optional<mobil> findByPlatNomor(String platNomor);

}

* mobilRepository: Interface yang otomatis dibuatkan implementasinya oleh Spring Boot.
* Mewarisi semua method dari JpaRepository, seperti:
  + findAll()
  + findById(Integer id)
  + save(mobil entity)
  + deleteById(Integer id)

**Method Khusus**

Optional<mobil> findByPlatNomor(String platNomor);

* Method ini adalah **query method** otomatis dari Spring Data JPA.
* Akan menghasilkan query:
* SELECT \* FROM data\_mobil WHERE plat\_nomor = ?
* Return type Optional<mobil>:
  + Menghindari NullPointerException jika data tidak ditemukan.
  + Digunakan untuk pengecekan kondisi ada/tidaknya mobil berdasarkan platNomor.

**Kegunaan Umum**

* Mengecek apakah mobil dengan plat tertentu sudah ada di sistem.
* Menyediakan fitur pencarian mobil berdasarkan plat nomor.
* Digunakan oleh service/controller untuk proses transaksi atau validasi data.

**b. PelangganRepository.java**

**Lokasi**

src/main/java/com/rental/api/repository/PelangganRepository.java

**Fungsi Utama**

Interface ini merupakan **repository** untuk entitas Pelanggan.  
Digunakan untuk melakukan operasi CRUD (Create, Read, Update, Delete) terhadap data pelanggan dalam database, tanpa perlu menulis query SQL manual.

**Penjelasan Kode**

package com.rental.api.repository;

import com.rental.api.model.Pelanggan;

import java.util.Optional;

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

**Keterangan:**

* Pelanggan: Class entitas yang dipetakan ke tabel data\_pelanggan.
* JpaRepository: Interface dari Spring Data JPA yang menyediakan berbagai fungsi bawaan untuk manipulasi data.
* Optional: Digunakan untuk menangani hasil pencarian yang bisa saja null (tidak ditemukan).

**Interface Repository**

public interface PelangganRepository extends JpaRepository<Pelanggan, Integer> {

Optional<Pelanggan> findByEmail(String email);

}

* PelangganRepository adalah interface yang secara otomatis diimplementasikan oleh Spring Boot.
* JpaRepository<Pelanggan, Integer>:
  + Pelanggan: Entitas yang dikelola.
  + Integer: Tipe data dari primary key (idPelanggan).

**Method Kustom**

Optional<Pelanggan> findByEmail(String email);

* Query method otomatis yang menghasilkan query:
* SELECT \* FROM data\_pelanggan WHERE email = ?
* Berguna untuk:
  + Mengecek apakah pelanggan dengan email tertentu sudah terdaftar.
  + Mencari data pelanggan secara unik berdasarkan email.

**Fungsi Bawaan dari JpaRepository**

| **Method** | **Fungsi** |
| --- | --- |
| findAll() | Mengambil semua data transaksi |
| findById(id) | Mengambil transaksi berdasarkan ID |
| save(obj) | Menyimpan transaksi baru atau memperbarui yang lama |
| deleteById(id) | Menghapus transaksi berdasarkan ID |
| count() | Menghitung jumlah transaksi |

**Kegunaan Umum**

* Mendaftarkan pelanggan baru (cek apakah email sudah digunakan).
* Mengambil data pelanggan berdasarkan email (untuk login, profil, dll).
* Menyediakan akses data ke layer service atau controller.

**c. transaksiRepository.java**

**Lokasi**

src/main/java/com/rental/api/repository/transaksiRepository.java

**Fungsi Utama**

Interface ini adalah **repository** untuk entitas transaksi.  
Berfungsi sebagai jembatan antara aplikasi dan database untuk memanipulasi data transaksi (menyewa dan mengembalikan mobil).

**Penjelasan Kode**

package com.rental.api.repository;

import com.rental.api.model.transaksi;

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

* transaksi: Class entity yang memetakan tabel data\_transaksi di database.
* JpaRepository: Interface dari Spring Data JPA yang menyediakan operasi database secara otomatis.

