7. Creating REST services

-Nowadays, web browser no longer reigns as the primary means of accessing the I. Mobile devices, tablets, smart watches.. are commonplace. Many browser-based apps are running JavaScript apps rather than letting the browser be a dumb terminal for server-rendered content

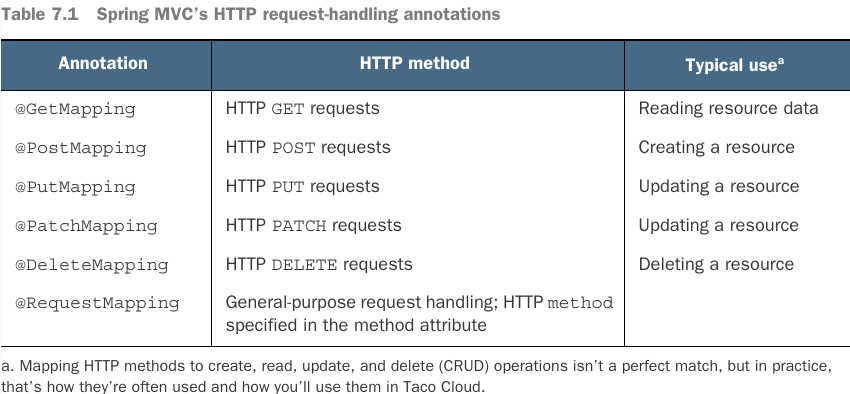
-Many apps have adopted a common design where the user interface is pushed closer to client and the server exposes an API which all kinds of clients can interact with back-end.

# 7.1 Writing RESTful controllers

-REST APIs aren’t much different form websites:

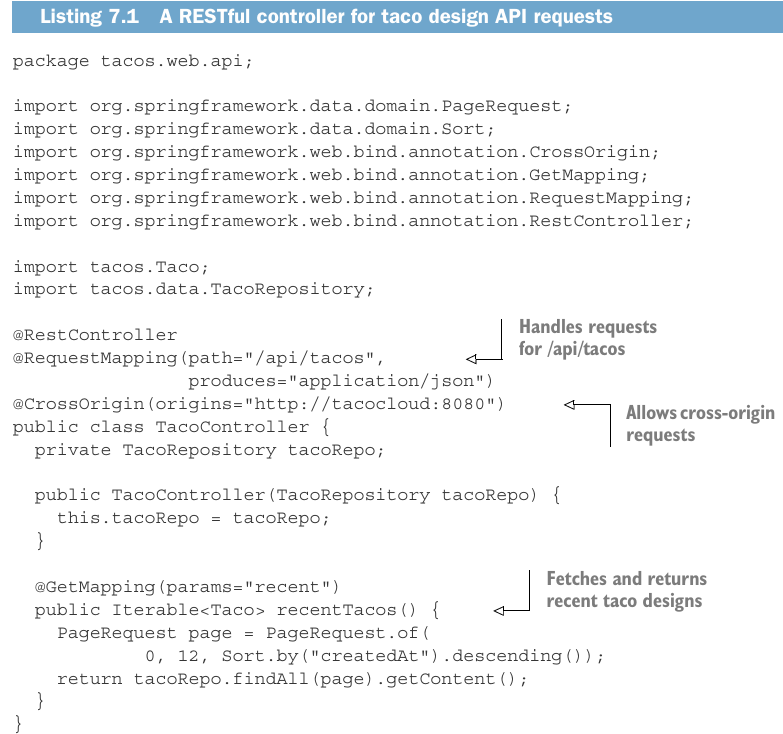
+Both involve responding to HTTP requests.

+Instead of responding to requests with HTML, REST APIs responds with a data-oriented format: JSON, XML



## 7.1.1 Retrieving data from the server

-We create an endpoint that handles GET requests for /api/tacos which include a “recent” parameter and responds with a list of recently designed tacos.



