DEZSYS_GK862_Notes

Autor: Manuel Fellner
Version: 09.04.2024

Questions

1. ORM (Object-Relational Mapping) and JPA:

- **ORM** is a technology that bridges the gap between object-oriented programming languages (like Java) and relational databases. It simplifies data access by allowing you to work with objects instead of raw SQL queries.
- **JPA (Java Persistence API)** is a popular ORM framework for Java that provides a standardized way to map Java classes (entities) to database tables. It handles the translation between your object properties and database columns behind the scenes.

2. application.properties:

- This file is used to configure various aspects of your Spring Boot application. It can store settings like database connection details, server port, logging configuration, and more.
- The `application.properties` file should be placed in the root directory of your Spring Boot project, alongside your main application class.

3. Entity Annotations:

These annotations are used to define the mapping between your Java classes and database tables. Here are some frequently used annotations:

- `@Entity`: Marks a class as an entity that can be mapped to a database table.
- `@Id`: Defines the primary key for an entity.
- `@Table (name="tableName")`: Optionally specifies the name of the database table for the entity.
- `@Column (name="columnName")`: Defines the mapping between a class property and a database column.
- `@ManyToOne`, `@OneToMany`, `@OneToOne`: Annotations for defining relationships between entities.

Key Points for Annotations:

- An entity class must be annotated with `@Entity`.

- Each entity must have a single property annotated with `@Id`.
- Property names and column names can be matched using `@Column(name="columnName")`.
- Use relationship annotations (`@ManyToOne`, etc.) to define connections between entities.

4. CRUD operations:

CRUD stands for Create, Read, Update, and Delete. These are the fundamental operations used to manage data in a database. Here are the methods typically used for CRUD with JPA:

- **Create:** Use the `save()` method of the JPA repository to persist a new entity object in the database.
- **Read:** Use the `findById()`, `findAll()`, or other finder methods provided by the JPA repository to retrieve entities from the database.
- **Update:** Use the `save()` method again to update an existing entity object. Any changes to the object will be reflected in the database.
- **Delete:** Use the `deleteById()` method of the JPA repository to remove an entity from the database.

1. Requirements for GKü

For the steps that are required for GKü, I'll just follow the following Tutorial: https://spring.io/guides/gs/accessing-data-mysql

1.1 Run MySQL Container

Before we start, we need to run a MySQL Docker container:

```
$ docker run --name dezsys-gk862-mysql -e MYSQL_ROOT_PASSWORD=password -d
mysql
```

1.2 Setup Database

Now that the MySQL Container is running, we need to set it up:

```
$ docker exec -it dezsys-gk862-mysql bash

bash-4.4# mysql --password
Enter password: password
```

Next, we'll create a new database and a new user, which will both be used in our app.

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

1.3 Create the application.properties File

Create a new file at `src/main/resources/application.properties`:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/db_example
spring.datasource.username=springuser
spring.datasource.password=ThePassword
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.show-sql: true
```

1.4 Create the @Entity Model

Create a new file at `src/main/java/com/example/accessingdatamysql/User.java`

- Putting the `@Entity` Notation in front of a class name tells Hibernate that it should make a db table out of this class
- The `@Id` Notation indicates that this attribute is the primary key of the table
- `@GeneratedValue` automatically generates values for the attribute

```java
package main.java.com.example.accessingdatamysql;

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

@Entity // This tells Hibernate to make a table out of this class
public class User {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Integer id;

  private String name;

  private String email;

  public Integer getId() {
    return id;
  }
  public void setId(Integer id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getEmail() {
    return email;
  }

  public void setEmail(String email) {
    this.email = email;
  }
}
```

- Hibernate automatically translates the entity into a table

1.5 Create the Repository

- We need to create a repository that holds user records, as the following listing (in `src/main/java/com/example/accessingdatamysql/Userrepository.java`) shows:

```
package main.java.com.example.accessingdatamysql;

import org.springframework.data.repository.CrudRepository;

import com.example.accessingdatamysql.User;

// This will be AUTO IMPLEMENTED by Spring into a Bean called
userRepository
// CRUD refers Create, Read, Update, Delete

public interface UserRepository extends CrudRepository<User, Integer> {

}
```

1.6 Create a Controller

- Now we'll need to implement the controller that handles all HTTP requests and processes them

- Create new file in `src/main/java/com/example/accessingdatamysql/MainController.java`:

```
package main.java.com.example.accessingdatamysql;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller // This means that this class is a Controller
@RequestMapping(path="/demo") // This means URL's start with /demo (after
Application path)
public class MainController {
  @Autowired // This means to get the bean called userRepository
         // Which is auto-generated by Spring, we will use it to handle
the data
  private UserRepository userRepository;

  @PostMapping(path="/add") // Map ONLY POST Requests
  public @ResponseBody String addNewUser (@RequestParam String name
      , @RequestParam String email) {
    // @ResponseBody means the returned String is the response, not a view
name
    // @RequestParam means it is a parameter from the GET or POST request

    User n = new User();
    n.setName(name);
    n.setEmail(email);
    userRepository.save(n);
    return "Saved";
  }

  @GetMapping(path="/all")
  public @ResponseBody Iterable<User> getAllUsers() {
    // This returns a JSON or XML with the users
    return userRepository.findAll();
  }
}
```

1.7 Create an Application Class

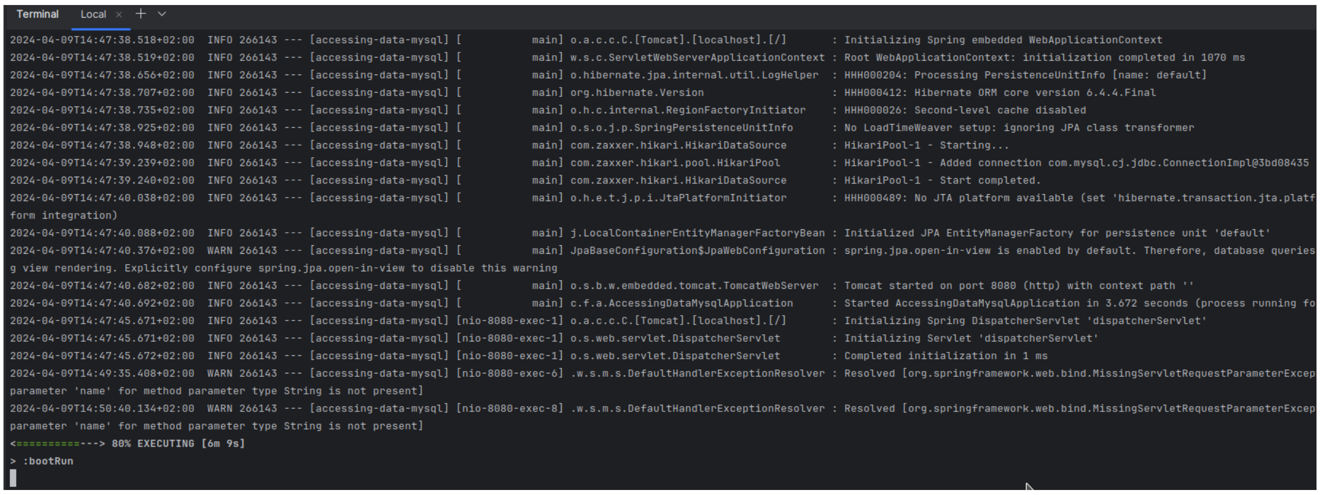
- The Spring Initializr already created a simple class for the Application, as the following `src/main/java/com/example/accessingdatamysql/AccessingDataMysqlApplication.java` shows:

```java
package com.example.accessingdatamysql;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AccessingDataMysqlApplication {

    public static void main(String[] args) {
       SpringApplication.run(AccessingDataMysqlApplication.class, args);
    }

}
```

- For this example, we don't need to modify the `AccessingDataMysqlApplication` class

1.8 Start & test the App

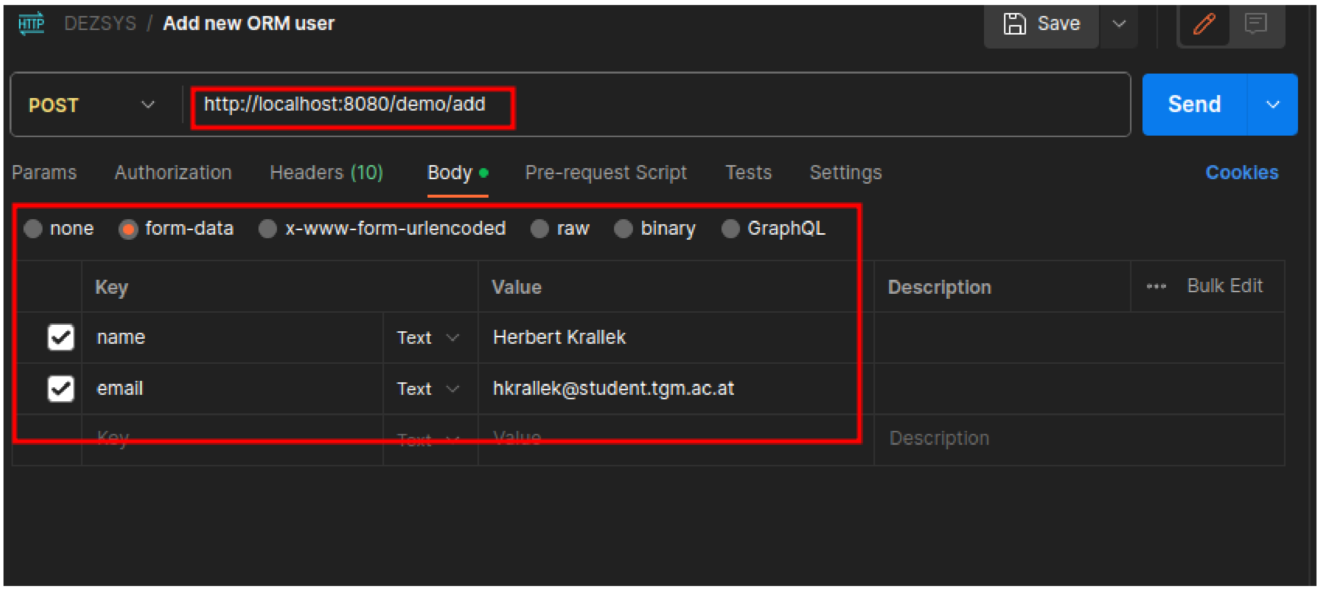
So, now we are just going to start the app with the `./gradlew bootRun` command:



```
2024-04-09T14:47:38.518+02:00  INFO 266143 --- [accessing-data-mysql] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-09T14:47:38.519+02:00  INFO 266143 --- [accessing-data-mysql] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1070 ms
2024-04-09T14:47:38.656+02:00  INFO 266143 --- [accessing-data-mysql] [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-04-09T14:47:38.707+02:00  INFO 266143 --- [accessing-data-mysql] [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.4.4.Final
2024-04-09T14:47:38.735+02:00  INFO 266143 --- [accessing-data-mysql] [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-04-09T14:47:38.925+02:00  INFO 266143 --- [accessing-data-mysql] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-04-09T14:47:38.948+02:00  INFO 266143 --- [accessing-data-mysql] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-04-09T14:47:39.239+02:00  INFO 266143 --- [accessing-data-mysql] [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@3bd08435
2024-04-09T14:47:39.240+02:00  INFO 266143 --- [accessing-data-mysql] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-04-09T14:47:40.038+02:00  INFO 266143 --- [accessing-data-mysql] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platf
form integration)
2024-04-09T14:47:40.088+02:00  INFO 266143 --- [accessing-data-mysql] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-04-09T14:47:40.376+02:00  WARN 266143 --- [accessing-data-mysql] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries
g view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-04-09T14:47:40.682+02:00  INFO 266143 --- [accessing-data-mysql] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-09T14:47:40.692+02:00  INFO 266143 --- [accessing-data-mysql] [           main] c.f.a.AccessingDataMysqlApplication      : Started AccessingDataMysqlApplication in 3.672 seconds (process running fo
2024-04-09T14:47:45.671+02:00  INFO 266143 --- [accessing-data-mysql] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-09T14:47:45.671+02:00  INFO 266143 --- [accessing-data-mysql] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-09T14:47:45.672+02:00  INFO 266143 --- [accessing-data-mysql] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2024-04-09T14:49:35.408+02:00  WARN 266143 --- [accessing-data-mysql] [nio-8080-exec-6] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.bind.MissingServletRequestParameterExcep
parameter 'name' for method parameter type String is not present]
2024-04-09T14:50:40.134+02:00  WARN 266143 --- [accessing-data-mysql] [nio-8080-exec-8] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.bind.MissingServletRequestParameterExcep
parameter 'name' for method parameter type String is not present]
<==========---> 80% EXECUTING [6m 9s]
> :bootRun
```

After that, we are going to add a new user to the database.

For that, we can use `Postman` to send a `POST` Request to the `/demo/add` endpoint:

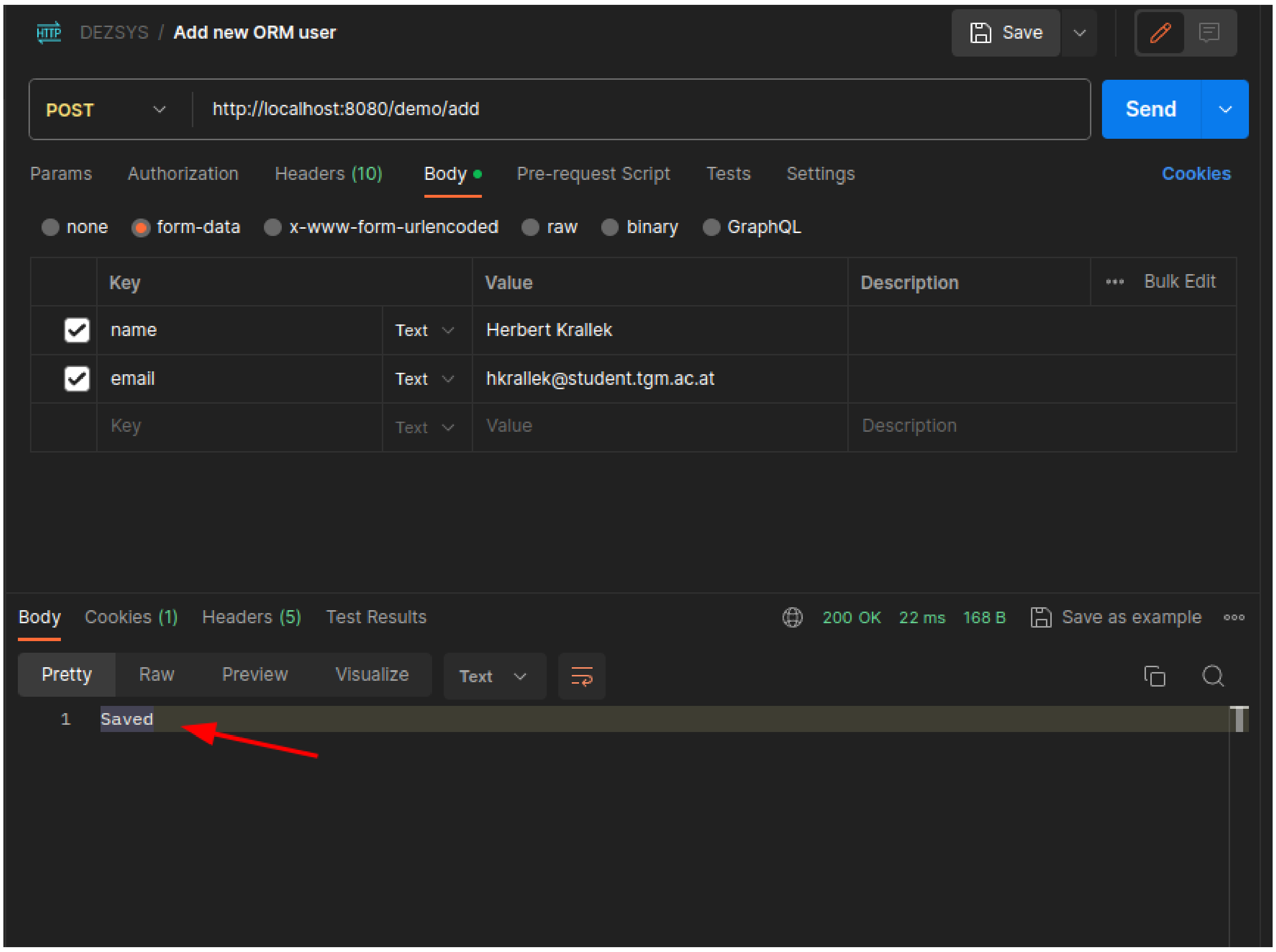


POST http://localhost:8080/demo/add

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	Key		Value	Description
✓	name	Text	Herbert Krallek	
✓	email	Text	hkrallek@student.tgm.ac.at	
	Key	Text	Value	Description

As you can see, the needed parameters for the data is already there. With this request, we'll create a new User called `Herbert Krallek` with the email `hkrallek@student.tgm.ac.at`. This should get saved into the database - Let's execute it!



POST http://localhost:8080/demo/add

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

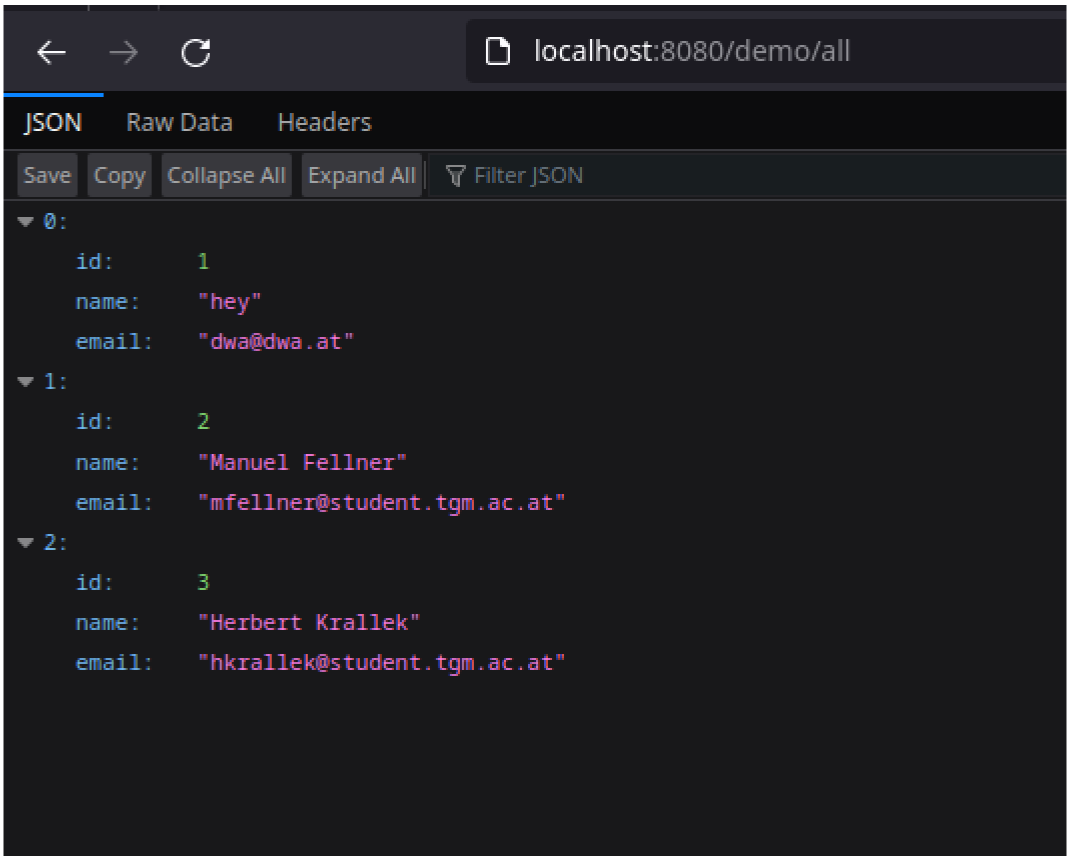
	Key		Value	Description
✓	name	Text	Herbert Krallek	
✓	email	Text	hkrallek@student.tgm.ac.at	
	Key	Text	Value	Description

200 OK 22 ms 168 B

