Hands On Lab 1 - Spring Data REST

**Developing RESTful APIs with Ease**

[Introduction 1](#_Toc18197)

[What You Will Build 2](#_Toc3528)

[Scaffolding an App with Spring Boot 3](#_Toc9640)

[Creating RESTful APIs for JPA Entities with Spring Data REST 4](#_Toc24589)

[Issuing requests to the Spring Data REST endpoints 9](#_Toc6397)

[Data Validation 11](#_Toc11073)

[Conclusion 17](#_Toc7400)

Learn how to develop REST APIs with ease by using Spring Boot and Spring Data REST.

In this lab, you will learn how to develop REST APIs with ease by using Spring Data REST and Spring Boot together. Throughout the lab exercise, you will scaffold a new Spring Boot application, create a JPA entity, and use Spring Data REST to provide some basic operations over it. Besides that, you will also learn how to validate the data your API is dealing with.

## **Introduction**

Developing RESTful APIs is a task that most developers have faced at least once in their lives. Nowadays, with the increasing popularity of front-end frameworks like Angular, React, and Vue.js and with the mass adoption of smartphones, RESTful APIs became the most popular approach backend developers leverage to provide an interface for their applications. Knowing that, the Spring community, as you will see in this article, worked hard to build [Spring Data REST](https://spring.io/projects/spring-data-rest" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank), a framework that facilitates the life of developers while creating these APIs.

Spring Data REST builds on top of Spring Data repositories, analyzes your application’s domain model and exposes hypermedia-driven HTTP resources for aggregates contained in the model. — [Spring Data REST](https://spring.io/projects/spring-data-rest" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank)

## **What You Will Build**

To learn about Spring Data REST, you will build a simple RESTful API that exposes CRUD (Create, Retrieve, Update, and Delete) operations over an entity called Ad. The Ad entity, in this case, stands for advertisements and will be used to represent some product or service that a user is trying to sell. As such, your API will enable users to manipulate ads that hold information like title, owner, description, and price.

As mentioned before, while developing this RESTful API, you will learn how to:

* provide an API with different basic operations over the entity created (Ad);
* And validate data.

## **Scaffolding an App with Spring Boot**

For starters, before diving into developing your RESTful API, you will need to scaffold your new application. To do so, open [the Spring Initializr website](https://start.spring.io/" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank) in your browser and fill the presented form as follows:

* Generate a Maven Project with Java and Spring Boot 2.X
* Group: com.auth0.
* Artifact: ads.

Then, on the Dependencies section, use the search for dependencies field to add five dependencies:

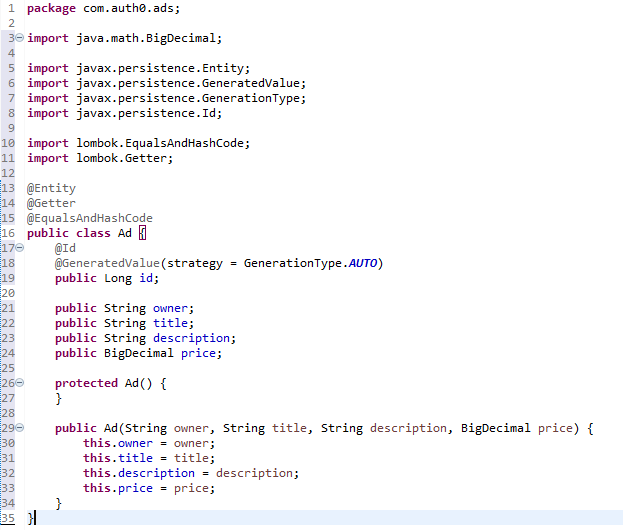
* Web: A library that helps you develop web applications with Tomcat and Spring MVC.
* Rest Repositories: The library that will allow you to expose your database as a RESTful API.
* JPA: The Java Persistence API library that will help you map SQL databases to objects and vice-versa.
* Lombok: Java annotation library that [helps you code faster by reducing boilerplate code](https://auth0.com/blog/avoiding-java-boilerplate-code-with-project-lombok/" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank).
* H2: An embedded SQL database that you will allow you to spin up an in-memory database instead of having to install and configure a new one.

After filling this page, click on the Generate Project button to download your new application. When done, you will have to extract the downloaded .zip file and move the extracted folder to your Spring Tool Suite workspace.

With that in place, use your IDE to import your new project.