**Interface Repository**

public interface transaksiRepository extends JpaRepository<transaksi, Integer> {

}

* transaksiRepository: Interface ini akan otomatis diimplementasikan oleh Spring.
* JpaRepository<transaksi, Integer>:
  + transaksi: Entitas yang dikelola.
  + Integer: Tipe dari primary key (idTransaksi).

**Fungsi Bawaan dari JpaRepository**

| **Method** | **Fungsi** |
| --- | --- |
| findAll() | Mengambil semua data transaksi |
| findById(id) | Mengambil transaksi berdasarkan ID |
| save(obj) | Menyimpan transaksi baru atau memperbarui yang lama |
| deleteById(id) | Menghapus transaksi berdasarkan ID |
| count() | Menghitung jumlah transaksi |

**Kegunaan Umum**

* Digunakan oleh service dan controller untuk menyimpan, mengambil, dan menghapus transaksi.
* Digunakan dalam proses:
  + Penyewaan mobil.
  + Pengembalian mobil.
  + Riwayat transaksi pelanggan.
  + Laporan transaksi.

**d. userRepository.java**

**Lokasi**

src/main/java/com/rental/api/repository/userRepository.java

**Fungsi Utama**

Interface ini digunakan untuk **mengelola data user/admin** di aplikasi rental mobil.  
Berbasis Spring Data JPA, userRepository memungkinkan akses ke data pengguna tanpa perlu menulis SQL secara manual.

**Penjelasan Kode**

package com.rental.api.repository;

import com.rental.api.model.user;

import java.util.Optional;

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

* user: Entitas pengguna (admin/operator) dalam sistem.
* JpaRepository<user, Integer>: Menyediakan method CRUD standar untuk entitas user dengan Integer sebagai tipe ID.
* Optional: Digunakan untuk menangani kemungkinan data tidak ditemukan (null).

**Interface Repository**

public interface userRepository extends JpaRepository<user, Integer> {

Optional<user> findByUsername(String username);

}

* userRepository: Interface yang diimplementasikan secara otomatis oleh Spring Boot.
* findByUsername(String username):
  + Metode query otomatis berdasarkan nama field username pada entitas user.
  + Akan menghasilkan query:
  + SELECT \* FROM user WHERE username = ?

**Fungsi Bawaan dari JpaRepository**

| **Method** | **Fungsi** |
| --- | --- |
| findAll() | Mengambil semua data transaksi |
| findById(id) | Mengambil transaksi berdasarkan ID |
| save(obj) | Menyimpan transaksi baru atau memperbarui yang lama |
| deleteById(id) | Menghapus transaksi berdasarkan ID |
| count() | Menghitung jumlah transaksi |

**Kegunaan Umum**

* **Login Autentikasi**: findByUsername sering digunakan untuk mengecek apakah user ada saat login.
* **Manajemen Admin**: Menambah, mengedit, atau menghapus data admin/operator aplikasi.
* **Keamanan Akses**: Digunakan dalam integrasi dengan Spring Security (jika diterapkan).

#### 4. **Service class (**src/main/java/com/rental/api/service**)**

**a. mobilService.java**

**Lokasi**

src/main/java/com/rental/api/service/mobilService.java

**Fungsi Utama**

mobilService adalah komponen **layanan bisnis (service layer)** yang mengelola logika aplikasi terkait entitas mobil.  
Service ini berfungsi sebagai perantara antara controller dan repository.

**Penjelasan Kode**

@Service

public class mobilService {

* Memberi tahu Spring bahwa ini adalah **bean service** dan akan otomatis dikelola (@Component berbasis @Service).

**Dependency Injection**

@Autowired

private mobilRepository repository;

* Spring secara otomatis menyuntikkan (inject) implementasi dari mobilRepository.

**Method: getAllMobil()**

public List<mobil> getAllMobil() {

return repository.findAll();

}

* Mengambil semua data mobil dari database.