+This controller is REST controller, as indicated by **@RestController**: marks a class for discovery by component scanning + all handler methods in controller should have return value written to the body of the response, rather than being carried in the model to a view for rendering.

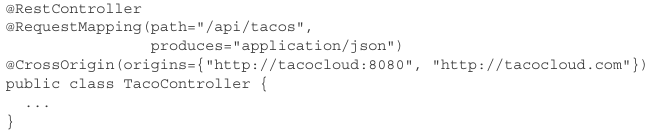
+You can annotate TacoController with **@Controller** but all handler methods **@ResponseBody** to achieve the same result. Another way is return **ReponseEntity**

+**produces** attributes: the method handles requests only if the client sends a request with an Accept header that include “application/json”. -> limit your API to only producing JSON results. + allow for another controller to handle request with the same path.

+You can set produces to an array of String for multiple content types:

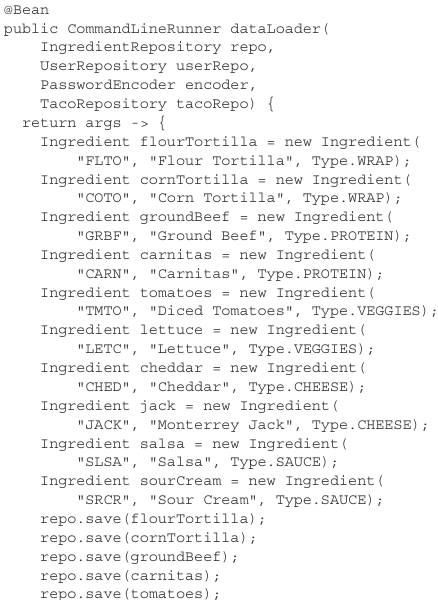


+**@CrossOrigin**: It’s common for JavaScript-based user interface, to be served from a separate host and/or port from API and the web browser will prevent your client from consuming the API. This restriction can be overcome by including **CORS** (cross-origin resource sharing) headers in server responses. In the example, @CrossOrigin allows clients from localhost, port 8080 to access API. The **origins** attribute accepts an **array**:



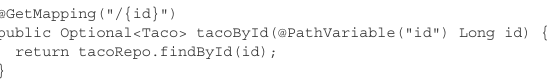
-Start of a Taco Cloud API for you client: For development testing purpose, you want to use command-line **curl** or **HTTPie** (https://httpie.org/) to poke about the API.

-If the database is empty:





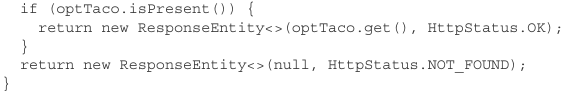
-Offer an endpoint that fetches a single taco by its ID.



+The actual value in request is given to id parameter, which is mapped to {id} by **@PathVariable**

-You can set up **status code**:



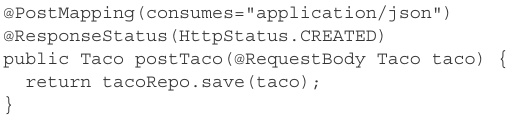


+If the taco is found, you wrap Taco object in ResponseEntity with HTTP status OK.

## 7.1.2 Sending data to the server

-Disable csrf:  in SecurityConfig

-Although you can use @CommandLineRunner bean to preload the database with some test taco data, you can write method handle request containing taco designs and save them to database:



+**consumes**: say that the method will only **handle** **requests** whose **Content-type** matches application/json

**+@RequestBody**: the body of request should be converted to Taco object and bound to parameter.

->important

[+@ResponseStatus(HttpStatus.CREATED)](mailto:+@ResponseStatus(HttpStatus.CREATED)): HTTP Status 201: request successful + resource was created.

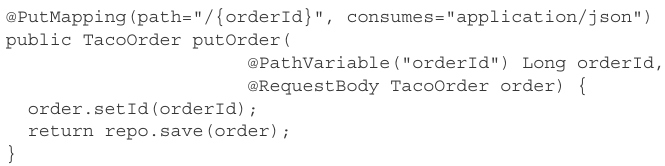
## 7.1.3 Updating data on the server

-We can use PUT or PATCH to update resources.