## **Creating RESTful APIs for JPA Entities with Spring Data REST**

Now that you have scaffolded a new Spring Boot project and that you have opened it in your IDE, the first thing you will do is to create the JPA entity that you will expose with the help of Spring Data REST. To do so, create a new class called Ad inside the com.auth0.ads package, and add the following code to it:

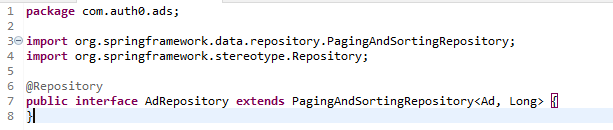


As you can see from the code above, you are defining a new JPA @Entity called Ad to hold five different fields. Most of these fields are self-explanatory (their names should be enough for you to understand what they will hold). The only one that might need some explanation is the Long id field. As you can see, this field is marked with two annotations:

* @Id: This annotation marks the field as the unique identifier of the ad (i.e., the primary key in the database).
* @GeneratedValue: This annotation tells JPA that the database will need to supply its value. In this case, the database will auto-generate (GenerationType.AUTO) this field, no matter how (the strategy to do so depends on what database you are using).

After creating the Ad JPA entity, you will need to focus on creating a class that will allow you to interface with the database. Also, you will need to map the database operations into RESTful API endpoints so external clients can use them. Sounds like a lot of work? On the contrary! This is where Spring Data REST shines.

To achieve both things described in the last paragraph (map to the database and expose the operations in your API), you will need to do only one thing. You will need to create an interface called AdRepository (inside the com.auth0.ads package) and add the following code to it:



That's it! With this interface in place, you are ready to run your application and start issuing requests to it.

What is happening here is that Spring Boot (a framework that is bound to the "convention over configuration" strategy) and Spring Data REST identify that you defined an interface that extends PagingAndSortingRepository and work together to create a bunch of endpoints for you.

If you check the JavaDocs of the PagingAndSortingRepository interface, you will see that this interface is an ["extension of CrudRepository to provide additional methods to retrieve entities using pagination and sorting"](https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank).

More specifically, as described on [the Core Concepts documentation](https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html" \l "repositories.core-concepts" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank) of the Spring Data library, together, PagingAndSortingRepository and CrudRepository add the following methods to your application to allow it to manipulate your entities in a SQL database:

* save(S entity): A method that enables the app to save (insert or update) entities.
* findOne(ID primaryKey): A method that enables the app to retrieve an entity based on its primary key.
* findAll(): A method that enables the app to retrieve all entities saved to the database.
* Long count(): A method that returns how many entities exist on the database.
* delete(T entity): A method to remove a specific entity from the database.
* exists(ID primaryKey): A method to check if a particular entity, based on a primary key, exists on the database.
* findAll(Sort sort): A method to return a list of all entities, sorted by some criteria, saved in the database.
* findAll(Pageable pageable): A method to return subsets (pages) of the entities saved in the database.

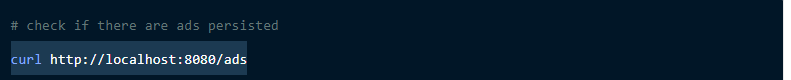
Then, as you are using Spring Data REST, this library creates RESTful endpoints to exposes all the methods defined by PagingAndSortingRepository and CrudRepository.

**Start the application from the Spring Boot Dashboard in Spring Tool Suite.**

After the application starts executing (it will take a few seconds for it to be ready), you can start sending HTTP requests to its endpoints. To do so, you can use [a graphical HTTP client like Postman](https://www.getpostman.com/" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank), or you can open a new terminal and use a tool like curl. The next subsection will show you how to use curl to issue some requests to your new application. However, translating them to Postman or a similar client should not be a problem.

## **Issuing requests to the Spring Data REST endpoints**

For starters, to confirm that your application is working as expected, you can issue the following request (you might need to run the command below in a new terminal or, as mentioned, you can use Postman):



Running this command will make your API return all the ads persisted on the database (which will be none: "ads" : [ ]), and a few other things like:

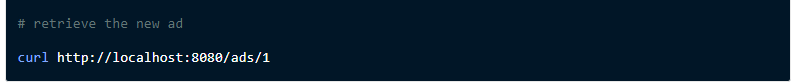
* page.size: the number of ads that the API will return on each page (20).
* page.totalElements:: the number of ads returned on this page (0).
* page.totalPages:: the number of pages available (0).
* page.number:: the current page number (0).

As you haven't created any entity yet, the results won't be very interesting. So, the next thing you can do is to issue a request to insert a new ad in your API:



Issuing this command will output the details of the ad created (all the properties above) along with a link to the ad itself (self.href). You can ignore any warnings related to resolving ' host: POST’ or ‘ host: I’.