```
Saved
```

We got our expected response!

Then, let's navigate to the `demo/all` endpoint in our browser to verify if it worked:

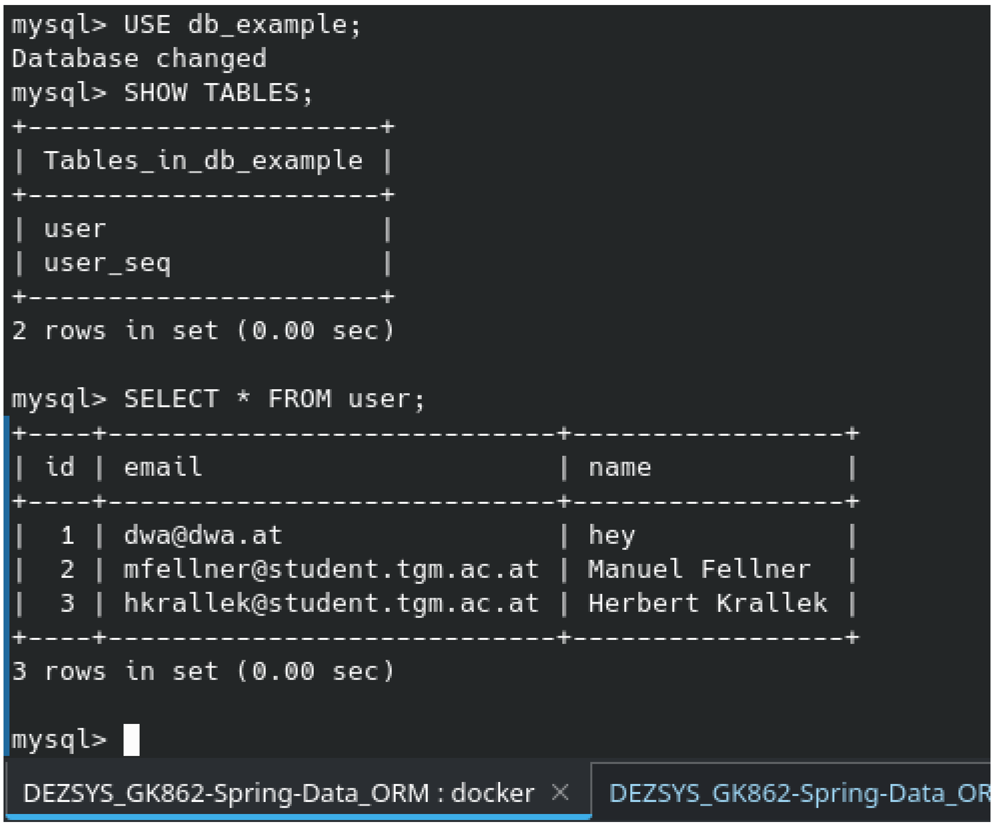


```
0:
    id:       1
    name:     "hey"
    email:    "dwa@dwa.at"
1:
    id:       2
    name:     "Manuel Fellner"
    email:    "mfellner@student.tgm.ac.at"
2:
    id:       3
    name:     "Herbert Krallek"
    email:    "hkrallek@student.tgm.ac.at"
```

Yup, also correct!

And now, to be 100 % sure, we can also check the database itself with the following commands:

```
mysql> USE db_example;
mysql> SHOW TABLES;
+----------------------+
| Tables_in_db_example |
+----------------------+
| user                 |
| user_seq             |
+----------------------+
2 rows in set (0.00 sec)
mysql> SELECT * FROM user;
+----+----------------------------+-----------------+
| id | email                      | name            |
+----+----------------------------+-----------------+
|  1 | dwa@dwa.at                 | hey             |
|  2 | mfellner@student.tgm.ac.at | Manuel Fellner  |
|  3 | hkrallek@student.tgm.ac.at | Herbert Krallek |
+----+----------------------------+-----------------+
3 rows in set (0.00 sec)
```



```
mysql> USE db_example;
Database changed
mysql> SHOW TABLES;
+----------------------+
| Tables_in_db_example |
+----------------------+
| user                 |
| user_seq             |
+----------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM user;
+----+----------------------------+-----------------+
| id | email                      | name            |
+----+----------------------------+-----------------+
|  1 | dwa@dwa.at                 | hey             |
|  2 | mfellner@student.tgm.ac.at | Manuel Fellner  |
|  3 | hkrallek@student.tgm.ac.at | Herbert Krallek |
+----+----------------------------+-----------------+
3 rows in set (0.00 sec)

mysql> 
```

DEZSYS_GK862-Spring-Data_ORM : docker
DEZSYS_GK862-Spring-Data_OR

2. Requirements for GKv

Next, we'll need to customize this ORM experience for our Warehouse application!

2.1 Setup database

For this customization, we'll need a new Database:

```
$ docker exec -it dezsys-gk862-mysql bash

bash-4.4# mysql --password
Enter password: password
```

Then, after logging in, we'll create a new db and grant the springuser access to it:

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

2.2 Create the Entity Models

2.2.1 Product Entity

First, let's create an Entity that is fitting for our Products.

```java
package com.fellner.warehouse.orm.entities;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;

import java.util.UUID;

/**
 * Product Entity * * @author Manuel Fellner
 * @version 2024-04-09
 */
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String name;

    private String productCategory;

    private int productQuantity;

    private String productUnit;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "warehouse_id")
    @JsonIgnore
    private Warehouse warehouse;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

        // ... setter and getter for all attributes
}
```

- It is important that we add the `@JsonIgnore` Annotation here at our warehouse field!
- This ignores the typical JSON serialization, which would destroy our output when we'd try to render this at an API endpoint.

2.2.2 Warehouse Entity

Next, we'll create the Warehouse entity.
It has one "special" Attribute: the `products` attribute. This variable will have to contain one or more `Product` entities.
Therefore, we need to define a relation for that.

```java
package com.fellner.warehouse.orm.entities;
import jakarta.persistence.*;

import java.time.LocalDateTime;
import java.util.List;

/**
 * Warehouse Entity * * @author Manuel Fellner
 * @version 2024-04-09
 */
@Entity
public class Warehouse {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String name;

    private String address;

    private int postalCode;

    private String city;

    private String country;

    private LocalDateTime timestamp;

    @OneToMany(mappedBy = "warehouse", cascade = CascadeType.ALL)
    private List<Product> products;

    public void addProduct (Product product)    {
        if (product != null)    {
            this.products.add(product);
        }
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
        // ... setter and getter for all attributes
}
```

So, we chose the following relation:
`@OneToMany(mappedBy = "warehouse", cascade = CascadeType.ALL)`

This represents a One-To-Many relationship which basically means, that `One` Warehouse can have `Many` Products.

2.3 Create the Repositories

We don't need any fancy repository for the Repositories of our Entities: Just two Files named `ProductRepository.java` and `WarehouseRepository.java`.

2.3.1 Product Repository

~~~
package com.fellner.warehouse.orm.repositories;

import com.fellner.warehouse.orm.entities.Product;
import org.springframework.data.repository.CrudRepository;


// This will be AUTO IMPLEMENTED by Spring into a Bean called
userRepository
// CRUD refers Create, Read, Update, Delete

/**
 * Product repository * * @author Manuel Fellner
 * @version 2024-04-09
 */public interface ProductRepository extends CrudRepository<Product,
Integer> {
}```

#### 2.3.1 Warehouse Repository

```java
package com.fellner.warehouse.orm.repositories;

import com.fellner.warehouse.orm.entities.Warehouse;
import org.springframework.data.repository.CrudRepository;

// This will be AUTO IMPLEMENTED by Spring into a Bean called
userRepository
// CRUD refers Create, Read, Update, Delete

/**
 * Warehouse repository * * @author Manuel Fellner
~~~

```
 * @version 2024-04-09
 */public interface WarehouseRepository extends CrudRepository<Warehouse,
Integer> {
}
```

