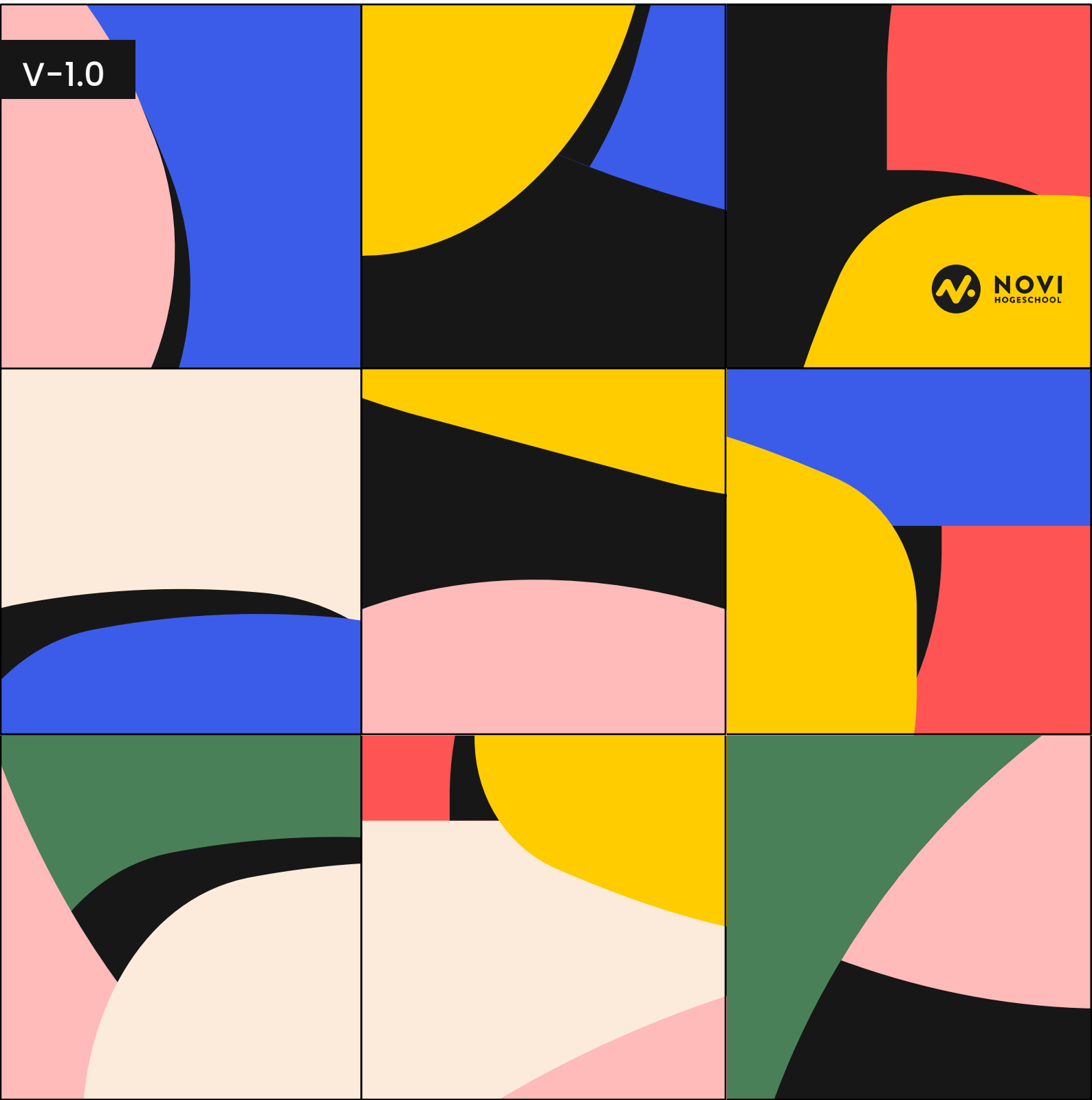
CAR REPOSITORY



Inhoud

Opdracht car repository

Fase 1: Projectopzet met Maven

- **Doel:** Creëer een nieuw Spring Boot-project met Maven.
- **Opdracht:** Gebruik Spring Initializr om een project te genereren met de volgende afhankelijkheden: Spring Web, Spring Data JPA, en PostgreSQL Driver. Je mag zelf de naam van je project kiezen.
- **Richtlijnen:**
  - Selecteer Maven als je build tool en Java als programmeertaal.
  - Voeg de genoemde dependencies toe aan je project.
  - Download en importeer het project in je IDE.

