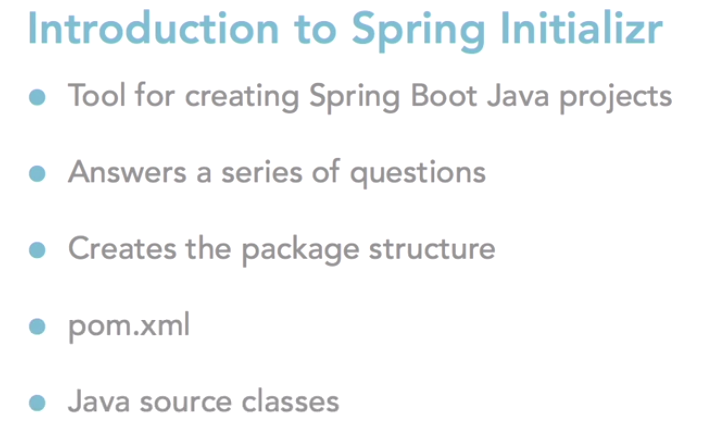


To build a Java application, the first step is to create a Java project. Most Java projects rely on third-party Java archive dependencies, and these third-party archives usually have dependencies of their own. On top of that, each version of the dependencies relies on other versions. Managing all these dependencies is a nightmare that Java developers have nicknamed JAR hell.

To avoid JAR hell, we use build dependency management systems like Maven or Gradle. But even with Maven and Gradle, versioning between individual .jar files can be a nuisance.

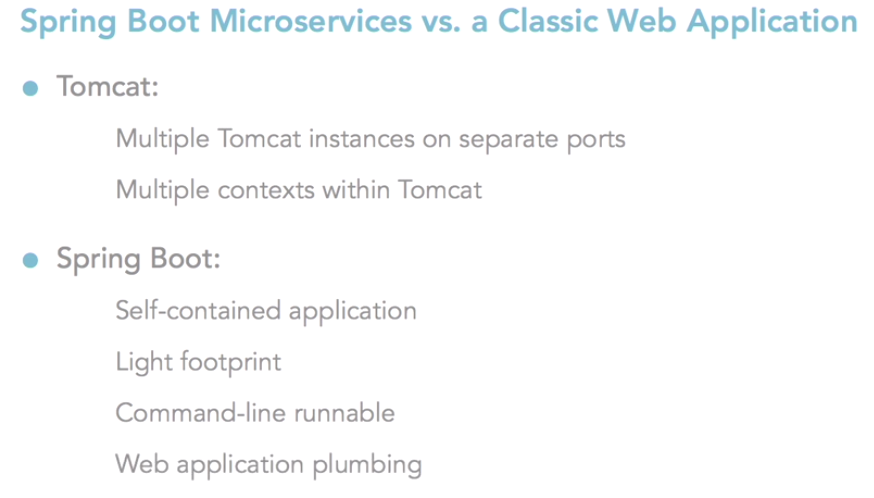
Spring Boot recognizes this, and created the notion of a Spring Boot Starter, which bundles several dependencies into a grouping that is easier to manage. There are a lot, and I mean a lot of Spring Boot Starter dependencies so even cobbling together a project on your own can be difficult.



<https://start.spring.io/>

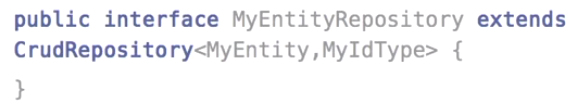
Adding the **@SpringBootApplication** annotation to this class with the main method tells Java here is where our Spring Boot Microservice starts. Command line parameters or special startup logic resides here.

With this one pom file, and one Java source file, we can now build and run the project as a Spring Boot application.



