irian

Spring Data

- makes it easy to easily implement JPA based repositories.
- enhanced support for JPA based data access layers.
- it makes it easier to build Spring-powered applications that use data access technologies.

irian

Spring Data JPA

Declaring a dependency to a Spring Data module

```
<dependencies>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
  </dependency>
<dependencies>
```

irian

Spring Data Repository

The goal of the Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers

The central interface in the Spring Data repository abstraction is `Repository`. It takes the domain class to manage as well as the ID type of the domain class as type arguments.

The `CrudRepository` provides sophisticated CRUD functionality for the entity class that is being managed.

irian

Spring Data

```
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);      1

  Optional<T> findById(ID primaryKey); 2

  Iterable<T> findAll();               3

  long count();                        4

  void delete(T entity);               5

  boolean existsById(ID primaryKey);   6

  // … more functionality omitted.
}
```

Annotation-driven configuration of base packages

```
@EnableJpaRepositories(basePackages = "com.acme.repositories.jpa")
@EnableMongoRepositories(basePackages = "com.acme.repositories.mongo")
interface Configuration { }
```

```
13  @Configuration
14  @EnableJpaRepositories("ro.irian.fullstack.pizza.service")
15  @ComponentScan(basePackages = "ro.irian.fullstack.pizza.service")
16  @ImportResource("classpath:/META-INF/pizza-persistence.spring.xml")
17  public class PizzaServiceConfig {
18
19  }
20
```

irian

Spring Data

Exercise - TODO 1

Enable Spring Data Repositories in PizzaServiceConfig

Create a Spring Data Repository for Pizza named PizzaCrudRepository (an interface that extends the CrudRepository)

Change the getAllPizzas method from Service to use the new Repository

irian

Query Creation

```
interface PersonRepository extends Repository<Person, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

irian

Spring Data

Exercise - TODO 2

Create Spring Data Repository for Reviews

Create a method to findAllReviewsByAuthor in the repository

Create methods in the Controller and Service classes in order to send the list of Reviews to the client

On top of the `CrudRepository`, there is a `PagingAndSortingRepository` abstraction that adds additional methods to ease paginated access to entities:

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
  extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

irian

Spring Data

org.springframework.data.domain

Interface Pageable

Modifier and Type	Method and Description
`Pageable`	`first()` Returns the `Pageable` requesting the first page.
`long`	`getOffset()` Returns the offset to be taken according to the underlying page and page size.
`int`	`getPageNumber()` Returns the page to be returned.
`int`	`getPageSize()` Returns the number of items to be returned.
`Sort`	`getSort()` Returns the sorting parameters.

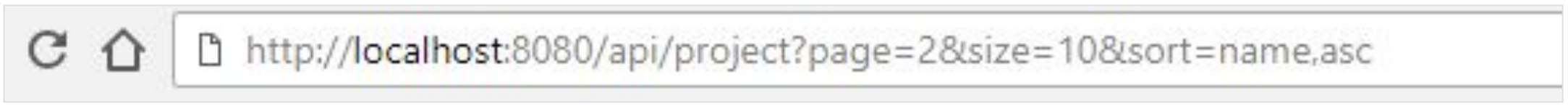
irian

Spring Data

org.springframework.data.domain

Interface Pageable

```
@RequestMapping(method = {RequestMethod.GET}, value = "/page")
public Page<Pizza> getPagedPizzas(Pageable pageable) {
```



with Pageable


```
{
  "content": [
    {
      "_id": "pizza1",
      "version": 0,
      "createdAt": "2018-05-08T12:44:21.671+0000",
      "name": "4 STAGIONI",
      "price": 27.5,
      "weight": 550,
      "image": "images/quattro.png",
      "ingredients": "sos rosii, mozzarella, ciuperci, salam, sunca presata, oregano, anghinare",
      "reviews": [
        {
          "_id": "d81224d346e649c6b4d8ac00c43d334e",
          "version": 0,
          "createdAt": "2018-05-08T12:44:21.671+0000",
          "stars": 5,
          "body": "I love this pizza!",
          "author": "joe@example.org",
          "createdOn": 100000000,
          "transient": false
        },
        {
          "_id": "74c935399c754b18b7d96a9b580c4e13",
          "version": 0,
          "createdAt": "2018-05-08T12:44:21.671+0000",
          "stars": 4,
          "body": "It's great!",
          "author": "miha@example.org",
          "createdOn": 100000000,
          "transient": false
        }
      ],
      "canPurchase": true,
      "soldOut": false,
      "transient": false
    },
    ...
  ],
  "pageable": {
    "sort": {
      "sorted": false,
      "unsorted": true
    },
    "offset": 0,
    "pageSize": 10,
    "pageNumber": 0,
    "paged": true,
    "unpaged": false
  },
  "last": true,
  "totalElements": 3,
  "totalPages": 1,
  "size": 10,
  "number": 0,
  "sort": {
    "sorted": false,
    "unsorted": true
  },
  "numberOfElements": 3,
  "first": true
}
```


without Pageable


```
[
  {
    "_id": "pizza1",
    "version": 0,
    "createdAt": "2018-05-08T12:44:21.671+0000",
    "name": "4 STAGIONI",
    "price": 27.5,
    "weight": 550,
    "image": "images/quattro.png",
    "ingredients": "sos rosii, mozzarella, ciuperci, salam, sunca presata, oregano, anghinare",
    "reviews": [
      {
        "_id": "d81224d346e649c6b4d8ac00c43d334e",
        "version": 0,
        "createdAt": "2018-05-08T12:44:21.671+0000",
        "stars": 5,
        "body": "I love this pizza!",
        "author": "joe@example.org",
        "createdOn": 100000000,
        "transient": false
      },
      {
        "_id": "74c935399c754b18b7d96a9b580c4e13",
        "version": 0,
        "createdAt": "2018-05-08T12:44:21.671+0000",
        "stars": 4,
        "body": "It's great!",
        "author": "miha@example.org",
        "createdOn": 100000000,
        "transient": false
      }
    ],
    "canPurchase": true,
    "soldOut": false,
    "transient": false
  },
  ...
    "canPurchase": true,
    "soldOut": false,
    "transient": false
  }
]
```

with Pageable

irian