Fase 2: Database Configuratie

- **Doel:** Stel de verbinding met je PostgreSQL-database in.
- **Opdracht:** Voeg de volgende database settings toe aan je `application.properties` bestand:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/jouw_database_naam
spring.datasource.username=postgres
spring.datasource.password=jouw_wachtwoord

# forces to generate the data structures
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create

# spring
spring.sql.init.platform=postgres

## JPA
spring.jpa.database=postgresql
## laat de sql commands zien
spring.jpa.show-sql=true
```

- **Richtlijnen:**
  - Vervang `jouw_database_naam` en `jouw_wachtwoord` met je eigen database naam en wachtwoord.
  - Zorg ervoor dat PostgreSQL draait en toegankelijk is op de aangegeven URL.

Fase 3: Entity Definitie

- **Doel:** Definieer een entiteit voor je applicatie.
- **Opdracht:** Definieer een entiteit met de naam `Car` met velden voor `id`, `brand`, en `model`. Gebruik de geschikte JPA annotaties om de entiteit te mappen naar een database tabel.
- **Richtlijnen:**
  - Kies zelf een package naam voor je entiteiten.
  - Gebruik annotaties zoals `@Entity`, `@Id`, en `@GeneratedValue`.

Fase 4: Repository Laag

- **Doel:** Implementeer de repository laag.
- **Opdracht:** Maak een interface genaamd `CarRepository` die `JpaRepository` uitbreidt en voeg een methode toe om auto's op merk te zoeken.
- **Richtlijnen:**
  - Bepaal zelf de package waar je de repository interface plaatst.

Fase 5: Service Laag

- **Doel:** Bouw de service laag.
- **Opdracht:** Creëer een service klasse met methodes om auto's toe te voegen en op te halen.
- **Richtlijnen:**
  - Gebruik de repository binnen je service voor interactie met de database.
  - Behandel de logica voor het vinden van auto's op basis van id of merk.

Fase 6: Controller Laag

- **Doel:** Implementeer de controller laag.
- **Opdracht:** Maak een controller klasse met API endpoints voor het toevoegen van een nieuwe auto (POST) en het ophalen van auto's (GET, zowel alle auto's als gefilterd op merk).
- **Richtlijnen:**
  - Maak gebruik van `@RestController` en `@RequestMapping` annotaties om je controller te definiëren.
  - De base URL voor je endpoints kan bijvoorbeeld `/cars` zijn, maar je mag zelf een naam kiezen.

Testen met Postman

- **Doel:** Test de functionaliteit van je API.
- **Opdracht:** Gebruik Postman om de POST en GET requests naar je API te sturen en te verifiëren dat de operaties zoals verwacht werken.
- **Richtlijnen:**
  - Test het toevoegen van een nieuwe auto met een POST request.
  - Test het ophalen van auto's met GET requests, zowel zonder filters als gefilterd op merk.

Les 3 Opdracht – hints

Deze uitwerking dient als referentie bij het oplossen van veelvoorkomende problemen.

Stap 1: Projectopzet met Maven

1. **Spring Initializr:** Gebruik Spring Initializr om een nieuw project aan te maken. Selecteer Maven als de build tool, Java als de programmeertaal, en voeg de dependencies toe voor Spring Web, Spring Data JPA, en PostgreSQL Driver.

Stap 2: Database Configuratie

1. **Application.properties Instellingen:** Zorg ervoor dat je `application.properties` bestand correct is geconfigureerd om te verbinden met je PostgreSQL-database. Een veelvoorkomend probleem is een verkeerd geconfigureerde URL, gebruikersnaam of wachtwoord. Controleer ook of de database draait en toegankelijk is.

Stap 3: Entity Definitie

1. **Car Entity:** Definieer de `Car` entiteit binnen een package naar keuze. Zorg ervoor dat je de juiste annotaties gebruikt:

java

```
@Entity
@TableName("cars")
public class Car {
    @Id
    @GeneratedValue
    private Long id;
    //  @Column(name = "car_brand")
    private String brand;
    private String model;
    // Getters en setters
}
```

Let op het gebruik van `@Entity` voor de klasse, `@Id` voor het primaire sleutelveld, en `@GeneratedValue` om de ID-waarde automatisch te laten genereren.

Stap 4: Repository Laag

1. **CarRepository Interface:** Implementeer de `CarRepository` interface die `JpaRepository<Car, Long>` uitbreidt. Als je een methode wilt toevoegen om auto's op merk te zoeken, kun je deze als volgt definiëren:

java

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

Stap 5: Service Laag

1. **CarService Klasse:** Implementeer de service laag door een `CarService` klasse te maken die de `CarRepository` gebruikt. Voorbeeldmethoden om een auto toe te voegen en auto's op te halen kunnen er als volgt uitzien:

java

```
@Service
public class CarService {
    private final CarRepository carRepository;

    public CarService(CarRepository carRepository) {
        this.carRepository = carRepository;
    }

    public Car saveCar(Car car) {
        return carRepository.save(car);
    }

    public List<Car> getCars(String brand) {
        if (brand != null) {
            return carRepository.findByBrand(brand);
        }
        return carRepository.findAll();
    }
}
```

Stap 6: Controller Laag

1. **CarController Klasse:** Voor de controller laag, zorg ervoor dat je `@RestController` en `@RequestMapping` annotaties correct gebruikt. Bijvoorbeeld:

java

