codecademy

Cheatsheets / **Data with JPA**

Spring Data and JPA

Spring Java Data Model

A *data model* is the set of objects that represents the concepts in your problem domain, whose data you will want to ultimately store in a database.
For example in a music playlist application, the data model would consist of artists, albums, and tracks, and the possible relations between them.

Spring JPA Entity

Applying the `@Entity` annotation to a class with JPA declares that the class definition will correspond to a database table with a similar name.

Spring JPA @Id

The `@Id` annotation can be applied to a member of a class to designate that this member will uniquely identify the entity in the database.

```
@Entity
@Table(name="Library")
public class Book {
  @Id
  private Integer id;

  @Column(name="Title")
  private String Title;
}
```

Spring JPA @GeneratedValue

The `@GeneratedValue` annotation can be used with parameters alongside `@Id` to designate how an entity's unique ID value will be generated. If no parameters are provided, the ID will be generated according to the default algorithm used by the underlying database.

```
@Entity
@Table(name="TELEVISIONS")
public class Television {
  @Id
  
@GeneratedValue(strategy=GenerationType.IDENTITY)
  private Integer id;

  @Column(name="Brand")
  private String brand;
}
```

Spring JPA Repository

With Spring Data JPA, the developer creates a data repository by writing a repository interface and adding custom finder methods. Spring provides the implementation automatically.

```
import
org.springframework.data.repository.CrudRepository;
import
com.codecademy.people.entities.Person;

public interface PersonRepository extends
CrudRepository<Person, Integer> {}
```

Spring JPA Repository Injection

With Spring Data JPA, listing repositories as properties allows them to be made available to other classes. This functionality is facilitated by the repository instance being injected into the class at runtime.

```
import
org.springframework.web.bind.annotation.R
estController;
import
com.codecademy.people.repositories.Person
Repository;

@RestController
public class PersonController {
  private final PersonRepository
personRepository;

  public PersonController(final
PersonRepository personRepository) {
    this.personRepository =
personRepository;
}
```

Spring JPA Save Method

The `save` method of the `CrudRepository` can be used to create or update an entity in the database. It returns the newly-saved / updated entity.

```
@PostMapping("/people")
public Person
createNewPerson(@RequestBody Person
person) {
  Person newPerson =
this.personRepository.save(person);
  return newPerson;
}
```

Spring JPA FindAll Method

The `findAll` method of the `CrudRepository` returns an iterable collection of entities from the database.

```
@GetMapping("/people")
public Iterable<Person> getAllPeople() {
  return this.personRepository.findAll();
}
```

Spring JPA Delete Method

The `delete` method of the `CrudRepository` removes a given entity.

```
@DeleteMapping("/people/{id}")
public Person
deletePerson(@PathVariable("id") Integer
id) {
  Optional<Person> personToDeleteOptional
= this.personRepository.findById(id);
  if
(!personToDeleteOptional.isPresent()) {
    return null;
  }
  Person personToDelete =
personToDeleteOptional.get();

this.personRepository.delete(personToDele
te);
  return personToDelete;
}
```

Spring JPA FindByID Method

The `findById` method of the `CrudRepository` retrieves an entity by its ID.

```
@GetMapping("/people/{id}")
public Optional<Person>
getPersonById(@PathVariable("id") Integer
id) {
  return
this.personRepository.findById(id);
}
```

Spring JPA H2 Console

Spring Boot configures an H2 console that allows developers to inspect the application's database. The console can be accessed via the browser at the URI `/h2-console`.

Print Share

codecademy