```
"pageable": {
    "sort": {
        "sorted": false,
        "unsorted": true
    },
    "offset": 0,
    "pageSize": 10,
    "pageNumber": 0,
    "paged": true,
    "unpaged": false
},
"last": true,
"totalElements": 3,
"totalPages": 1,
"size": 10,
"number": 0,
"sort": {
    "sorted": false,
    "unsorted": true
},
"numberOfElements": 3,
"first": true
}
```

irian

Limiting Query Results

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

irian

Spring Data

Exercise - TODO 3

Add paging functionality to the new Pizza repository

Add a getPagedPizzas method to the Pizza service & controller

Url to test:

Persist your data

SAVING & LOADING DATA (JPA)

Loading data eficient

- Loading collections of object-trees for read-only purpose is not efficient
- Projection -> constructor expressions
- Load only the fields you need, from all the joined entities into a ValueObject
- VO – Pojo with no setters, having constructor with all properties as params

… Projections - > VOs

```
em.createQuery(
        qlString: "select new ro.irian.fullstack.pizza.domain.ReviewVO("
                + "                  p._id, p.name, "
                + "                  r._id, r.stars, r.body, r.author, r.createdOn) "
                + " from Pizza p "
                + "    join p.reviews r   "
                + " where r.createdAt > :dateFrom")
.setParameter( name: "dateFrom", from)
.getResultList();
```

TODO VOs

- Create a method in the JpaPizzaRepository to load the ReviewVOs of *a certain author*.

- The ReviewVOs has pizzaName, reviewStars, reviewBody and reviewAuthor.

- Change the Controller and Service methods that get all the Reviews for an author to return ReviewVOs, using the new repository method

Sending data to server and save it

- RequestMethod.PUT,
  RequestMethod.POST,
  RequestMethod.PATCH
- @RequestBody on method param
- mixed with @PathVariable
- ResponseEntity<?> response wrapper with builder methods

ResponseEntity

```
ResponseEntity.ok().build();
```

```
ResponseEntity.created(
 new URI( str: "/rest/pizzas/" + pizza.get_id()))
 .build();
```

```
ResponseEntity.badRequest().body(
 new ValidationError(
          fieldName: "name",
          errorMessage: "Pizza name must be unique"));
```

… Controller PUT / POST ?

```
@RequestMapping(method = {RequestMethod.POST})
public ResponseEntity<?> savePizza(@RequestBody Pizza pizza) throws URISyntaxException {
    if (pizzaService.findPizzaByName(pizza.getName()) == null) {
        pizzaService.save(pizza);
        return ResponseEntity.created(
                new URI( str: "/rest/pizzas/" + pizza.get_id()))
                .build();
    }
    else {
        return ResponseEntity.badRequest().body(
                new ValidationError(
                        fieldName: "name",
                        errorMessage: "Pizza name must be unique"));
    }
}
```

Use POSTMAN for creation

POST http://localhost:8080//rest/pizzas

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "name": "FRESCA",
    "price": 27.5,
    "weight": 550,
    "image": "images/quattro.png",
    "ingredients": "sos rosii, mozzarella, ciuperci, salam, sunca presata, oregano, anghinare",
    "canPurchase": true,
    "soldOut": false
}
```

Body Cookies Headers (3) Tests

Content-Length → 0

Date → Wed, 09 May 2018 12:37:53 GMT

Location → /rest/pizzas/88b29f518ffd4209a23f051eb93111d5

TODO save

- Implement savePizza method in controller (POST)
- Update the Service method to savePizza using the Spring Data Pizza Repository
- Pizza name must be unique (check if Pizza with same name exists)
- Return ResponseEntity.created(...) if the pizza is new
- Else return ResponseEntity.badRequest(...)

Spring's `Validator` interface

Spring features a `Validator` interface that you can use to validate objects.

The `Validator` interface works using an `Errors` object so that while validating, validators can report validation failures to the `Errors` object.

```java
public class PersonValidator implements Validator {

    /**
    * This Validator validates *just* Person instances
    */
    public boolean supports(Class clazz) {
        return Person.class.equals(clazz);
    }

    public void validate(Object obj, Errors e) {
        ValidationUtils.rejectIfEmpty(e, "name", "name.empty");
        Person p = (Person) obj;
        if (p.getAge() < 0) {
            e.rejectValue("age", "negativevalue");
        } else if (p.getAge() > 110) {
            e.rejectValue("age", "too.darn.old");
        }
    }
}
```

```java
@RequestMapping(method = {RequestMethod.POST})
public ResponseEntity<?> savePizza(@RequestBody Pizza pizza,
                                   BindingResult result)
                                throws URISyntaxException {
    validator.validate(pizza, result);
    if (result.hasErrors()) {
        return ResponseEntity.badRequest().body(result.getFieldError());
    }
```

```
public interface BindingResult
extends Errors
```

General interface that represents binding results. Extends the `interface` for error registration capabilities, allowing for a `Validator` to be applied, and adds binding-specific analysis and model building.

- Exercise

- Move the validation logic from the savePizza method to a custom Validator

- Change the savePizza method to use the Validator

Entity state