**Method: saveMobil(mobil mobil)**

public mobil saveMobil(mobil mobil) {

if (repository.findByPlatNomor(mobil.getPlatNomor()).isPresent()) {

throw new IllegalArgumentException("Plat nomor sudah digunakan!");

}

return repository.save(mobil);

}

* Menyimpan data mobil baru.
* Validasi plat nomor agar **tidak duplikat**.
* Lempar exception jika plat nomor sudah digunakan.

**Method: updateMobil(int id, mobil mobilDetails)**

public mobil updateMobil(int id, mobil mobilDetails) {

return repository.findById(id).map(existingMobil -> {

...

return repository.save(existingMobil);

}).orElseThrow(() -> new IllegalArgumentException("Mobil dengan ID " + id + " tidak ditemukan"));

}

* Update data mobil berdasarkan ID.
* Gunakan map() dari Optional untuk update hanya jika mobil ditemukan.
* Lempar exception jika ID tidak ditemukan.

**Method: getMobilById(int id)**

public Optional<mobil> getMobilById(int id) {

return repository.findById(id);

}

* Ambil data mobil berdasarkan ID.
* Mengembalikan Optional, aman terhadap null.

**Method: deleteMobil(int id)**

public void deleteMobil(int id) {

repository.deleteById(id);

}

* Hapus data mobil berdasarkan ID.

**Manfaat Service Layer**

1. **Pemisahan Tanggung Jawab (Separation of Concerns)**:
   * Logika bisnis tidak dicampur di controller.
2. **Konsistensi Validasi**:
   * Misalnya: Validasi plat nomor unik hanya dilakukan di sini.
3. **Mudah Diuji (Unit Test)**:
   * Service dapat diuji tanpa memanggil controller.

**b. PelangganService.java**

**Lokasi**

src/main/java/com/rental/api/service/PelangganService.java

**Fungsi Utama**

PelangganService bertanggung jawab mengelola logika bisnis terkait data pelanggan.  
Service ini menjadi jembatan antara controller dan repository untuk operasi CRUD data pelanggan.

**Penjelasan Kode**

@Service

public class PelangganService {

* Anotasi @Service menandakan class ini adalah komponen service Spring.

@Autowired

private PelangganRepository repository;

* Menggunakan dependency injection Spring untuk mengakses data pelanggan via repository.

**Method: getAllPelanggan()**

public List<Pelanggan> getAllPelanggan() {

return repository.findAll();

}

* Mengambil seluruh data pelanggan dari database.

**Method: savePelanggan(Pelanggan pelanggan)**

public Pelanggan savePelanggan(Pelanggan pelanggan) {

if (repository.findByEmail(pelanggan.getEmail()).isPresent()) {

throw new IllegalArgumentException("Email nomor sudah digunakan!");

}

return repository.save(pelanggan);

}

* Menyimpan data pelanggan baru.
* Mengecek apakah email pelanggan sudah terdaftar, jika ya, melempar exception.

**Method: updatePelanggan(int id, Pelanggan pelangganDetails)**

public Pelanggan updatePelanggan(int id, Pelanggan pelangganDetails) {

return repository.findById(id).map(existingPelanggan -> {

existingPelanggan.setNama(pelangganDetails.getNama());

existingPelanggan.setAlamat(pelangganDetails.getAlamat());

existingPelanggan.setNoTelepon(pelangganDetails.getNoTelepon());

existingPelanggan.setEmail(pelangganDetails.getEmail());

return repository.save(existingPelanggan);

}).orElseThrow(() -> new IllegalArgumentException("Pelanggan dengan ID " + id + " tidak ditemukan"));

}

* Memperbarui data pelanggan berdasarkan ID.
* Jika data pelanggan dengan ID tersebut tidak ditemukan, akan melempar exception.