2.4 Create a Controller

Now, we need a Controller to manage all HTTP requests:

2.4.1 Warehouse Controller

The Controller that manages everything related to the Warehouse entity (adding a new Warehouse, getting all warehouses, getting a specific warehouse with id):

```
package com.fellner.warehouse.orm.Controller;

import com.fellner.warehouse.orm.entities.Product;
import com.fellner.warehouse.orm.entities.Warehouse;
import com.fellner.warehouse.orm.repositories.ProductRepository;
import com.fellner.warehouse.orm.repositories.WarehouseRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.Optional;

/**
 * WarehouseController - Manages all HTTP requests * * @author Manuel
Fellner
 * @version 2024-04-09
 */
@Controller
@RequestMapping(path = "/warehouse")
public class WarehouseController {

    @Autowired
    private WarehouseRepository warehouseRepository;
    @Autowired
    private ProductRepository productRepository;

    @PostMapping(path = "/add")
    public @ResponseBody String addNewWarehouse(@RequestParam String name,
@RequestParam String address, @RequestParam int postalCode, @RequestParam
String city, @RequestParam String country) {
        Warehouse warehouse = new Warehouse();
        warehouse.setName(name);
        warehouse.setAddress(address);
```

~~~~
        warehouse.setPostalCode(postalCode);
        warehouse.setCity(city);
        warehouse.setCountry(country);
        warehouse.setTimestamp(LocalDateTime.now());

        warehouseRepository.save(warehouse);
        return "Warehouse with the name " + name + " saved!";
    }

    @PostMapping (path = "{id}/addProduct")
    public @ResponseBody String addProductToWarehouse (@PathVariable int
id, @RequestParam String name, @RequestParam String productCategory,
@RequestParam int productQuantity, @RequestParam String productUnit) {
        Product product = new Product();
        product.setName(name);
        product.setProductCategory(productCategory);
        product.setProductQuantity(productQuantity);
        product.setProductUnit(productUnit);

        Optional<Warehouse> warehouseOptional =
warehouseRepository.findById(id);

        if (warehouseOptional.isPresent()) {
            Warehouse warehouse = warehouseOptional.get();
            warehouse.addProduct(product);
            product.setWarehouse(warehouse);
            warehouseRepository.save(warehouse);
            return "Product " + name + " added to warehouse " +
warehouse.getName();
        } else {
            return "Warehouse with ID " + id + " not found!";
        }
    }

    @GetMapping(path = "/all")
    public @ResponseBody Iterable<Warehouse> getAllWarehouses() {
        return warehouseRepository.findAll();
    }

    @GetMapping(path = "/{id}")
    public @ResponseBody Optional<Warehouse>
getSpecificWarehouse(@PathVariable int id) {
        return warehouseRepository.findById(id);
    }
}```

#### 2.4.2 Product Controller

The Controller for everything related for the Product entity (adding a new
~~~~

~~~
Product, getting all products or getting a specific product with id):

```java
package com.fellner.warehouse.orm.Controller;
import com.fellner.warehouse.orm.entities.Product;
import com.fellner.warehouse.orm.repositories.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;
import java.util.UUID;

/**
 * Product - Manages all HTTP requests * * @author Manuel Fellner
 * @version 2024-04-09
 */
@Controller
@RequestMapping(path = "/products")
public class ProductController {
    @Autowired
    private ProductRepository productRepository;

    @PostMapping(path = "/add")
    public @ResponseBody String addNewProduct (@RequestParam String name,
@RequestParam String productCategory, @RequestParam int productQuantity,
@RequestParam String productUnit)    {
        Product product = new Product();
        product.setName(name);
        product.setProductCategory(productCategory);
        product.setProductQuantity(productQuantity);
        product.setProductUnit(productUnit);

        productRepository.save(product);
        return "Product with the name " + name + " saved!";
    }

    @GetMapping(path = "/all")
    public @ResponseBody Iterable<Product> getAllProducts ()  {
        return productRepository.findAll();
    }

    @GetMapping(path = "/{id}")
    public @ResponseBody Optional<Product>
getSpecificProduct(@PathVariable int id) {
        return productRepository.findById(id);
    }
~~~

```
}
```

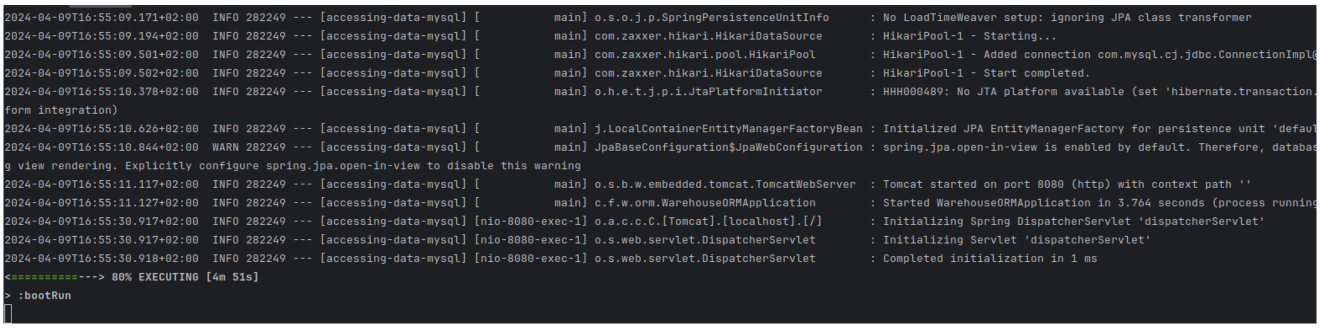
2.5 Create an Application Class

Now, the last thing to do is just creating an Application Class so that our app can run normally.
We can just "paste" the template here into `WarehouseORMApplication.java`:

```java
package com.fellner.warehouse.orm;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Warehouse App class that starts the application * * @author Manuel
Fellner
 * @version 2024-04-09
 */@SpringBootApplication
public class WarehouseORMApplication {
    public static void main(String[] args) {
        SpringApplication.run(WarehouseORMApplication.class, args);
    }
}
```

2.6 Start & Test the app

It is finally time to test our app! With the `./gradlew bootRun` command, the app gets startet:



```
2024-04-09T16:55:09.171+02:00  INFO 282249 --- [accessing-data-mysql] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-04-09T16:55:09.194+02:00  INFO 282249 --- [accessing-data-mysql] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-04-09T16:55:09.501+02:00  INFO 282249 --- [accessing-data-mysql] [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@
2024-04-09T16:55:09.502+02:00  INFO 282249 --- [accessing-data-mysql] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-04-09T16:55:10.378+02:00  INFO 282249 --- [accessing-data-mysql] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.
form integration)
2024-04-09T16:55:10.626+02:00  INFO 282249 --- [accessing-data-mysql] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'defaul
2024-04-09T16:55:10.844+02:00  WARN 282249 --- [accessing-data-mysql] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, databas
g view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-04-09T16:55:11.117+02:00  INFO 282249 --- [accessing-data-mysql] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-09T16:55:11.127+02:00  INFO 282249 --- [accessing-data-mysql] [           main] c.f.w.orm.WarehouseORMApplication        : Started WarehouseORMApplication in 3.764 seconds (process running
2024-04-09T16:55:30.917+02:00  INFO 282249 --- [accessing-data-mysql] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-09T16:55:30.917+02:00  INFO 282249 --- [accessing-data-mysql] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-09T16:55:30.918+02:00  INFO 282249 --- [accessing-data-mysql] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
<==========---> 80% EXECUTING [4m 51s]
> :bootRun
```

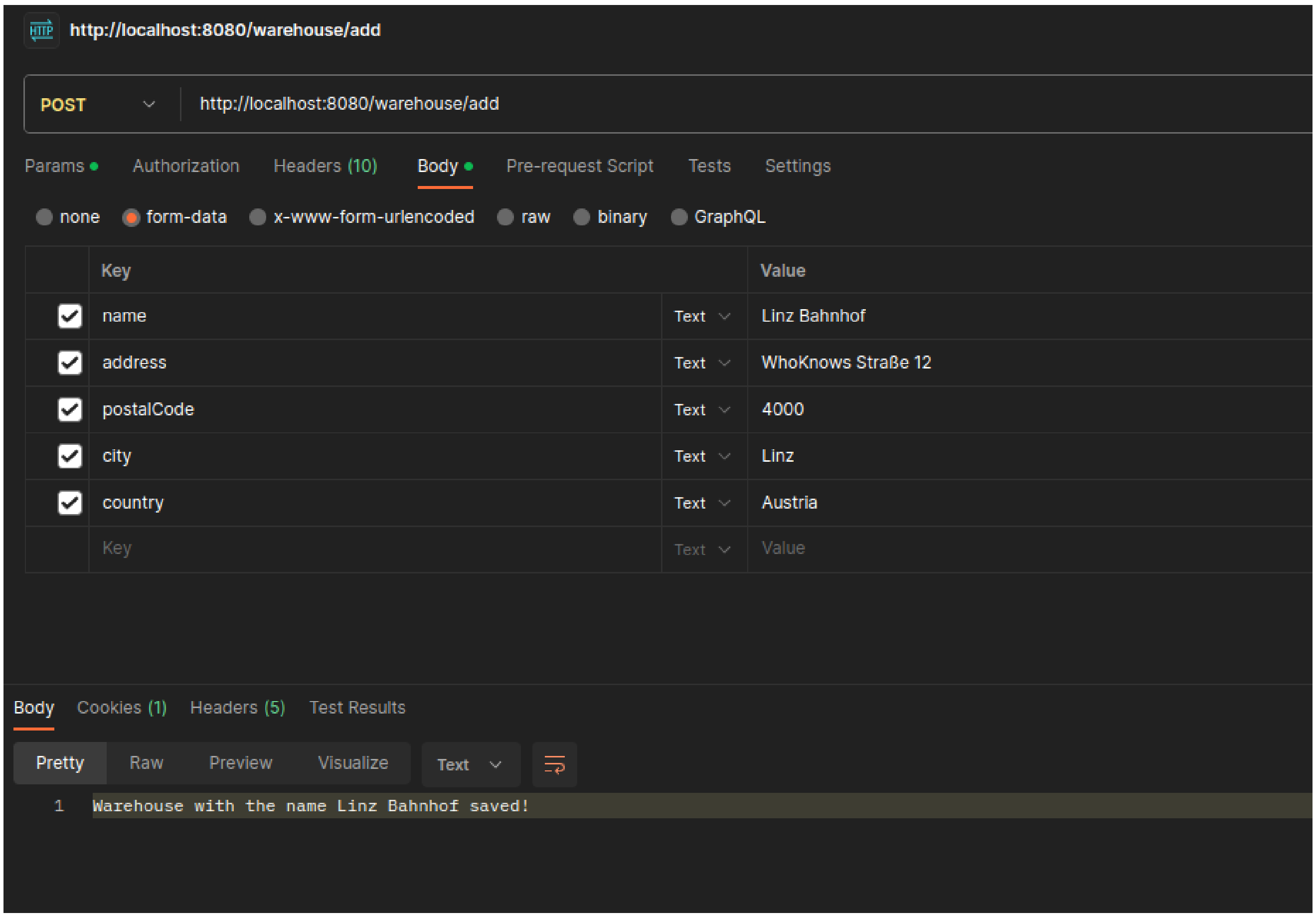
2.6.1 Add Warehouse records

The task description states that we'll need to add two Warehouse records:

- LINZ
- WIEN

The other part of the information are just random values, so e.g. the address, postal code, etc.

We do that by making a `POST` request via `Postman` to the `/warehouse/add` endpoint:



http://localhost:8080/warehouse/add

POST http://localhost:8080/warehouse/add

Params · Authorization · Headers (10) · Body · Pre-request Script · Tests · Settings

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	Key		Value
✓	name	Text	Linz Bahnhof
✓	address	Text	WhoKnows Straße 12
✓	postalCode	Text	4000
✓	city	Text	Linz
✓	country	Text	Austria

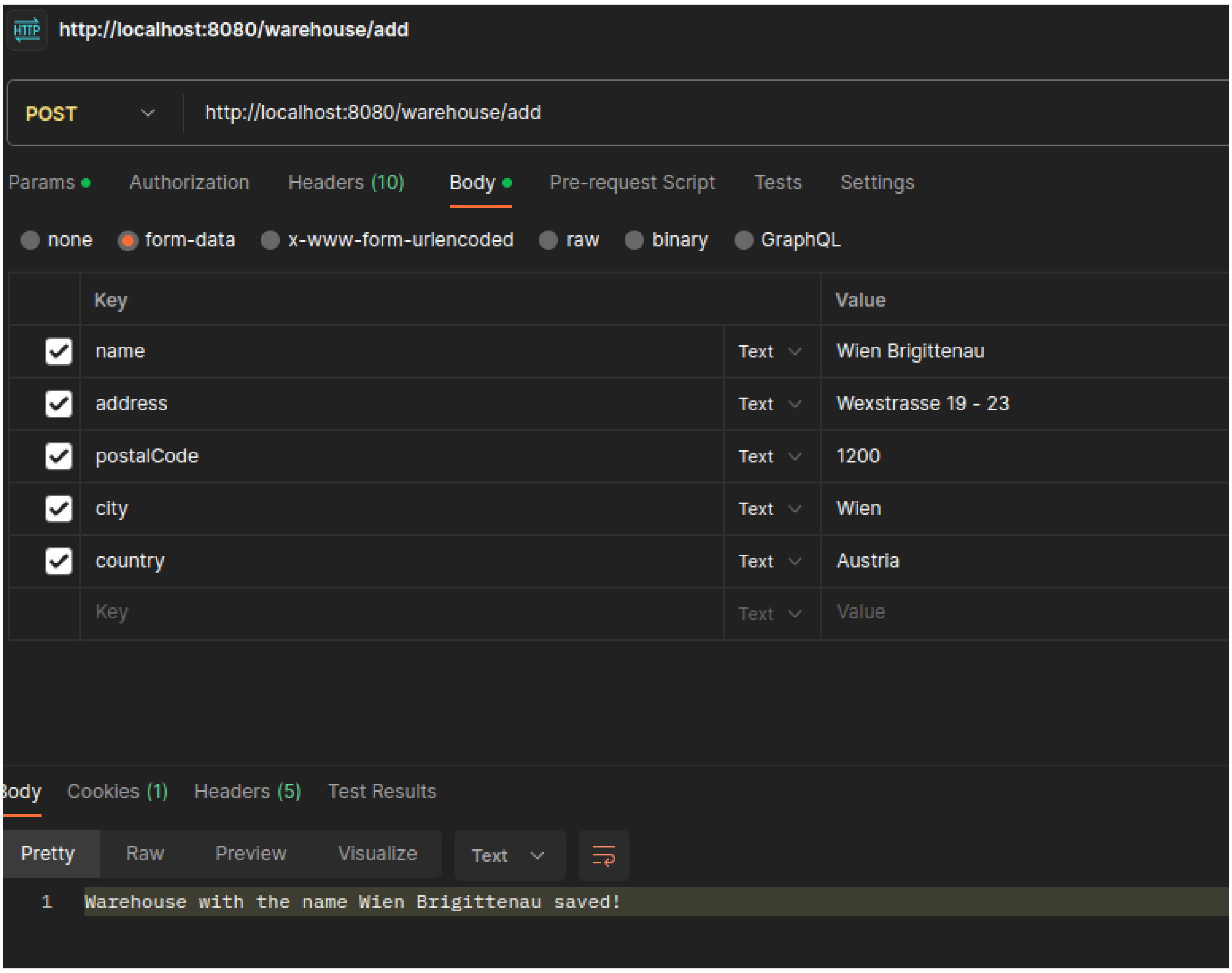
Body · Cookies (1) · Headers (5) · Test Results

Pretty · Raw · Preview · Visualize · Text

