Lab 9 - Build a Hypermedia-Driven RESTful Web Service with Spring Data REST

In this lab we'll utilize Spring Boot, Spring Data, and Spring Data REST to create a fully-functional hypermedia-driven RESTful web service. We'll then deploy it to Cloud Foundry (using Pivotal Web Services).

NOTE The completed code for this lab can be found at `$COURSE_HOME/day_01/session_03/lab_09/complete/cities`.

Initializing the Application

1. Change to the lab directory:

```
$ cd $COURSE_HOME/day_01/session_03/lab_09/initial/cities
```

2. Optionally generate project files for your favorite IDE by running `./gradlew idea` or `./gradlew eclipse`. Then open the project in your editor or IDE of choice.

3. Add a runtime dependency on the HyperSQL in-memory database (http://hsqldb.org/) to `build.gradle`:

```groovy
dependencies {
    // ...
    runtime("org.hsqldb:hsqldb")
}
```

4. Create the package `org.example.cities.domain` and in that package create the class `City`. Into that file you can paste the following source code, which represents cities based on postal codes, global coordinates, etc:

```java
@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;

    @Column(nullable = false)
    private String county;

    @Column(nullable = false)
    private String stateCode;

    @Column(nullable = false)
    private String postalCode;

    @Column
    private String latitude;

    @Column
    private String longitude;

    public String getName() { return name; }

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

    public String getPostalCode() { return postalCode; }

    public void setPostalCode(String postalCode) { this.postalCode = postalCode; }

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

    public String getStateCode() { return stateCode; }

    public void setStateCode(String stateCode) { this.stateCode = stateCode; }

    public String getCounty() { return county; }

    public void setCounty(String county) { this.county = county; }

    public String getLatitude() { return latitude; }

    public void setLatitude(String latitude) { this.latitude = latitude; }

    public String getLongitude() { return longitude; }

    public void setLongitude(String longitude) { this.longitude = longitude; }
}
```

Notice that we're using JPA annotations on the class and its fields. You'll need to use your IDE's features to add the appropriate import statements.

5. Create the package `org.example.cities.repositories` and in that package create the interface `CityRepository`. Paste the following code and add appropriate imports:

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

6. Add JPA and REST Repository support to the `org.example.cities.Application` class that was generated by Spring Initializr.

```java
@SpringBootApplication
@EnableJpaRepositories // <---- Add this
@Import(RepositoryRestMvcConfiguration.class) // <---- And this
public class Application {

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

7. Build the application:

```bash
$ ./gradlew assemble
```

8. Run the application:

```bash
$ java -jar build/libs/cities-0.0.1-SNAPSHOT.jar
```

9. Access the application using `curl`. You'll see that the primary endpoint automatically exposes the ability to page, size, and sort the response JSON.

   So what have you done? Created four small classes and one build file, resulting in a fully-functional REST microservice. The application's `DataSource` is created automatically by Spring Boot using the in-memory database because no other `DataSource` was detected in the project.

```bash
$ curl -i localhost:8080/cities
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Application-Context: application
Content-Type: application/hal+json
Transfer-Encoding: chunked
Date: Tue, 27 May 2014 19:34:45 GMT

{
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/cities{?page,size,sort}",
      "templated" : true
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 0,
    "totalPages" : 0,
    "number" : 0
  }
}
```

Next we'll import some data.

Importing Data

1. Add this import.sql file (https://github.com/cf-platform-eng/spring-boot-cities/blob/master/cities-service/src/main/resources/import.sql) to `src/main/resources`. This is a rather large dataset containing all of the postal codes in the United States and its territories. This file will automatically be picked up by Hibernate and imported into the in-memory database.
2. Build the application:

```bash
$ ./gradlew assemble
```

3. Run the application:

```bash
$ java -jar build/libs/cities-0.0.1-SNAPSHOT.jar
```

4. Access the application again using `curl`. Notice the appropriate hypermedia is included for `next`, `previous`, and `self`. You can also select pages and page size by utilizing `?size=n&page=n` on the URL string. Finally, you can sort the data utilizing `?sort=fieldName`.

```bash
$ curl -i localhost:8080/cities
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Application-Context: application
Content-Type: application/hal+json
Transfer-Encoding: chunked
Date: Tue, 27 May 2014 19:59:58 GMT

{
  "_links" : {
    "next" : {
      "href" : "http://localhost:8080/cities?page=1&size=20"
    },
    "self" : {
      "href" : "http://localhost:8080/cities{?page,size,sort}",
      "templated" : true
    }
  },
  "_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" : "CASTANER",
      "county" : "LARES",
      "stateCode" : "PR",
      "postalCode" : "00631",
      "latitude" : "+18.269187",
      "longitude" : "-066.864993",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/cities/20"
        }
      }
    } ]
  },
  "page" : {
    "size" : 20,
    "totalElements" : 42741,
    "totalPages" : 2138,
    "number" : 0
  }
}
```

5. Try the following `curl` statements to see how the application behaves:

```bash
$ curl -i "localhost:8080/cities?size=5"
$ curl -i "localhost:8080/cities?size=5&page=3"
$ curl -i "localhost:8080/cities?sort=postalCode,desc"
```

   Next we'll add searching capabilities.

Adding Search

1. Let's add some additional finder methods to `CityRepository`:

```java
@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);
```

2. Build the application:

```bash
$ ./gradlew assemble
```

3. Run the application:

```bash
$ java -jar build/libs/cities-0.0.1-SNAPSHOT.jar
```

4. Access the application again using `curl`. Notice that hypermedia for a new `search` endpoint has appeared.

```bash
$ curl -i "localhost:8080/cities"
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Application-Context: application
Content-Type: application/hal+json
Transfer-Encoding: chunked
Date: Tue, 27 May 2014 20:33:52 GMT

{
  "_links" : {
    "next" : {
      "href" : "http://localhost:8080/cities?page=1&size=20"
    },
    "self" : {
      "href" : "http://localhost:8080/cities{?page,size,sort}",
      "templated" : true
    },
    "search" : {
      "href" : "http://localhost:8080/cities/search"
    }
},
// (Remainder omitted...)
```

5. Access the new `search` endpoint using `curl`:

```bash
$ curl -i "localhost:8080/cities/search"
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Application-Context: application
Content-Type: application/hal+json
Transfer-Encoding: chunked
Date: Tue, 27 May 2014 20:38:32 GMT

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

Note that we now have new search endpoints for each of the finders that we added.

6. Try a few of these endpoints. Feel free to substitute your own values for the parameters.

```bash
$ curl -i "http://localhost:8080/cities/search/postalCode?q=75202"
$ curl -i "http://localhost:8080/cities/search/name?q=Boston"
$ curl -i "http://localhost:8080/cities/search/nameContains?q=Fort&size=1"
```

Pushing to Cloud Foundry

1. Create an application manifest in `manifest.yml`:

```yml
---
applications:
- name: cities
  host: cities-${random-word}
  memory: 512M
  instances: 1
  path: build/libs/cities-0.0.1-SNAPSHOT.jar
  timeout: 180 # to give time for the data to import
```

2. Push to Cloud Foundry:

```bash
$ cf push

...

1 of 1 instances running

App started

Showing health and status for app cities...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: cities-undeliverable-iatrochemistry.cf.mycloud.com

     state     since                    cpu    memory         disk
#0   running   2014-05-27 04:15:05 PM   0.0%   433M of 512M   128.9M of 1G
```

3. Access the application at the random route provided by CF:

```bash
$ curl -i cities-undeliverable-iatrochemistry.cf.mycloud.com/cities
```

Last updated 2015-06-13 15:39:17 EDT