**Method: getPelangganById(int id)**

public Optional<Pelanggan> getPelangganById(int id) {

return repository.findById(id);

}

* Mengambil data pelanggan berdasarkan ID, mengembalikan Optional untuk menangani kasus data tidak ditemukan.

**Method: deletePelanggan(int id)**

public void deletePelanggan(int id) {

repository.deleteById(id);

}

* Menghapus data pelanggan berdasarkan ID.

**Manfaat Service Layer**

* **Memisahkan logika bisnis** dari controller.
* Menjaga konsistensi validasi (misal pengecekan email unik).
* Mempermudah testing unit.

**c. transaksiService.java**

**Lokasi**

src/main/java/com/rental/api/service/transaksiService.java

**Fungsi Utama**

transaksiService adalah service yang mengelola logika bisnis terkait transaksi penyewaan mobil, menghubungkan controller dengan transaksiRepository untuk operasi CRUD data transaksi.

**Penjelasan Kode**

@Service

public class transaksiService {

* Anotasi @Service menunjukkan class ini adalah service Spring.

@Autowired

private transaksiRepository repository;

* Menggunakan dependency injection untuk mengakses data transaksi.

**Method: getAllTransaksi()**

public List<transaksi> getAllTransaksi() {

return repository.findAll();

}

* Mengambil semua data transaksi yang tersimpan di database.

**Method: saveTransaksi(transaksi transaksi)**

public transaksi saveTransaksi(transaksi transaksi) {

return repository.save(transaksi);

}

* Menyimpan data transaksi baru ke database tanpa validasi khusus.

**Method: updateTransaksi(int id, transaksi transaksiDetails)**

public transaksi updateTransaksi(int id, transaksi transaksiDetails) {

return repository.findById(id).map(existingTransaksi -> {

existingTransaksi.setPelanggan(transaksiDetails.getPelanggan());

existingTransaksi.setTanggalMulai(transaksiDetails.getTanggalMulai());

existingTransaksi.setTanggalSelesai(transaksiDetails.getTanggalSelesai());

existingTransaksi.setTotalHarga(transaksiDetails.getTotalHarga());

existingTransaksi.setDenda(transaksiDetails.getDenda());

existingTransaksi.setStatus(transaksiDetails.getStatus());

return repository.save(existingTransaksi);

}).orElseThrow(() -> new IllegalArgumentException("Transaksi dengan ID " + id + " tidak ditemukan"));

}

* Memperbarui data transaksi berdasarkan ID.
* Jika transaksi tidak ditemukan, melempar exception IllegalArgumentException.

**Method: getTransaksiById(int id)**

public Optional<transaksi> getTransaksiById(int id) {

return repository.findById(id);

}

* Mengambil data transaksi berdasarkan ID, menggunakan Optional untuk menghindari null.

**Method: deleteTransaksi(int id)**

public void deleteTransaksi(int id) {

repository.deleteById(id);

}

* Menghapus transaksi berdasarkan ID.

**d. userService.java**

**Lokasi**

src/main/java/com/rental/api/service/userService.java

**Fungsi Utama**

userService mengelola logika bisnis terkait data pengguna (user) aplikasi, menghubungkan controller dengan repository untuk melakukan operasi CRUD dan login.

**Penjelasan Kode**

@Service

public class userService {

* Anotasi @Service menandakan ini adalah komponen service Spring.

@Autowired

private userRepository repository;

* Dependency injection untuk akses data user via repository.

**Method: getAllUser()**

public List<user> getAllUser() {

return repository.findAll();

}

* Mengambil semua data user dari database.

**Method: saveUser(user user)**

public String saveUser(user user) {

try {

repository.save(user);

return "User berhasil ditambahkan!";

} catch (DataIntegrityViolationException e) {

return "Error: Username sudah digunakan!";

}

}

* Menyimpan data user baru.
* Menggunakan try-catch untuk menangani error jika username sudah ada (melanggar constraint unik).
* Mengembalikan pesan sukses atau error dalam bentuk string.

