DEZSYS_GK862_WAREHOUSE_ORM

Autor: Kevin Bauer 4CHIT

Zuerst habe ich den mysql container laufen lassen:

```
docker pull mysql:latest
```

```
docker run --name mysql-spring -e MYSQL_ROOT_PASSWORD=test123 -e
MYSQL_DATABASE=spring_example -e MYSQL_USER=springuser -e
MYSQL_PASSWORD=test12 -p 3306:3306 -d mysql:5.7
```

Folgend kann man sich als root damit verbinden:

```
docker exec -it mysql-spring mysql -u root -p
```

Hierbei wird die Datenbank erstellt und ein User dem Rechte zugewiesen werden:

```
create database db_example;
create user 'springuser'@'%' identified by 'ThePassword';
grant all on db_example.* to 'springuser'@'%';
```

Nun muss man den Container zum laufen bringen:

Anschließend muss eine User erstellt werden(name, password), dem dann Zugriff gewährt wird mittels Grant Befehl:

Dieser User muss in die application.properties eingetragen werden.

Hiermit kann das gradle file ganz einfach erstellt werden:
https://start.spring.io/

Bei folgender Route werden alle warehouses ausgegeben

```
@GetMapping("/all")
public ResponseEntity<?> getAllWarehouses() {
    try {
        List<Warehouse> warehouses = warehouseRepository.findAll();
        if (!warehouses.isEmpty()) {
            return new ResponseEntity<>(warehouses, HttpStatus.OK);
```

```
        } else {
            return new ResponseEntity<>("Kein warehouse gefunden",
HttpStatus.NOT_FOUND);
        }
    } catch (Exception e) {
        return new ResponseEntity<>("keine warehouses bekommen",
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Bei folgender Route werden alle warehouses mit gewisser ID ausgegeben

```
@GetMapping("/retrieve/{id}")
public ResponseEntity<?> retrieveWarehouse(@PathVariable("id") Integer id) {
    try {
        Warehouse warehouse = warehouseRepository.findById(id).orElse(null);
        if (warehouse != null) {
            return new ResponseEntity<>(warehouse, HttpStatus.OK);
        } else {
            return new ResponseEntity<>("Warehouse nicht gefunden",
HttpStatus.NOT_FOUND);
        }
    } catch (Exception e) {
        return new ResponseEntity<>("Kein warehouse bekommen",
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Bei folgender Route wird ein zufälliges Warehouse hinzugefügt:

```
@PostMapping("/insert")
public ResponseEntity<?> insertRandomWarehouse() {
    try {
        Warehouse warehouse = createRandomWarehouse();

        warehouseRepository.save(warehouse);
        return new ResponseEntity<>("Zufälliges Warehouse wurde
hinzugefügt", HttpStatus.CREATED);
    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(e.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Questions

Was ist ORM und wie wird JPA verwendet?

ORM (Object-Relational Mapping) ist eine Programmierungstechnik, die verwendet wird, um Daten zwischen inkompatiblen Typsystemen in objektorientierten Programmiersprachen umzuwandeln. Dies wird in der Regel automatisch von ORM-Software durchgeführt, was es Entwicklern ermöglicht, Daten in einer Datenbank mithilfe der hochrangigen Konstrukte der Programmiersprache anstelle von SQL zu erstellen und zu manipulieren. Dies kann die Produktivität der Entwickler und die Wartbarkeit der Anwendungen erheblich verbessern.

JPA (Java Persistence API) ist eine Java-Spezifikation für den Zugriff, die Persistenz und die Verwaltung von Daten zwischen Java-Objekten/Klassen und einer relationalen Datenbank. JPA ist Teil der Java Enterprise Edition Plattform, kann aber auch in Java Standard Edition Anwendungen verwendet werden. Es bietet ein ORM-Framework, das Java-Klassen auf Datenbanktabellen abbildet und es Entwicklern ermöglicht, über Java mit ihrer Datenbank zu interagieren.

Was ist die application.properties und wo muss sie gespeichert werden?

Die `application.properties` Datei ist eine Konfigurationsdatei in Spring Boot und anderen Java-Anwendungen, die verwendet wird, um verschiedene Aspekte der Anwendungskonfiguration zu definieren, wie Datenbankverbindungen, Serverkonfigurationen und andere anwendungsspezifische Einstellungen. Diese Datei muss im Verzeichnis `src/main/resources` in einem typischen Maven- oder Gradle-Projekt gespeichert werden. Diese Platzierung stellt sicher, dass sie beim Erstellen des Projekts korrekt in das resultierende JAR oder WAR-Archiv aufgenommen wird.

Häufig verwendete Annotationen für Entitätstypen und wichtige zu beachtende Punkte

Bei der Verwendung von JPA gibt es mehrere Annotationen, die häufig verwendet werden, um Entitätstypen zu definieren:

- `@Entity`: Kennzeichnet eine Klasse als eine Entitätstabelle in der Datenbank.
- `@Table`: Wird verwendet, um die spezifische Tabelle in der Datenbank anzugeben, mit der die Entitätsklasse verbunden ist.
- `@Id`: Kennzeichnet ein Feld als Primärschlüssel der Entität.
- `@Column`: Wird verwendet, um das Feld einer Entität auf eine spezifische Spalte in der Datenbanktabelle abzubilden.

- `@ManyToOne`, `@OneToMany`, `@OneToOne`, `@ManyToMany`: Annotationen, um verschiedene Arten von Beziehungen zwischen Entitäten zu definieren.

Wichtige Punkte, die beachtet werden müssen, sind unter anderem die korrekte Definition von Primärschlüsseln, das Verständnis der Lebenszyklusverwaltung von Entitäten, und das effiziente Management von Beziehungen (z.B. Lazy vs. Eager Loading).

Erforderliche Methoden für CRUD-Operationen

Für CRUD-Operationen (Create, Read, Update, Delete) sind in der Regel folgende Methoden erforderlich:

- **Create**: Methoden zum Einfügen neuer Datensätze in die Datenbank.
- **Read**: Methoden zum Abrufen von Daten aus der Datenbank. Dies kann über einfache Abfragen oder komplexere Suchoperationen erfolgen.
- **Update**: Methoden zum Aktualisieren bestehender Datensätze in der Datenbank.
- **Delete**: Methoden zum Löschen von Datensätzen aus der Datenbank.

In Java/JPA werden diese Operationen häufig über ein Repository-Interface gehandhabt, das von `JpaRepository` oder einem ähnlichen Spring Data Interface erbt, welches Methoden für all diese Operationen bereitstellt.