```
@RestController
@RequestMapping("/cars")
public class CarController {
    private final CarService carService;

    public CarController(CarService carService) {
        this.carService = carService;
    }

    @PostMapping
    public ResponseEntity<Car> createCar(@RequestBody Car car) {
        Car savedCar = carService.saveCar(car);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedCar);
    }
```

```
    @GetMapping
    public ResponseEntity<List<Car>> getAllCars(@RequestParam(required =
false) String brand) {
        List<Car> cars = carService.getCars(brand);
        return ResponseEntity.ok(cars);
    }
}
```

Let op het gebruik van `@PostMapping` voor het toevoegen van een nieuwe auto en `@GetMapping` voor het ophalen van auto's.

Testen met Postman

- **Postman Testen:** Zorg ervoor dat je endpoints correct functioneren door ze te testen met Postman. Maak een POST request om een nieuwe auto toe te voegen en een GET request om de toegevoegde auto's op te halen.

Volledige uitwerking

Fase 1

zorg dat je project aangemaakt is en de dependency toegevoegd

Fase 2 database settings

ga naar je project - src-main-resources-Application.properties

plaats daarin deze tekst en gebruik je eigen settings. Let op de poort, gebruikers naam en wachtwoord. De database naam moet ook kloppen met een bestaande database in PGadmin. Als deze niet bestaat maak de database eerst aan.

yml

```
spring.datasource.url=jdbc:postgresql://localhost:5432/jouw_database_naam
spring.datasource.username=postgres
spring.datasource.password=jouw_wachtwoord

  # forces to generate the data structures
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create

  # spring
spring.sql.init.platform=postgres

  ## JPA
spring.jpa.database=postgresql
  ## laat de sql commands zien
spring.jpa.show-sql=true
```

Fase 3

De car entiteit:

java

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
@Table(name = "cars")
public class Car {

    @Id
    @GeneratedValue
    private long id;
```

```
    //  @Column(name = "carBrand")
    private String brand;
    private String model;

    public long getId() {
        return id;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }
}
```

Fase 4 Repostiory

Maak een package repositories aan en daarin de class CarRepository

java

```
import java.util.List;

public interface CarRepository extends JpaRepository<Car, Long> {
    List<Car> findByBrand(String brand);
}
```

Fase 5 Servicelaag

Maak een package services aan en daarin de class CarService

java

```
import java.util.List;
import java.util.Optional;

@Service
public class CarService {

    private CarRepository carRepository;

    public CarService(CarRepository carRepository) {
```

```
        this.carRepository = carRepository;
    }

    public List<Car> getAllCars() {
        return carRepository.findAll();
    }

    public Optional<Car> getCarById(Long id) {
        return carRepository.findById(id);
    }

    public List<Car> getCarsByBrand(String brand) {
        return carRepository.findByBrand(brand);
    }

    public Car save(Car car) {
        return carRepository.save(car);
    }

    public void deleteById(Long id) {
        carRepository.deleteById(id);
    }
}
```

Fase 6 Controllerlaag

java

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("cars")
public class CarController {


    private CarService carService;

    public CarController(CarService carService) {
        this.carService = carService;
    }

    // CREATE
    @PostMapping
    public ResponseEntity<Car> createCar(@RequestBody Car car) {
        Car savedCar = carService.save(car);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedCar);
```

```
    }

    // READ: alle auto's of gefilterd op merk
    @GetMapping
    public ResponseEntity<List<Car>> getAllCars(@RequestParam(name = "brand",
required = false) String brand) {
        List<Car> cars = (brand == null) ? carService.getAllCars() :
carService.getCarsByBrand(brand);
        return ResponseEntity.ok(cars);
    }
}
```

Voor een verdere uitbreiding van de controller voor alle crud opdrachten kun je dit toevoegen

Java

```
    //optioneel

// READ: specifieke auto
@GetMapping("/{id}")
public ResponseEntity<Car> getCar(@PathVariable Long id){
        return carService.getCarById(id)
        .map(ResponseEntity::ok)
        .orElseGet(()->ResponseEntity.notFound().build());
        }

// UPDATE
@PutMapping("/{id}")
public ResponseEntity<Car> updateCar(@PathVariable Long id,@RequestBody Car
carDetails){
        return carService.getCarById(id)
        .map(car->{
        car.setBrand(carDetails.getBrand());
        car.setModel(carDetails.getModel());
        Car updatedCar=carService.save(car);
        return ResponseEntity.ok(updatedCar);
        }).orElseGet(()->ResponseEntity.notFound().build());
        }

// DELETE
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteCar(@PathVariable Long id){
        if(carService.getCarById(id).isEmpty()){
        return ResponseEntity.notFound().build();
        }
        carService.deleteById(id);
        return ResponseEntity.noContent().build();
        }
```