**Method: updateUser(int id, user userDetails)**

public user updateUser(int id, user userDetails) {

return repository.findById(id).map(existingUser -> {

existingUser.setUsername(userDetails.getUsername());

existingUser.setName(userDetails.getName());

existingUser.setPassword(userDetails.getPassword());

return repository.save(existingUser);

}).orElseThrow(() -> new IllegalArgumentException("User dengan ID " + id + " tidak ditemukan"));

}

* Memperbarui data user berdasarkan ID.
* Jika user tidak ditemukan, melempar exception.

**Method: getUserById(int id)**

public Optional<user> getUserById(int id) {

return repository.findById(id);

}

* Mengambil user berdasarkan ID dalam bentuk Optional.

**Method: deleteUser(int id)**

public void deleteUser(int id) {

repository.deleteById(id);

}

* Menghapus user berdasarkan ID.

**Method: login(String username, String password)**

public Optional<user> login(String username, String password) {

return repository.findByUsername(username)

.filter(user -> user.getPassword().equals(password));

}

* Mengecek validasi login dengan mencocokkan username dan password.
* Jika cocok, mengembalikan user, jika tidak cocok, Optional kosong.

#### 5. **Controller class (**src/main/java/com/rental/api/controller**)**

## a. mobilController.java

### Lokasi

src/main/java/com/rental/api/controller/mobilController.java

## Fungsi Utama

mobilController adalah kelas controller REST API yang mengatur endpoint terkait data **mobil**. Controller ini bertugas menerima request HTTP dari client, memprosesnya dengan memanggil service, lalu mengembalikan response.

## Penjelasan Kode

@RestController

@RequestMapping("/api/mobil")

public class mobilController {

* @RestController = Kombinasi @Controller dan @ResponseBody, membuat class ini jadi REST controller.
* @RequestMapping("/api/mobil") = Semua endpoint diawali dengan /api/mobil.

@Autowired

private mobilService service;

* Dependency injection service untuk mengakses logic bisnis mobil.

### Endpoint HTTP

#### 1. **GET /api/mobil**

@GetMapping

public List<mobil> getAllMobil() {

return service.getAllMobil();

}

* Mengembalikan daftar semua mobil yang ada di database.

#### 2. **POST /api/mobil**

@PostMapping

public ResponseEntity<?> registerMobil(@RequestBody mobil mobil) {

try {

mobil savedMobil = service.saveMobil(mobil);

return ResponseEntity.ok(savedMobil);

} catch (IllegalArgumentException e) {

return ResponseEntity.badRequest().body(e.getMessage());

}

}

* Menerima data mobil baru dalam body request JSON.
* Memanggil service untuk menyimpan data.
* Jika berhasil, mengembalikan data mobil baru dengan status 200 OK.
* Jika ada error (misalnya plat nomor sudah terdaftar), mengembalikan status 400 Bad Request dengan pesan error.

#### 3. **PUT /api/mobil/{id}**

@PutMapping("/{id}")

public ResponseEntity<?> updateMobil(@PathVariable int id, @RequestBody mobil mobilDetails) {

try {

mobil updatedMobil = service.updateMobil(id, mobilDetails);

return ResponseEntity.ok(updatedMobil);

} catch (IllegalArgumentException e) {

return ResponseEntity.badRequest().body(e.getMessage());

}

}

* Mengupdate data mobil berdasarkan id path parameter.
* Jika update berhasil, mengembalikan data mobil terbaru dengan status 200 OK.
* Jika gagal (misal id tidak ditemukan), mengembalikan status 400 dengan pesan error.

#### 4. **GET /api/mobil/{id}**

@GetMapping("/{id}")

public ResponseEntity<mobil> getMobilById(@PathVariable int id) {

return service.getMobilById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

* Mengambil data mobil berdasarkan id.
* Jika ditemukan, response status 200 dengan data mobil.
* Jika tidak ditemukan, response status 404 Not Found.