+**PUT**: the opposite of GET: send data from client to server. It performs a **wholesale replacement** operation rather than an update operation.

+**PATCH**: perform a patch or **partial update** of resource

-Example: change the address on an order by PUT:

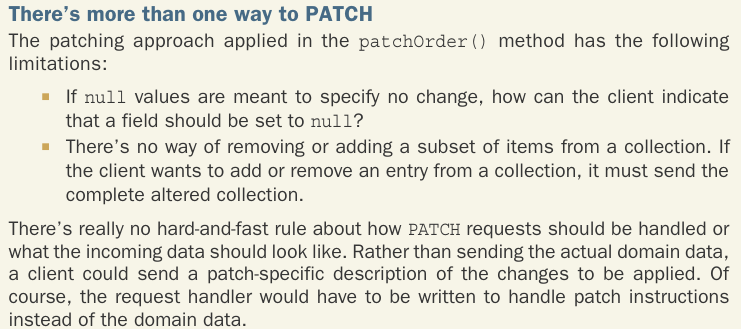


+It **requires** the **client** **submit** the **complete order data** in PUT request. If any order’s **properties** are **omitted**, that property’s value would be **overwritten** with **null**.

-Use HTTP PATCH:

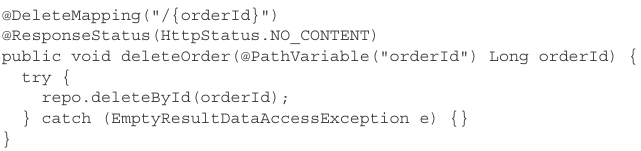


+Instead of completely replacing order with the new data sent in, it inspects each field of incoming object and applies any non-null values to existing order.->allow client to send only properties that should be changed + enable server to retain existing data.



## 7.1.4 Deleting data from the server

-Delete order:



+EmptyResultDataAccessException: You can return ResponseEntity setting body to null and HTTP.NOTFOUND

+HttpStatus.NO\_CONTENT: no need to communicate data back to client for a resource that no longer exists->responses have no body

# 7.2 Enabling data-backed services

-**Spring Data REST** is another member of Spring Data family that **automatically** **creates** **REST APIs** for **repositories** created by **Spring Data**.

-Add **dependency**:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-rest</artifactId>

</dependency>-rest

+The app gets autoconfiguraiton that enables automatic creation of a REST API for any repositories that were created by Spring Data (JPA, Mongo…)

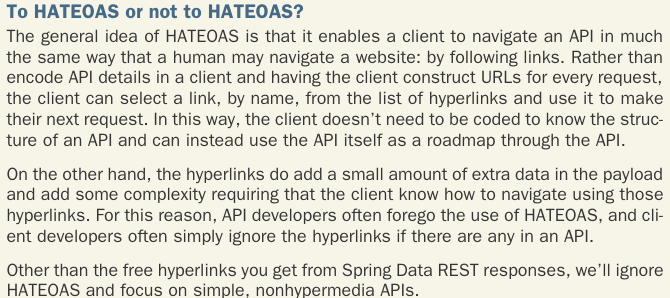
-Based on the set of repositories you’ve defined for Taco Cloud, you should be able to perform GET requests for tacos, ingredietns, orders, users.

+Get a list of all ingredients: 



+You get an endpoint for ingredients + **hyperlinks**. These hyperlinks are implementations of Hypermedia as the Engine of Application State (**HATEOAS**). A client can use these as a guide for navigating API and perform the next request.

+Spring HATEOAS (https://spring.io/projects/spring-hateoas) provides general support for adding hypermedia in Spring MVC controller responses.

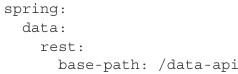


+Follow the self link for flour tortilla entry:



+You can POST to /ingredients and DELETE /ingredients/FLTO.

