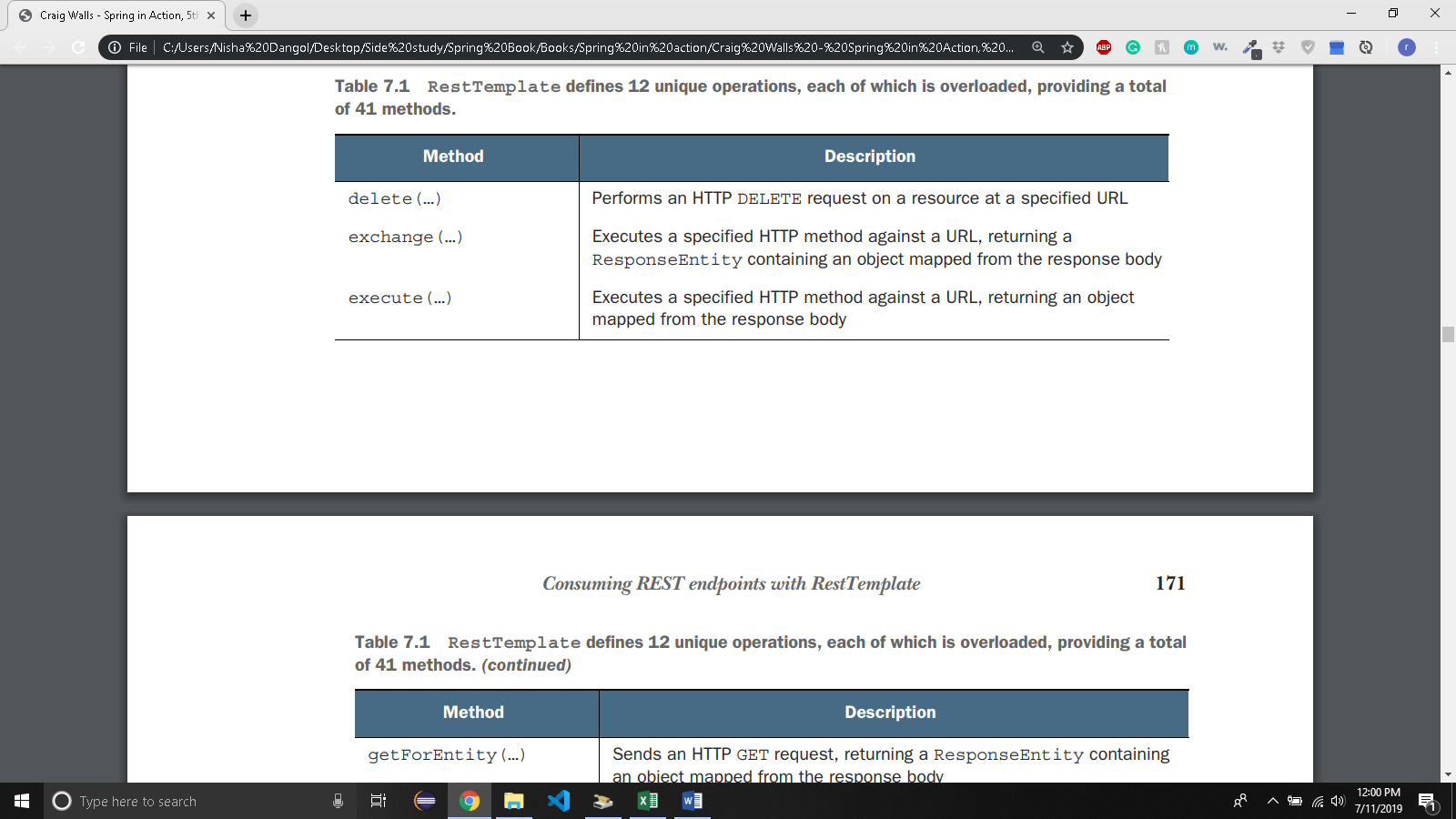
CHAPTER 7 – CONSUMING REST SERVICES

* A spring application can consume REST API WITH:
  + RestTemplate
    - A straightforwared synchronous REST client provided by the core spring framework
  + Traverson
    - A hyperlink-aware, synchronous REST client provided by spring HATEOAS.
  + WebClient
    - A reactive, asynchronous REST client introduced in spring 5.
* 7.1 CONSUMING REST ENDPOINT WITH RESTTEMPLATE
  + RestTemplate frees you from dealing with the tedium of consuming REST resources
  + The 12 most important REST operations are :
  + 
  + 
  + To use RestTemplate you need to either create and instance at the point you need it like this:

RestTemplate rest = new RestTemplate();

* + Or, you can declare it as a bean and inject it where you need it:

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

* + Has 4 primary methos: GET, PUT, DELETE, POST
  + 7.1.1 GETTING RESOURCES:
    - getForObject
    - Suppose you want to fetch an ingredient from the Taco cloud API by its id.
    - Assuming API isn’t HATEOAS enable, you can use getForObject() to get the ingredient

Public Ingredient getIngredientById(String ingredientId){

return rest.getForObject([http://localhost:8080/ingredients/{id}](http://localhost:8080/ingredients/%7bid%7d),

Ingredient.class,

ingredientId);

