Pivotal

Cloud Native Applications

Polyglot Persistence with Spring Data REST

Spring Data

Provides a *familiar* and *consistent*, Spring-based programming model for data access while still retaining the special traits of the underlying data store.

It makes it *easy to use* data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services.

Pivotal

Spring Data JPA

Add the Spring Data JPA starter to our pom.xml file

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

Sprinkle a database connector into our pom.xml

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Sprinkle a little @EnableJpaRepositories annotation into our Spring Boot application

```
@SpringBootApplication
@EnableJpaRepositories
public class CloudNativeSpringApplication {
```

Pivotal

Spring Data JPA, @Entity & Repository

Let's create an @Entity to manage

```
@Entity
@Table(name="city")
public class City implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    private long id;

    @Column(nullable = false)
    private String name;
```

Let's create a JPA Repository to manage our @Entity

```
@RepositoryRestResource(collectionResourceRel = "cities", path = "cities")
public interface CityRepository extends PagingAndSortingRepository<City, Long> {
}
```

Pivotal

Spring Data REST

Goal is to provide a solid foundation on which to expose ***CRUD*** repositories to our *repository managing entities* using plain ***HTTP REST*** semantics

Add a dash of Spring Data REST starter into our pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

Pivotal

Spring Data REST, what happens

- For this repository, Spring Data REST exposes a resource collection at “/cities”
- Context path is derived from the *un-capitalized*, *pluralized*, *simple class name* of the domain class being managed
- Exposes an item resource for each of these items managed by the repository under the URI template /cities/{id}

localhost:8080/cities

Paul

localhost:8080/cities


```
{
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/cities{?page,size,sort}",
      "templated" : true
    },
    "next" : {
      "href" : "http://localhost:8080/cities?page=1&size=20{&sort}",
      "templated" : true
    },
    "search" : {
      "href" : "http://localhost:8080/cities/search"
    }
  },
  "_embedded" : {
    "cities" : [ {
      "name" : "HOLTSVILLE",
      "county" : "SUFFOLK",
      "stateCode" : "NY",
      "postalCode" : "00501",
      "latitude" : "+40.922326",
      "longitude" : "-072.637078",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/cities/1"
        }
      }
    }, {
      "name" : "HOLTSVILLE",
      "county" : "SUFFOLK",
```


Pivotal

Support Search, or findBy*

Add some search methods using @RestResource to our CityRepository class

```
@RepositoryRestResource(collectionResourceRel = "cities", path = "cities")
public interface CityRepository extends PagingAndSortingRepository<City, Long> {
    @RestResource(path = "name", rel = "name")
    Page<City> findByNameIgnoreCase(@Param("q") String name, Pageable pageable);

    @RestResource(path = "nameContains", rel = "nameContains")
    Page<City> findByNameContainsIgnoreCase(@Param("q") String name, Pageable pageable);

    @RestResource(path = "state", rel = "state")
    Page<City> findByStateCodeIgnoreCase(@Param("q") String stateCode, Pageable pageable);

    @RestResource(path = "postalCode", rel = "postalCode")
    Page<City> findByPostalCode(@Param("q") String postalCode, Pageable pageable);
}
```

Pivotal

Spring Data REST, what happens?

For this repository, we now

see search methods when

we hit the /{repository}/

search endpoint

localhost:8080/cities/searc

Paul

localhost:8080/cities/search


```
{
  "_links" : {
    "name" : {
      "href" : "http://localhost:8080/cities/search/name{?q,page,size,sort}",
      "templated" : true
    },
    "postalCode" : {
      "href" : "http://localhost:8080/cities/search/postalCode{?q,page,size,sort}",
      "templated" : true
    },
    "state" : {
      "href" : "http://localhost:8080/cities/search/state{?q,page,size,sort}",
      "templated" : true
    },
    "nameContains" : {
      "href" : "http://localhost:8080/cities/search/nameContains{?q,page,size,sort}",
      "templated" : true
    }
  }
}
```


Pivotal

Spring Data REST, Custom Queries

Add a method “findByStateCode” to our CityRepository that defines an custom query using @Query notation and takes an @Param argument for the stateCode

```
@Query(value = "select c from City c where c.stateCode = :stateCode")
Collection<City> findByStateCode(@Param("stateCode") String stateCode);
```

Pivotal

Spring Data REST, what happens?

localhost:8080/cities/searc

localhost:8080/cities/search/findByStateCode?stateCode=VA


```
{
  "_embedded" : {
    "cities" : [ {
      "name" : "DULLES",
      "county" : "LOUDOUN",
      "stateCode" : "VA",
      "postalCode" : "20101",
      "latitude" : "+39.002125",
      "longitude" : "-077.442066",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/cities/8214"
        }
      }
    }, {
      "name" : "DULLES",
      "county" : "LOUDOUN",
      "stateCode" : "VA",
      "postalCode" : "20102",
      "latitude" : "+39.085309",
      "longitude" : "-077.645224",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/cities/8215"
        }
      }
    }, {
      "name" : "DULLES",
      "county" : "LOUDOUN",
      "stateCode" : "VA",
      "postalCode" : "20103",
      "latitude" : "+38.996242",
```


Pivotal

MongoRepository

```
import org.cloudfoundry.samples.music.domain.Album;
import org.cloudfoundry.samples.music.repositories.AlbumRepository;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface MongoAlbumRepository extends MongoRepository<Album, String>, AlbumRepository {
}
```

Pivotal

RedisRepository

```
import org.cloudfoundry.samples.music.domain.Album;
import org.springframework.data.repository.CrudRepository;

public interface AlbumRepository extends CrudRepository<Album, String> {
}
```

Pivotal

Supported Repositories

Spring

- JPA
- MongoDB
- Redis
- Solr
- GemFire
- KeyValue

Community

- Aerospike
- Cassandra
- Couchbase
- DynamoDB
- ElasticSearch
- Neo4J

Pivotal™

LAB