-You can set a bas path for API:





## 7.2.1 Adjusting resource paths and relation names

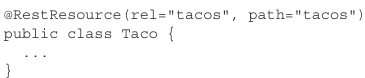
-Spring Data REST exposes a **home resource** that list **links** for all **endpoints**:





+Error right now.

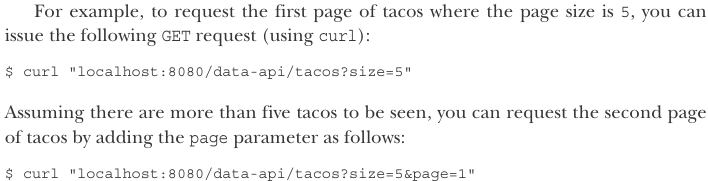
-You can tweak the **relation name + path**:



## 7.2.2 Paging and sorting

-By **default**, requests to a collection resource will return up to **20 items** per page from the **1st page**.

-You can change the page and the page size:



+Notice: the page parameter is **zero-based**.

-sort para lets you sort the list by any property of entity.

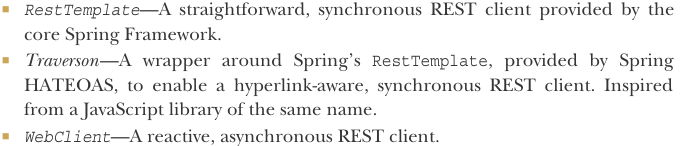


# 7.3 Consuming REST services

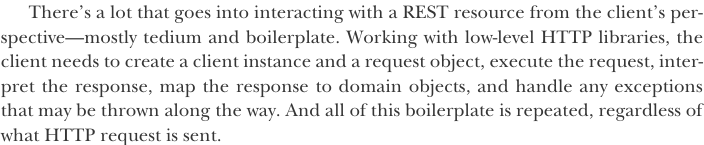
-A movie without an audience is kind of like an API without a client.

-It’s uncommon for Spring apps to both provide an API and make requests to another app’s APi (This is the world of microservice). ->worthwhile to look at how to use Spring to interact with REST APIs.

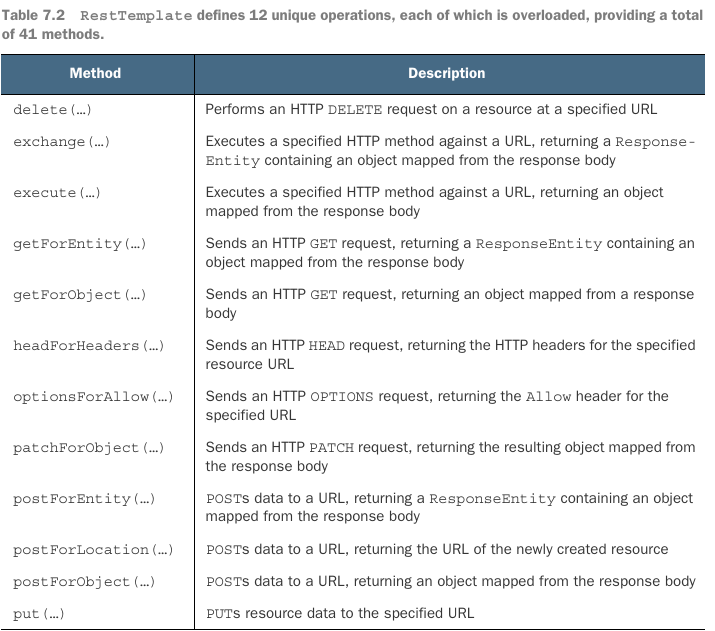
-A Spring app can consume a REST API:



+For now, we focus on creating clients with RestTemplate



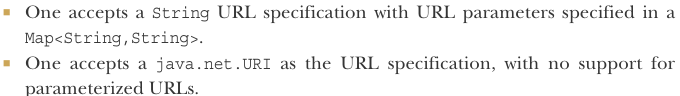
-RestTemplate provides 41 methods for interacting with REST resources.



