

Demo App

Backend mit Spring Boot, JPA und Keycloak

Übersicht

M294 Demo App - Vehicle Usage

- Dashboard
- Departments
- Employees
- Vehicles
- Vehicle Usage


Logout

Language
English

+ Add new vehicle usage data							
Date from	Date to	From	To	KM	Vehicle	Employee	Text
01.08.2022	03.08.2022	Basel	Bellinzona	250.00	BL 11111	123456	Team Event
Items per page: 10 1 - 1 of 1							

M-294

M-295

Swagger

powered by SMARTREAS

/v3/api-docs

Explore

DemoApp API

v1

OAS3

/v3/api-docs

Servers

http://localhost:9090 - Generated server url

Authorize

vehicle-usage-controller

GET

/api/vehicleusage/{id}

PUT

/api/vehicleusage/{id}

DELETE

/api/vehicleusage/{id}

GET

/api/vehicleusage

POST

/api/vehicleusage

Funktionen

- Erfassung (CRUD) von
 - Abteilungen
 - Fahrzeugen
 - Mitarbeitern
 - Fahrzeugnutzung
- Datenbankanbindung mit JPA an PostgreSQL
- Token (OAuth2) basiertes Login mit Keycloak
- API Dokumentation mit Swagger

Spring Framework & Spring Boot

- DI (Dependency Injection) Container
- Framework bietet viele Komponenten für
 - Web Applikationen
 - Datenbankbindung
 - Testing
 - Etc.
- Spring Boot bietet ein einfaches Setup einer Spring Applikation.



Datenbankanbindung

- Datasource, JPA Repository, Entities

```
spring:
  jpa:
    show-sql: true
    generate-ddl: true
    hibernate:
      ddl-auto: update
  datasource:
    url: jdbc:postgresql://localhost:5432/m-295-demo
    username: postgres
    password: postgres
    driverClassName: org.postgresql.Driver
```

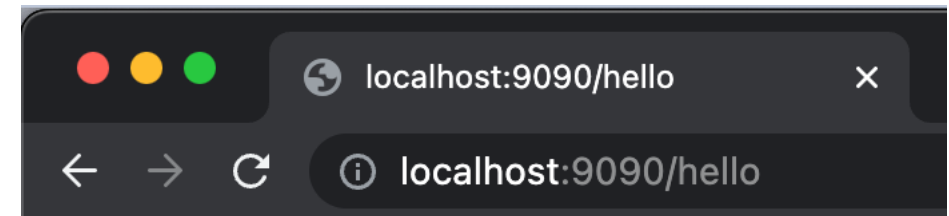
```
@Entity
public class Vehicle {
    @Id
    @GeneratedValue
    private Long id;
    1 usage
    @Column(length = 20, unique = true, nullable = false)
    private String licence;
    1 usage
    @Column(length = 255)
    private String description;
    1 usage
    @Column(nullable = false)
    private VehicleType vehicleType;
```

```
@Repository
public interface VehicleRepository extends JpaRepository<Vehicle, Long> {
    1 usage  👤 Michael Hauck
    List<Vehicle> findByOrderByLicenceAsc();
}
```

Rest(ful) Controller

- Controller bzw. dessen Methoden werden an URL's und HTTP Methoden (GET, POST, etc.) gebunden
- Rückgabewert enthält typischerweise Daten im JSON Format

```
@RestController()  
public class HelloWorldController {  
    // Michael Hauck *  
    @GetMapping("/hello")  
    ResponseEntity<String> hello() {  
        return new ResponseEntity<String>(body: "Hello there!", HttpStatus.OK);  
    }  
}
```



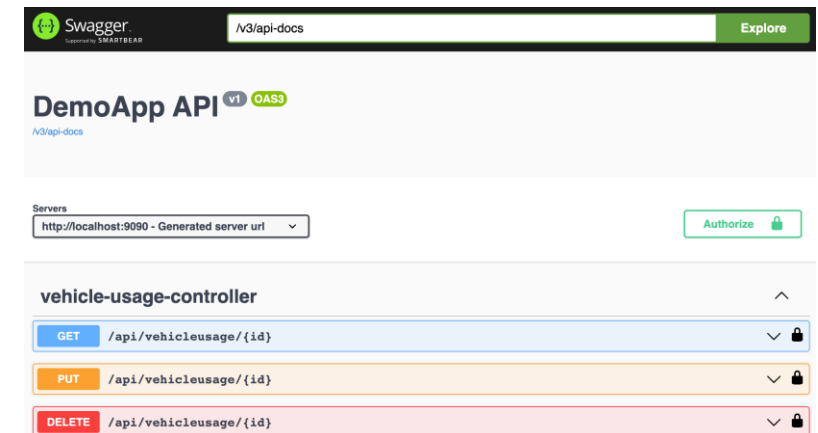
Hello there!

Security

- OAuth 2.0
- Token
- Rollen
- Cross-Origin Resource Sharing (CORS)

Dokumentation (OpenAPI / Swagger)

- OpenAPI 3 Standard (<https://spec.openapis.org/oas/v3.1.0>)
- Erlaubt die Dokumentation von Rest Controller / Endpunkten
- Testing über Web Oberfläche
- Einfache Integration in Spring Web Projekte



Testing

- Manuelle Tests
 - Rest Controller Tests mit Swagger UI
 - Rest Controller Tests mit Postman
- Automatische Unit Tests mit JUnit 5
 - Rest Controller Tests
 - DB Testing
 - Tests beliebiger weiterer Teile der Applikation

Testing Datenbank

- Testing mit In-Memory DB oder Live-DB
- Automatisierte JUnit Tests
- Testing von JPA Repositories

```
@DataJpaTest()
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
@Rollback(false)
class DBTests {

    2 usages
    @Autowired
    private VehicleRepository vehicleRepository;

    no usages  Michael Hauck
    @Test
    void insertVehicle() {
        Vehicle objCar = this.vehicleRepository.save(new Vehicle(licence: "BL 123", description: "", VehicleType.CAR));
        Assertions.assertNotNull(objCar.getId());
        Vehicle objBike = this.vehicleRepository.save(new Vehicle(licence: "Bike", description: "", VehicleType.BICYCLE));
        Assertions.assertNotNull(objBike.getId());
    }
}
```

JUnit 

Testing Controller

- Automatische Tests mit JUnit
- Start Backend Applikation
- Abfrage Security Token
- Aufruf Controller

```
@Test
@Order(1)
void testGetVehicles() throws Exception {

    String accessToken = obtainAccessToken();

    api.perform(get( urlTemplate: "/api/vehicle").header( name: "Authorization", ...values: "Bearer " + accessToken)
                .with(csrf()))
        .andDo(print()).andExpect(status().isOk())
        .andExpect(content().string(containsString( substring: "BL 123"))));
}
```