#### 5. **DELETE /api/mobil/{id}**

@DeleteMapping("/{id}")

public void deleteMobil(@PathVariable int id) {

service.deleteMobil(id);

}

* Menghapus mobil berdasarkan id.
* Tidak mengembalikan response khusus.

## b. PelangganController.java

### Lokasi

src/main/java/com/rental/api/controller/PelangganController.java

## Fungsi Utama

PelangganController adalah kelas controller REST API yang mengatur endpoint terkait data **pelanggan**. Controller ini bertugas menerima request HTTP dari client, memprosesnya dengan memanggil service, lalu mengembalikan response.

## Penjelasan Kode

@RestController

@RequestMapping("/api/pelanggan")

public class PelangganController {

* @RestController = Menandakan class ini sebagai REST controller.
* @RequestMapping("/api/pelanggan") = Semua endpoint diawali dengan /api/pelanggan.

@Autowired

private PelangganService service;

* Dependency injection service untuk mengakses logic bisnis pelanggan.

### Endpoint HTTP

#### 1. **GET /api/pelanggan**

@GetMapping

public List<Pelanggan> getAllPelanggan() {

return service.getAllPelanggan();

}

* Mengembalikan daftar semua pelanggan yang ada di database.

#### 2. **POST /api/pelanggan**

@PostMapping

public ResponseEntity<?> createPelanggan(@RequestBody Pelanggan pelanggan) {

try {

Pelanggan createPelanggan = service.savePelanggan(pelanggan);

return ResponseEntity.ok(createPelanggan);

} catch (IllegalArgumentException e) {

return ResponseEntity.badRequest().body(e.getMessage());

}

}

* Menerima data pelanggan baru dalam body request JSON.
* Memanggil service untuk menyimpan data.
* Jika berhasil, mengembalikan data pelanggan baru dengan status 200 OK.
* Jika ada error (misalnya email sudah terdaftar), mengembalikan status 400 Bad Request dengan pesan error.

#### 3. **PUT /api/pelanggan/{id}**

@PutMapping("/{id}")

public ResponseEntity<?> updatePelanggan(@PathVariable int id, @RequestBody Pelanggan pelangganDetails) {

try {

Pelanggan updatedPelanggan = service.updatePelanggan(id, pelangganDetails);

return ResponseEntity.ok(updatedPelanggan);

} catch (IllegalArgumentException e) {

return ResponseEntity.badRequest().body(e.getMessage());

}

}

* Mengupdate data pelanggan berdasarkan id path parameter.
* Jika update berhasil, mengembalikan data pelanggan terbaru dengan status 200 OK.
* Jika gagal (misal id tidak ditemukan), mengembalikan status 400 dengan pesan error.

#### 4. **GET /api/pelanggan/{id}**

@GetMapping("/{id}")

public ResponseEntity<Pelanggan> getPelangganById(@PathVariable int id) {

return service.getPelangganById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

* Mengambil data pelanggan berdasarkan id.
* Jika ditemukan, response status 200 dengan data pelanggan.
* Jika tidak ditemukan, response status 404 Not Found.

#### 5. **DELETE /api/pelanggan/{id}**

@DeleteMapping("/{id}")

public void deletePelanggan(@PathVariable int id) {

service.deletePelanggan(id);

}

* Menghapus pelanggan berdasarkan id.
* Tidak mengembalikan response khusus.

**c. transaksiController.java**

### Lokasi

src/main/java/com/rental/api/controller/transaksiController.java

**Fungsi utama:**

Mengatur endpoint REST API untuk resource **transaksi**, seperti mengambil data transaksi, menambah, update, dan hapus transaksi.

**Penjelasan Kode**

@RestController

@RequestMapping("/api/transaksi")

public class transaksiController {

* @RestController: Controller REST.
* Semua endpoint diawali /api/transaksi.

@Autowired

private transaksiService service;

* Menggunakan service untuk proses bisnis transaksi.

**Endpoint HTTP**

* **GET /api/transaksi**  
  Mengambil semua data transaksi.