* + - * The ingredientId parameter is used to fill the {id} placeholder.
      * Although there’s only one URL variable in this example, it’s important to know that the variable parameters are assigned to the placeholders in the order that they’re given.
      * The second parameter for getForObject() is the type that the response should be bound to. In this case, the response data (likely JSON format) should be deserealized into an Ingredient object that will be returned.
    - You can also use Map and URI parameter to do the above task. You can refer to that in the book.
    - getForEntity
      * works in much the same way as getForObject(), but instead of returning domain object that represents the response’s payload, it returns a ResponseEntity object that wraps the domain object.
      * ResponseEntity gives access to additional response details, such as the response headers.
      * Suppose, you want to get the ingredient data as well as the Date header from the response. You can do this with ResponseEntity like this:

public Ingredient getIngredientById(String ingredientId){

ResponseEntity<Ingredient> responseEntity =

rest.getForEntity([http://localhost:8080/ingredients/{id}](http://localhost:8080/ingredients/%7bid%7d),

Ingredient.class,ingredientId);

log.info(“Fetched time:”+responseEntity.getHeaders().getDate());

return responseEntity.getBody();

}

* + 7.1.2 PUTTING RESOURCES
    - Suppose that you want to replace an Ingredient resource with the data from a new Ingredient object.

public void updateIngredient(Ingredient ingredient){

rest.put([http://localhost:8080/ingredients/{id}](http://localhost:8080/ingredients/%7bid%7d),

ingredient,

ingredient.getId();

}

* + - * {id} is substituded by given Ingredient object’s id property.
      * The data to be sent is the Ingredient object itself.
  + 7.1.3 DELETING RESOURCES
    - Suppose taco cloud wants to remove an ingredient as an option.

public void deleteIngredient(Ingredient ingredient){

rest.delete([http://localhost:8080/ingredients/{id}](http://localhost:8080/ingredients/%7bid%7d),

ingredient.getId());

}

* + - * {id} is replaced by ingredient.getId().
  + 7.1.4 POSTING RESOURCE DATA
    - Let’s say you want to add a new ingredient in the Taco Cloud menu.
    - An http post request to …/ingredients endpoint with the ingredient data in the request body will make that happen.
    - If you want to receive the newly created Ingredient resource after the POST request, you’d use postForObject() like this:

public Ingredient createIngredient(Ingredient ingredient){

return rest.postForObject(<http://localhost:8080/ingredients>,

ingredient,

Ingredient.class);

}

* + - * The method takes a String url specification, the object to be poasted to the server, and the domain type that the reponse body should be bound to.
    - If you need the location of the newly created resource, then you can call postForLocation() instead:

Public URI createIngrediet(Ingredient ingredient){

return rest.postForLocation(<http://localhost:8080/ingredients>,

ingredient);

}

* + - * Returns URI of newly created resource instead of the resource object itself.
      * The URI returned is derived from the response’s Location header.
    - If you need both the payload and location, you can use postForEntity()

Public Ingredient createIngredient(Ingredient ingredient){

ResponseEntity<Ingredient> responseEntity =

Rest.postForEntity(<http://localhost:8080/ingredients>,

ingredient,

Ingredient.class);

log.info(“New resource created at “+ responseEntity.getHeaders().getLocation());

return responseEntity.getBody();

}

* + - If the API you’re consuming included hyperlinks in its reponse, RestTemplate isn’t as helpful. So, we use Traverson.
* 7.2 NAVIGATION REST APIs WITH TRAVERSON
  + Solution for consuming hypermedia APIs in spring application
  + You’ll consume an API by traversing the API on relation names
  + First, you need to instantiate a Traverson object with an API’s base URI:

Traverson traverson = new Traverson(

URI.create(<http://localhost:8080/api>),MediaTypes.HAL\_JSON);

* + - Pointed Traverson to the Taco Cloud’s base URL.
    - Also specify that the API will produce JSON responses with HAL-style hyperlinks so that Traverson knows how to parse the incoming resource data.
    - You can instantiate Traverson object or declare a bean and inject it.
  + Suppose you want to get a list of all ingredients. You know from previous chanpter that the ingredients link has an href property that links to the ingredient resource. You need to follow that link:

ParameterizedTypeReference<Resources<Ingredient>> ingredientType =

New ParameterizedTypeReference<Resources<Ingredient>(){ };

Resources<Ingredient> ingredientRes =

traverson

.follow(“ingredients”)

.toObject(“ingredientType);

Collection<Ingredient> ingredients = ingredientRes.getContent();

* + - Calling the follow() method on the Traverson object allows you to navigate to the resource whose link’s relation name is ingredients.
    - Then injest the contents of that resource by calling toObject().
    - The toObject() method required that you tell it what kind of object to read the data into.
    - You need to read it in as a Resources<Ingredient> object. ParameterizedTypeReference helps with that
  + Let’s you want to get recently created tacos:

ParameterizedTypeReference<Resources<Taco>> tacoType =

New ParameterizedTypeReference<Resources<Tacoo>>(){ };

Resources<Taco> tacoRes =

traverson

.follow(“tacos”,”recents)

.toObject(tacoType);

* + - Here, you follow the tacos link and then follow the recents link. Then get the resource of type tacoType.
  + Traverson makes it easy to navigate HATEOAS enabled API but doesn’t provide helps with writing and deleting from the APIs.
  + So, you can use both Traverson and RestTemplate.
  + For example, you want to add new Ingredient to the Taco cloud menu.

private Ingredient addIngredient(Ingredient ingredient){

String ingredientsUrl = traverson

.follow(“ingredients”)

.asLink()

.getHref();

return rest.postForObject(ingredientsUrl,

ingredient,

Ingredient.class);

}

* + - After following the ingredients link you ask for the link itself by calling asLink(). From that link, you ask for the link’s URL by calling getHref(). Then you call postForObject() on the RestTemplate instance and save the new ingredient.