-With the exception of TRACE, RestTemplate has at least 1 method for each of the standard HTTP methods. Execute() and exchange() provide lower-level, general-purpose methods for sending request with any HTTP method.

-Most of methods in table are overloaded into 3 forms:





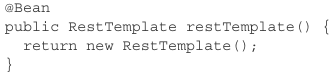
->Once getting to know 12 operations provided by RestTemplate, you’ll be well to write resource-consuming REST clients.

-Use RestTemplate:

+Create an instance at the point you need:

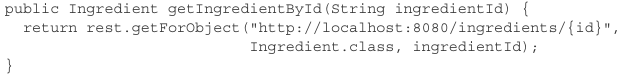


+Or declare it as a bean and inject it:



## 7.3.1 GETting resources

-Fetch an ingredient from TacoCloud API using **getForObject():**



+1st para: String URL, 2nd para: the type that the response should be bound to, 3rd: fill in {id} placeholder in URL

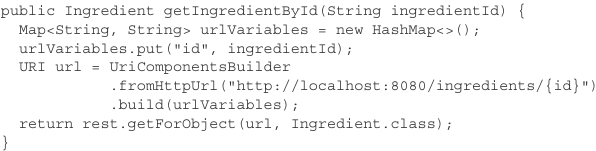
+Or use Map to specify URL variables:





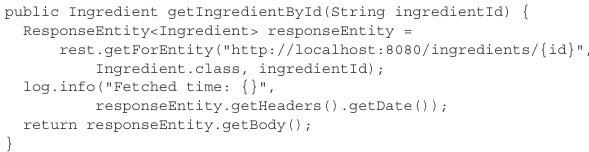
The value of ingredientId is mapped to a key of id. When request is made, {id} is replaced by the map entry whose key is id.

+Use URI:



-getForEntity(): instead of return a domain object that represents the respone’s payload, it returns a ResponseEntity object wrapping that domain object. It gives access to additional response details (headers..)

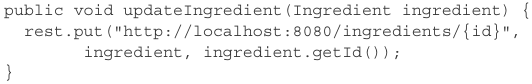
+Also inpsct Date header from response:



+getForEntity() is overloaded with same parameters as getForObject()-> can use URL variables + URI

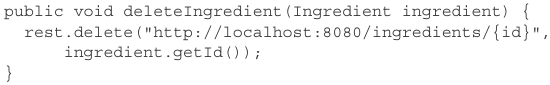
## 7.3.2 PUTting resources

-Use put(): Example: replace ingredient resource with data from a new Ingredient object



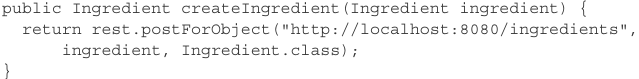
## 7.3.3 DELETEing resources

-Use delete():



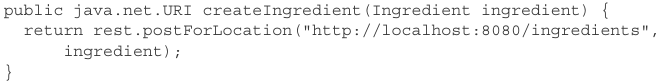
## 7.3.4 POSTing resource data

-postForObject(): receive the newly created Ingredient resource after POST request



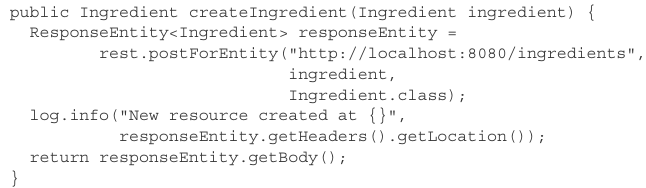
+4th parameter could be a Map of URL variable value or variable list of parameters to substitute into URL

-If need for location, **postForLocation()**



+URI returned is derived from response’s Location header.

-Get both location + response payload, postForEntity():



# -Summary