@GetMapping

public List<transaksi> getAllTransaksi() {

return service.getAllTransaksi();

}

* **POST /api/transaksi**  
  Membuat transaksi baru, data transaksi diambil dari body request JSON.

@PostMapping

public transaksi createTransaksi(@RequestBody transaksi transaksi) {

return service.saveTransaksi(transaksi);

}

* **PUT /api/transaksi/{id}**  
  Update transaksi berdasarkan id. Jika gagal (misal id tidak ditemukan), akan mengembalikan error 400 dengan pesan.

@PutMapping("/{id}")

public ResponseEntity<?> updateTransaksi(@PathVariable int id, @RequestBody transaksi transaksiDetails) {

try {

transaksi updateTransaksi = service.updateTransaksi(id, transaksiDetails);

return ResponseEntity.ok(updateTransaksi);

} catch (IllegalArgumentException e) {

return ResponseEntity.badRequest().body(e.getMessage());

}

}

* **GET /api/transaksi/{id}**  
  Ambil transaksi berdasarkan id. Jika tidak ditemukan, response 404.

@GetMapping("/{id}")

public ResponseEntity<transaksi> getTransaksiById(@PathVariable int id) {

return service.getTransaksiById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

* **DELETE /api/transaksi/{id}**  
  Hapus transaksi berdasarkan id. Tidak mengembalikan response, bisa ditingkatkan untuk feedback pengguna.

@DeleteMapping("/{id}")

public void deleteTransaksi(@PathVariable int id) {

service.deleteTransaksi(id);

}

## b. userController.java

### Lokasi

src/main/java/com/rental/api/controller/userController.java

### Fungsi Utama:

userController adalah kelas controller yang mengelola semua request terkait **user** pada aplikasi rental mobil. Controller ini bertugas untuk menerima HTTP request dari client (misalnya frontend atau aplikasi lain), memanggil service terkait, dan mengembalikan response ke client.

### Penjelasan Kode:

* **@RestController**  
  Menandakan kelas ini sebagai REST controller, otomatis meng-handle JSON input/output.
* **@RequestMapping("/api/user")**  
  Semua endpoint dalam controller ini diawali dengan URL /api/user.
* **@Autowired private userService service;**  
  Inject service userService yang berisi logic bisnis terkait user.

### Endpoint HTTP

1. **GET /api/user**  
   Mengambil semua data user yang ada.  
   **Response:** List data user dalam format JSON, status HTTP 200 OK.
2. **POST /api/user**  
   Membuat user baru dengan data yang dikirim di body request (format JSON).  
   **Request Body:**

{

"username": "string",

"password": "string",

"name": "string"

}

**Response:**

* + Jika berhasil, mengembalikan pesan string sukses.
  + Jika username sudah dipakai, mengembalikan pesan error dan status HTTP 400 Bad Request.

1. **PUT /api/user/{id}**  
   Memperbarui data user dengan ID tertentu.  
   **Path Variable:** id (int) — ID user yang ingin diupdate.  
   **Request Body:**

{

"username": "string",

"password": "string",

"name": "string"

}

**Response:**

* + Jika berhasil, mengembalikan data user yang sudah diperbarui (JSON).
  + Jika user tidak ditemukan, mengembalikan pesan error dan status HTTP 400 Bad Request.

1. **GET /api/user/{id}**  
   Mengambil data user berdasarkan ID.  
   **Path Variable:** id (int) — ID user yang ingin diambil datanya.  
   **Response:**
   * Jika user ditemukan, mengembalikan data user (JSON) dan status 200 OK.
   * Jika tidak ditemukan, mengembalikan status 404 Not Found.
2. **DELETE /api/user/{id}**  
   Menghapus user berdasarkan ID.  
   **Path Variable:** id (int) — ID user yang ingin dihapus.  
   **Response:** Saat ini tidak mengembalikan response khusus (disarankan gunakan status 204 No Content).
3. **POST /api/user/login**  
   Endpoint untuk login user. Menerima username dan password di request body.  
   **Request Body:**

{

"username": "string",

"password": "string"

}

**Response:**

* + Jika login berhasil, mengembalikan pesan sambutan "Welcome, [nama user]!" dengan status 200 OK.
  + Jika gagal, mengembalikan pesan error "Invalid username or password." dengan status 401 Unauthorized.

