|  |
| --- |
|  |
| Review the following two resources: | |
|  | |

|  |
| --- |
| https://www.baeldung.com/spring-data-rest-relationships |
|  |

|  |
| --- |
| https://hellokoding.com/jpa-many-to-many-relationship-mapping-example-with-spring-boot-maven-and-mysql/ |
|  |

|  |
| --- |
|  |
|  |

For a one-to-one relationship, two entity classes need to be defined using the @OneToOne annotation. Each association resource should have different names or else a JsonMappingException message will appear instructing to disambiguate the association resources. The association names can also be customized using the rel attribute by customizing the endpoint. An example of this is the @RestResource annotation. By extending the CrudRepository interface, two repository interfaces can be created to expose the entities as resources. With a specific instance to work with, the API returns the JSON object. For instance, if the entities Restaurant and Location were used, a PUT API would bind the association to the body containing the URI of the resource, leading to adding the location to the restaurant. Additionally, using a GET API, it would lead to the code retrieving the addresses of the locations. Finally, a DELETE method would remove the association of the location to a certain restaurant. In all, a one-to-one relationship with spring data takes two classes of data and combines them to work together such as locations and restaurants.

The one-to-many relationship is defined with @OneToMany and/or @ManyToOne annotations. These annotations can have an optional @RestResource as well to customize the association resource. Using the previous example of Restaurants and Locations, this would now include the association of Food to represent the “many” end of the relationship with the Restaurant entity. For this to be successful, a repository needs to be created, such as FoodRepository. To add a food to the restaurant, a Food instance needs to be created using the /food collection resource and the POST method. Using the PUT method associates the food with the restaurant and one can verify that the food is at that restaurant using the GET method. When a GET method is used, a returned JSON object will contain a food array. Just as before, a DELETE method would remove a food from the selected restaurant.

Finally, a many-to-many relationship is defined with the annotation of @ManyToMany with the added @RestResource. Using the same example of Restaurant, Location, and Food, a new model class of Sides can be created. Then there is an association between the Food and the Sides entities and a new Sides repository is created. Just as before, the resources must be created before an association can be established. One Side can be associated with multiple Foods, or multiple Sides can be associated with a single Food item using a PUT method, which can support the media type of text/uri-list which can receive multiple URIs. Sending multiple URIs, however, requires separation by a line break. The GET method is used to verify that the Food have been associated with the Sides. Also as before, the DELETE method can remove an association.

In conclusion, the one-to-one relationship is an association between two items. The one-to-many relationship lengthens the association with one entity to multiple other items. The many-to-many relationship takes those multiple other items and associates it with additional multiple items.