If you use this link in a curl command, you will retrieve the details of the ad:



Now, if you run the command to see the list of persisted ads again:



You will see that the ads array includes your new ad and that the page properties (size, totalElements, etc.) are updated accordingly.

After that, if you want, you can delete the ad by issuing the following command:



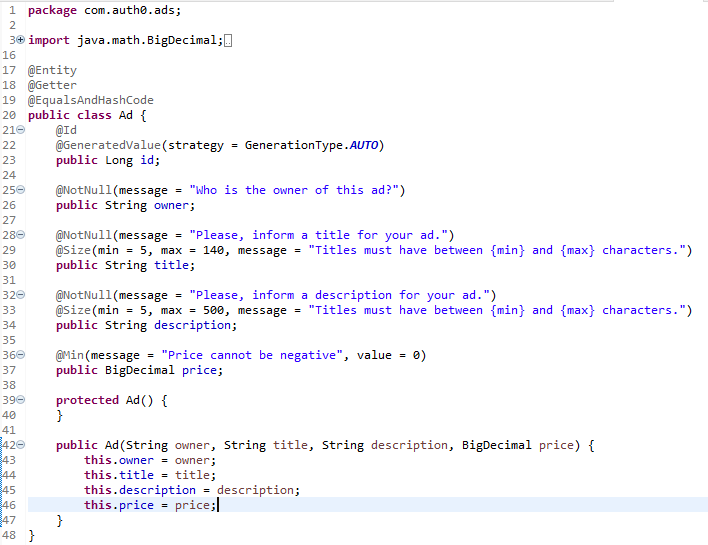
After scaffolding your application, you just had to create two things (the Ad entity and the AdRepository interface), and you got a runnable RESTful API.

## **Data Validation**

After seeing Spring Data REST in action, you might be thinking: "Ok, this library does facilitate creating RESTful APIs, but I need to validate the data before persisting it to the database. Will it be easy to validate my data?" The answer to that is yes! By using [the Bean Validation Java specification](https://beanvalidation.org/" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank), validating your data is as easy as adding some annotations to the fields in your JPA entity.

#### ["The Bean Validation specification allows you to validate data in your Spring Boot APIs very easily."](https://twitter.com/intent/tweet?text="The Bean Validation specification allows you to validate data in your Spring Boot APIs very easily." via @auth0 https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease" \t "https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/_blank)

If you want some proof, open the Ad class and update its code to look like this:



On the new version of this entity, you are importing and using three new annotations:

* Min: This annotation allows you to make a field accept only data that has a minimum value. For example, on the Ad class, you are making sure you won't get negative (@Min(..., value = 0)) values for the price field.
* NotNull: This annotation allows you to mark fields not to accept null values. In the case above, you are marking owner, title, and description as required fields.
* Size: This annotation allows you to make a field accept only values that have a length between a predefined range. For example, on the code above, you are making sure you won't persist ads with a title that contains less than 5 characters or more than 140 characters.

After updating the Ad class, you can rerun your application. When the app finishes booting, you can issue requests like the following to confirm that the restrictions are working:

# won't work, too few characters on the title property



# won't work, negative price



# won't work, no description



# will work, all properties contain valid values



While issuing the invalid requests above (i.e., the first three), you will see that your app returns a response that looks like this:



Also, check the error messages in your STS console.

Although this response is sufficient for the calling client to understand that something went wrong, it doesn't help to identify what the problem was. To fix that, you can create a new class called RestExceptionHandler inside the com.auth0.ads package and add the following code to it:



With this class, you are creating and registering a global exception handler (ResponseEntityExceptionHandler) for your application. Then, inside this handler, you are defining a method that will catch instances of TransactionSystemException to unwrap the real problem (ex.getCause()). If the unwrapped exception is not an instance of RollbackException, then this method throws the underlying exception.

If the unwrapped problem is an instance of RollbackException, then this method unwraps the cause one more time (cause.getCause()). This time, the goal is to check if the second unwrap will result in an instance of ConstraintViolationException. If this is not the case, the unwrapped result is thrown. If this is the case, this method generates a list of messages to send back as a response.

After creating this exception handler, if you restart your app and issue bogus requests, you will get some friendly messages. For example, if you issue the following request:



You will get a list of messages saying that the "price cannot be negative" and to "please, inform a description for your ad."



## **Conclusion**

In this lab, you learned about how easy it is to develop RESTful API with Spring Data REST and Spring Boot. More specifically, you started by using the Spring Initializr website to scaffold a new application. After that, you used Spring Data REST to expose API endpoints to manipulate a JPA entity. Then, in the end, you learned how to validate data.

With that in place, you are ready to start developing production-ready, secure APIs with Spring Boot, and Spring Data REST.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*