```
Warehouse with the name Linz Bahnhof saved!
```

Worked!

Now the WIEN Warehouse:



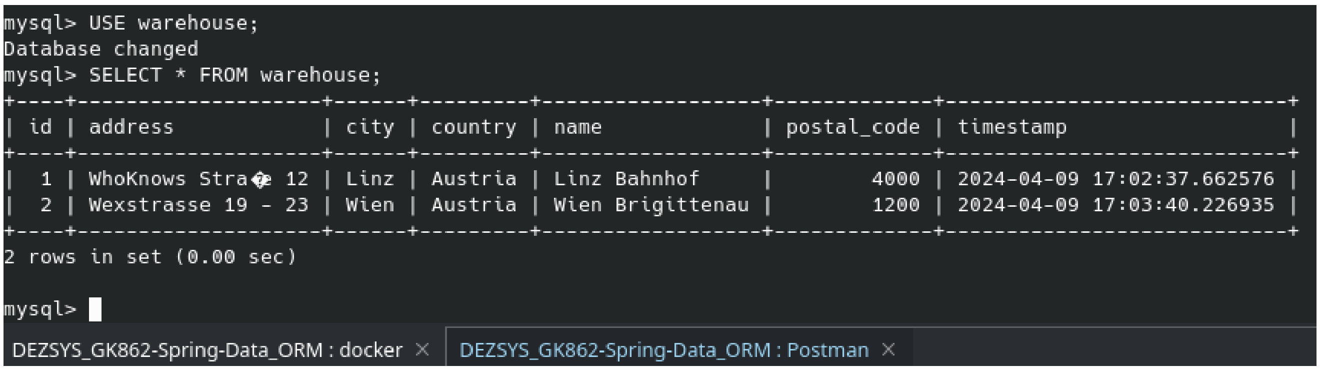
POST http://localhost:8080/warehouse/add

	Key		Value
✓	name	Text	Wien Brigittenau
✓	address	Text	Wexstrasse 19 - 23
✓	postalCode	Text	1200
✓	city	Text	Wien
✓	country	Text	Austria
	Key	Text	Value

```
Warehouse with the name Wien Brigittenau saved!
```

If we go into the `mysql` console and check the entries ourselves, we can see the following:

```
mysql> USE warehouse;
mysql> SELECT * FROM warehouse;
+----+--------------------+------+---------+-----------------+----------
--+----------------------------+
| id | address            | city | country | name             |
postal_code | timestamp                  |
+----+--------------------+------+---------+-----------------+----------
--+----------------------------+
|  1 | WhoKnows Stra�e 12 | Linz | Austria | Linz Bahnhof     |
4000 | 2024-04-09 17:02:37.662576 |
|  2 | Wexstrasse 19 - 23 | Wien | Austria | Wien Brigittenau |
1200 | 2024-04-09 17:03:40.226935 |
+----+--------------------+------+---------+-----------------+----------
--+----------------------------+
2 rows in set (0.00 sec)
```



```
mysql> USE warehouse;
Database changed
mysql> SELECT * FROM warehouse;
+----+--------------------+------+---------+------------------+-------------+----------------------------+
| id | address            | city | country | name             | postal_code | timestamp                  |
+----+--------------------+------+---------+------------------+-------------+----------------------------+
|  1 | WhoKnows Stra� 12  | Linz | Austria | Linz Bahnhof     |        4000 | 2024-04-09 17:02:37.662576 |
|  2 | Wexstrasse 19 - 23 | Wien | Austria | Wien Brigittenau |        1200 | 2024-04-09 17:03:40.226935 |
+----+--------------------+------+---------+------------------+-------------+----------------------------+
2 rows in set (0.00 sec)