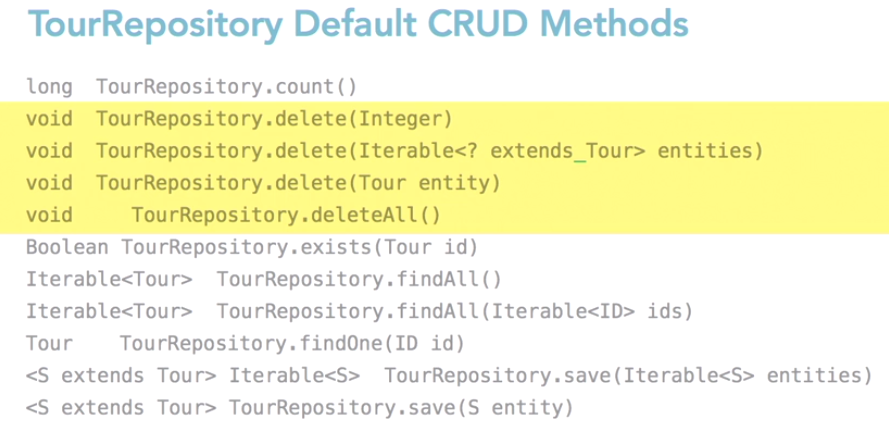
Beauty of a microservice, it's completely self-contained with a light footprint and it's runnable with a single command. This makes it a favorite with the dev ops and cloud computing crowds because the web application and server start up can be scripted as one command.



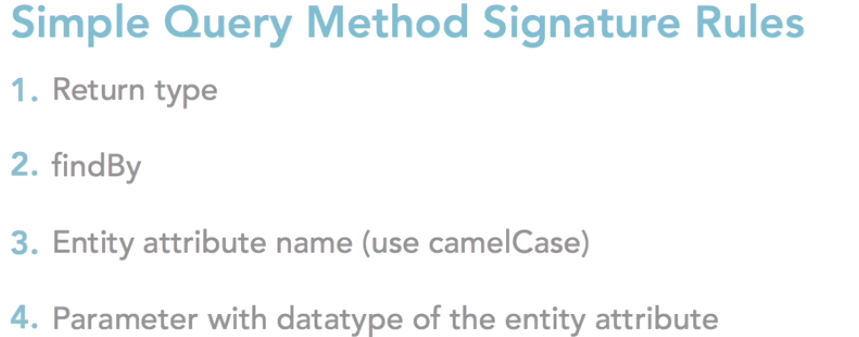


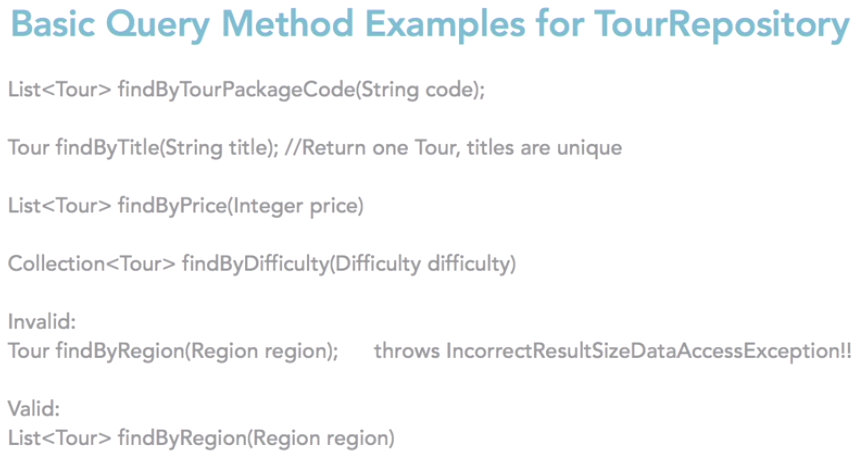
To use Spring Data JPA for a JPA entity we declare a Java interface class that extends from CrudRepository, where T is the domain type the repository manages and ID is the type of the ID of the entity the repository manages.

Ex:

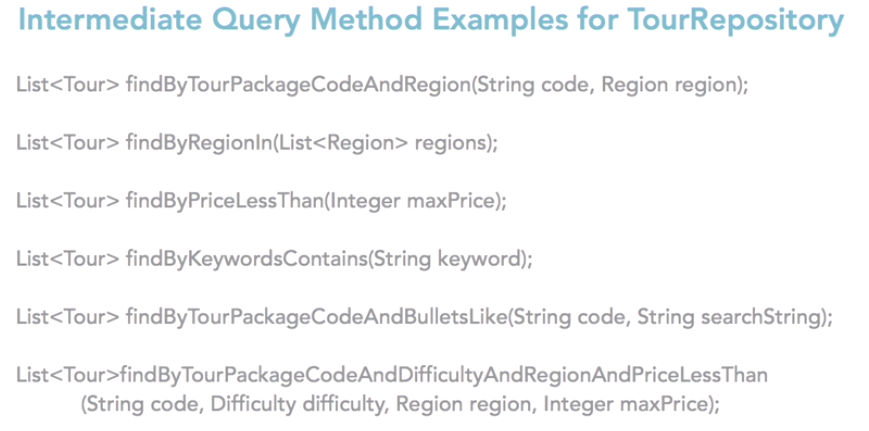


Spring Data JPA eliminates ugly boilerplate querying code. To create a simple query method, you only need to declare the return type. Begin the signature with findBy, followed by an attribute name in camelCase and query parameters whose type matches the attribute type.

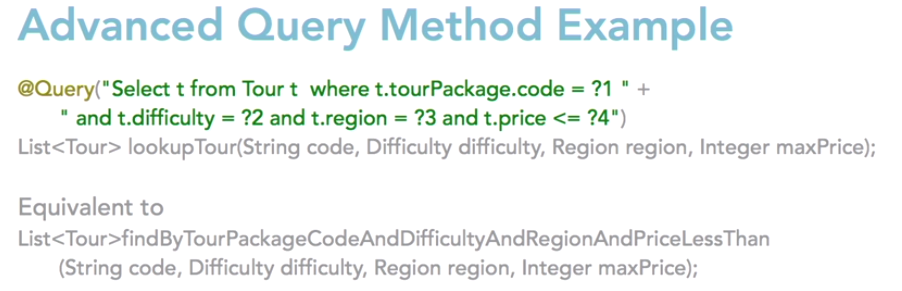




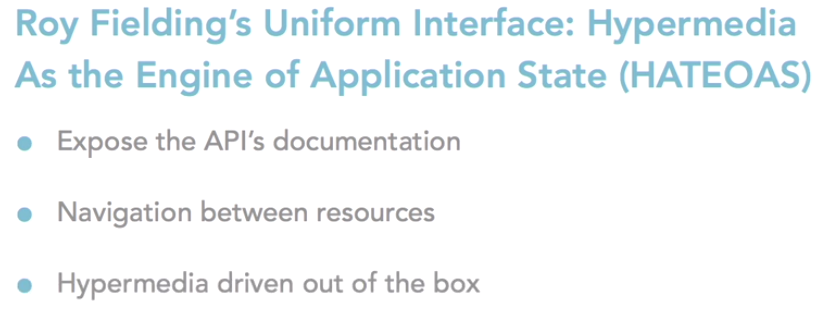




As you can see, the signatures on the methods become longer as the query method becomes more complex. You could leverage complex JQL queries inside a repository interface by using the @Query annotation.



**HATEOS:**



Spring Data REST which is a service for creating Hypermedia driven RESTful APIs. what is a Hypermedia driven RESTful API?

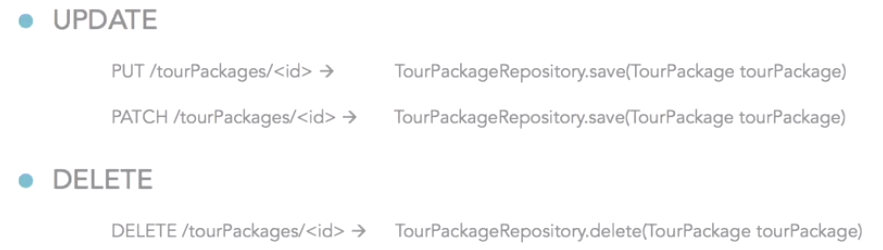
According to Roy Fielding, who first published the REST API Specification, an API is not truly Restful unless it follows a uniform interface. One of his constraints for a uniform interface is followed when Hypermedia as an Engine of Application State, or HATEOAS, is employed.

A RESTful API should do more than expose resource endpoints over HTTP. It should also expose the API's documentation and automatically provide navigation between resources. Hypermedia driven APIs accomplish just that. The implementers of Spring Data REST agreed, and that's why Spring Data REST APIs are Hypermedia driven out of the box. No extra configuration is needed. Spring Data REST employs the Hypermedia Application Language, or HAL standard, to associate resource objects to one another.