1. **POST /api/user/register**  
   Endpoint untuk registrasi user baru, mirip dengan POST /api/user.  
   **Request Body:**

{

"username": "string",

"password": "string",

"name": "string"

}

**Response:**

* + Jika berhasil, mengembalikan pesan sukses dengan status 200 OK.
  + Jika username sudah ada, mengembalikan pesan error dan status 400 Bad Request.

### Penjelasan Fungsi Per Endpoint:

* **getAllUser()**  
  Mengembalikan semua data user yang ada di database melalui service getAllUser().
* **createUser()**  
  Menerima data user baru lewat request body, kemudian memanggil service saveUser(). Jika username sudah ada, akan mengembalikan pesan error.
* **updateUser(id, userDetails)**  
  Mencari user berdasarkan ID, lalu update data user tersebut. Jika user tidak ditemukan, mengembalikan error 400 dengan pesan.
* **getPelangganById(id)**  
  Mengambil user berdasarkan ID yang dikirim lewat URL. Jika ada, kembalikan data user, jika tidak ada kembalikan 404.
* **deleteUser(id)**  
  Menghapus user berdasarkan ID yang diberikan. Saat ini tidak mengembalikan response body.
* **login(loginRequest)**  
  Menerima username dan password lewat JSON, kemudian mengecek data user. Jika valid, kembalikan pesan sambutan, jika tidak valid kembalikan status 401.
* **registerUser(user)**  
  Menerima data user baru untuk pendaftaran, lalu memanggil service saveUser(). Jika username sudah ada, mengembalikan error.

#### 6. **Configuration file (**src/main/resources/application.properties**)**

* **spring.application.name=api**  
  Menentukan nama aplikasi Spring Boot, berguna untuk identifikasi terutama dalam konteks cloud atau service discovery.
* **spring.datasource.url=jdbc:mysql://localhost:3306/rental?enabledTLSProtocols=TLSv1.2**  
  URL koneksi database MySQL, di sini menunjuk ke database bernama rental pada localhost port 3306.  
  Parameter enabledTLSProtocols=TLSv1.2 mengatur protokol TLS yang digunakan untuk keamanan koneksi database.
* **spring.datasource.username=root**  
  Username untuk mengakses database MySQL.
* **spring.datasource.password=password**  
  Password untuk user database MySQL.
* **spring.datasource.type=com.zaxxer.hikari.HikariDataSource**  
  Menentukan tipe datasource yang digunakan, di sini menggunakan HikariCP sebagai connection pool yang populer karena performanya yang baik.
* **spring.datasource.hikari.minimum-idle=10**  
  Jumlah minimum koneksi idle yang disiapkan di pool connection (artinya ada 10 koneksi siap pakai walaupun sedang tidak aktif).
* **spring.datasource.hikari.maximum-pool-size=50**  
  Jumlah maksimal koneksi yang dapat dibuat pada pool, yaitu maksimal 50 koneksi secara bersamaan.
* **spring.jpa.hibernate.ddl-auto=update**  
  Hibernate akan secara otomatis memperbarui (update) skema database sesuai entity yang ada tanpa menghapus data. Ini cocok untuk development, tapi hati-hati dipakai di production.
* **spring.jpa.show-sql=true**  
  Mengaktifkan logging query SQL yang dijalankan Hibernate ke console, memudahkan debug.

#### 7. target/

* Folder hasil build (output) Maven, berisi file .jar yang bisa di jalankan di terminal dengan perintah ‘java -jar api-0.0.1-SNAPSHOT.jar’