mysql> 
```

DEZSYS_GK862-Spring-Data_ORM : docker
DEZSYS_GK862-Spring-Data_ORM : Postman

Now we know that our two Warehouses are existing and saved in the database.

2.6.2 Add Product records to Warehouses

Next it is time to add a few Products to the warehouses.

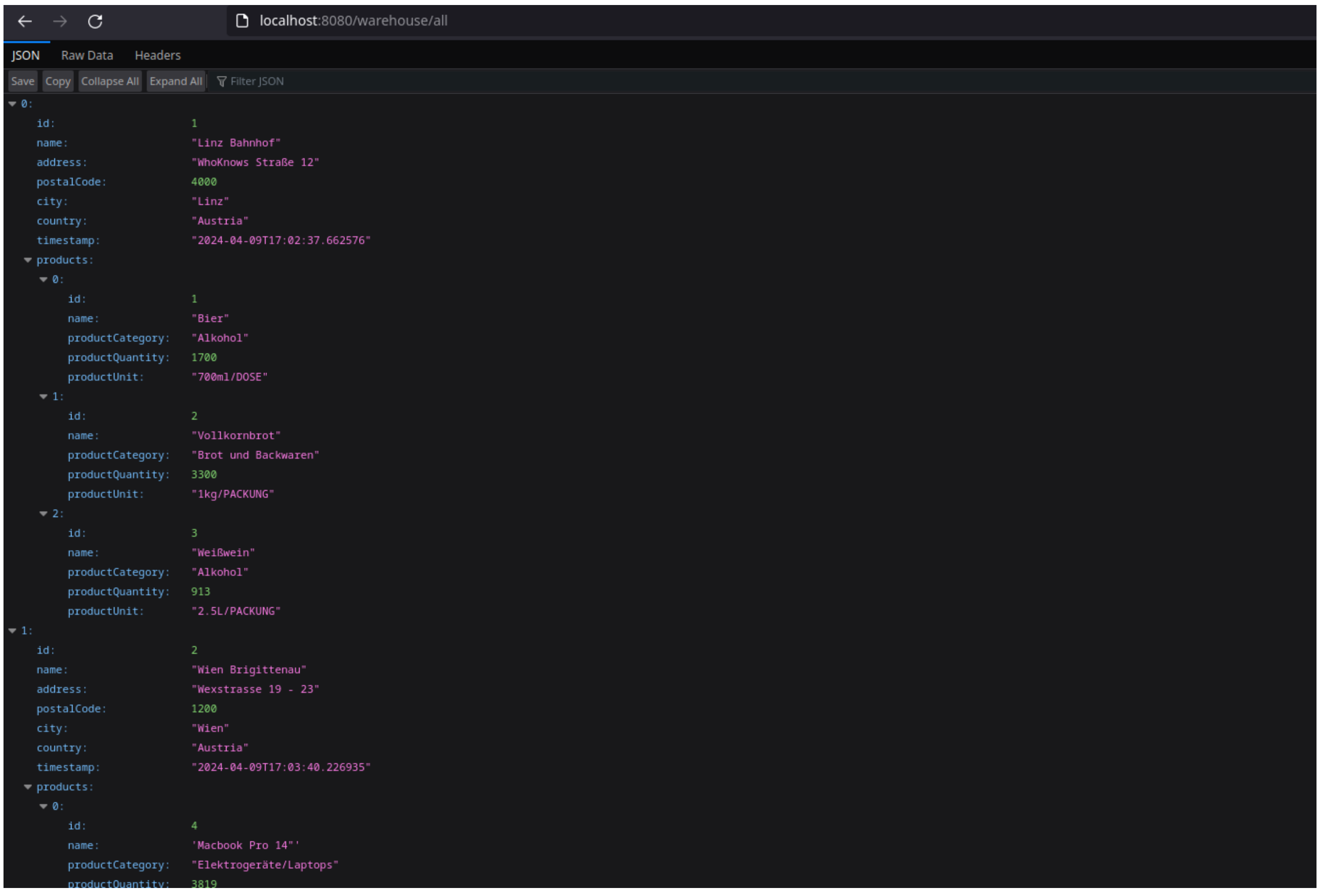
We'll split the Products 3:7;

- Warehouse LINZ has 3 Products:
  - Bier
  - Brot
  - Wein
- Warehouse WIEN has 7 Products:
  - MacBook Pro 14"
  - Gurken
  - Wasserflaschen
  - Mehl
  - Milka Schokolade
  - iPhone 14 Pro
  - Apple Watch ULTRA

For this, we also just need to send `POST` Requests to the `/warehouse/{id}/addProduct` endpoint with the Product data.
I won't show every Postman configuration because they are all really similar.

Now, if we go to `localhost:8080/warehouse/all` we'll see the following:

```json
[
  {
    "id": 1,
    "name": "Linz Bahnhof",
    "address": "WhoKnows Straße 12",
    "postalCode": 4000,
    "city": "Linz",
    "country": "Austria",
    "timestamp": "2024-04-09T17:02:37.662576",
      "products": [
        {
          "id": 1,
          "name": "Bier",
          "productCategory": "Alkohol",
          "productQuantity": 1700,
          "productUnit": "700ml/DOSE"
        },
        {
          "id": 2,
          "name": "Vollkornbrot",
          "productCategory": "Brot und Backwaren",
          "productQuantity": 3300,
          "productUnit": "1kg/PACKUNG"
        },
        {
          "id": 3,
          "name": "Weißwein",
          "productCategory": "Alkohol",
          "productQuantity": 913,
          "productUnit": "2.5L/PACKUNG"
        }
      ]
    },
    {
      "id": 2,
      "name": "Wien Brigittenau",
      "address": "Wexstrasse 19 - 23",
      "postalCode": 1200,
      "city": "Wien",
      "country": "Austria",
      "timestamp": "2024-04-09T17:03:40.226935",
      "products": [
        {
          "id": 4,
          "name": "Macbook Pro 14\"",
          "productCategory": "Elektrogeräte/Laptops",
          "productQuantity": 3819,
          "productUnit": "1/PACKUNG"
        },
        {
          "id": 5,
          "name": "Gurken",
          "productCategory": "Gemüse",
          "productQuantity": 11000,
          "productUnit": "1/PACKUNG"
        },
        {
          "id": 6,
          "name": "Vöslauer Wasserflaschen",
          "productCategory": "Getränke/Grundnahrungsmittel",
          "productQuantity": 13000,
          "productUnit": "500ml/FLASCHE"
        },
        {
          "id": 7,
          "name": "Weizenmehl",
          "productCategory": "Brot und Backwaren",
          "productQuantity": 7319,
          "productUnit": "750g/PACKUNG"
        },
        {
          "id": 8,
          "name": "iPhone 14 Pro",
          "productCategory": "Elektrogeräte/Smartphones",
          "productQuantity": 1293,
          "productUnit": "1/PACKUNG"
        },
        {
          "id": 9,
          "name": "APPLE WATCH ULTRA",
          "productCategory": "Elektrogeräte/Smartwatches",
          "productQuantity": 370,
          "productUnit": "1/PACKUNG"
        },
        {
          "id": 10,
          "name": "Milka Schokolade",
          "productCategory": "Süßigkeiten",
          "productQuantity": 748,
          "productUnit": "100g/PACKUNG"
        }
      ]
    }
  ]
```



```
▼ 0:
    id:                    1
    name:                  "Linz Bahnhof"
    address:               "WhoKnows Straße 12"
    postalCode:            4000
    city:                  "Linz"
    country:               "Austria"
    timestamp:             "2024-04-09T17:02:37.662576"
  ▼ products:
    ▼ 0:
        id:                1
        name:              "Bier"
        productCategory:   "Alkohol"
        productQuantity:   1700
        productUnit:       "700ml/DOSE"
    ▼ 1:
        id:                2
        name:              "Vollkornbrot"
        productCategory:   "Brot und Backwaren"
        productQuantity:   3300
        productUnit:       "1kg/PACKUNG"
    ▼ 2:
        id:                3
        name:              "Weißwein"
        productCategory:   "Alkohol"
        productQuantity:   913
        productUnit:       "2.5L/PACKUNG"
▼ 1:
    id:                    2
    name:                  "Wien Brigittenau"
    address:               "Wexstrasse 19 - 23"
    postalCode:            1200
    city:                  "Wien"
    country:               "Austria"
    timestamp:             "2024-04-09T17:03:40.226935"
  ▼ products:
    ▼ 0:
        id:                4
        name:              'Macbook Pro 14"'
        productCategory:   "Elektrogeräte/Laptops"
        productQuantity:   3819
```

It works!