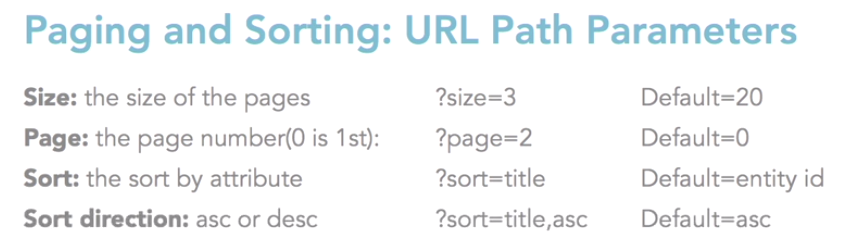
At application start up, Spring Data REST scans the classes and finds all the Spring Data repositories, creates an endpoint that matches the entity name, appends an s and exposes the operations as APIs. So here are the mappings for CREATE, the mapping for READ, the mapping for UPDATE and DELETE.At application start up, Spring Data REST scans the classes and finds all the Spring Data repositories, creates an endpoint that matches the entity name, appends an s and exposes the operations as APIs. So here are the mappings for CREATE, the mapping for READ, the mapping for UPDATE and DELETE.At application start up, Spring Data REST scans the classes and finds all the Spring Data repositories, creates an endpoint that matches the entity name, appends an s and exposes the operations as APIs. So here are the mappings for CREATE, the mapping for READ, the mapping for UPDATE and DELETE.At application start up, Spring Data REST scans the classes and finds all the Spring Data repositories, creates an endpoint that matches the entity name, appends an s and exposes the operations as APIs. So here are the mappings for CREATE, the mapping for READ, the mapping for UPDATE and DELETE.Bottom of Form

At application start up, Spring Data REST scans the classes and finds all the Spring Data repositories, creates an endpoint that matches the entity name, appends an s and exposes the operations as APIs. So here are the mappings for CREATE, the mapping for READ, the mapping for UPDATE and DELETE.

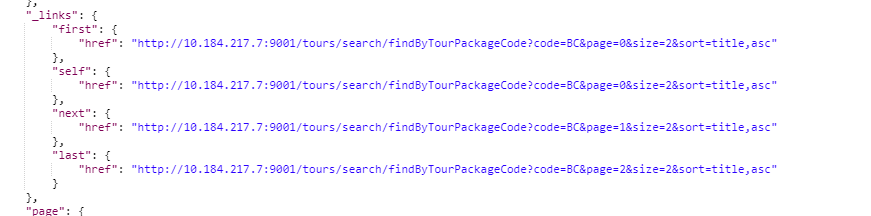




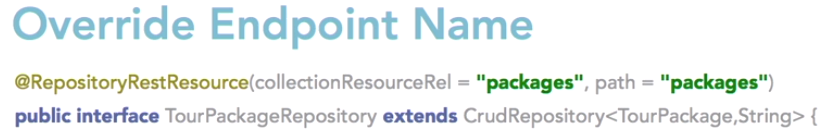
Every endpoint that Spring Data REST exposes also creates a /search resource



The response contains the hrefs called First, Last, Pre, and Next. Front end developers love this. By using these hrefs, they don't have to calculate the last page number, nor keep state of the current page. We also see at the bottom, there is some meta data. If the client chooses not to use those links, the API also provides more page meta data at the bottom of the response packets, so they can calculate it themselves. In the past, we would have had to code all these features ourselves.







Earlier we saw that we cannot only look up entities but also create update and delete them with HTTP post, put, patch and delete. But I do not think we want to allow the public to modify our database in this way.

There are two annotations to accomplish this.@RepositoryRestResource is used to control access at the class level and @RestResource is used to control access at the method level.

We can also use @RestRepositoryResource to override the default end point name. In this example, we set the TourPackages endpoint name to packages. So, in TourRepository, we'll use our IDE to help us override methods. So, the methods we're overriding here are save and delete.



Annotating a class with @RestController tells Spring Web MVC that this class follows RESTful web service stereotypical behavior.



A RestController class mediates between web requests and the internals of the application. It is responsible for directing requests to the appropriate services, then sending a response back to the client. In Spring Data REST, the framework itself mediates between web requestsand the persistent domain model.



