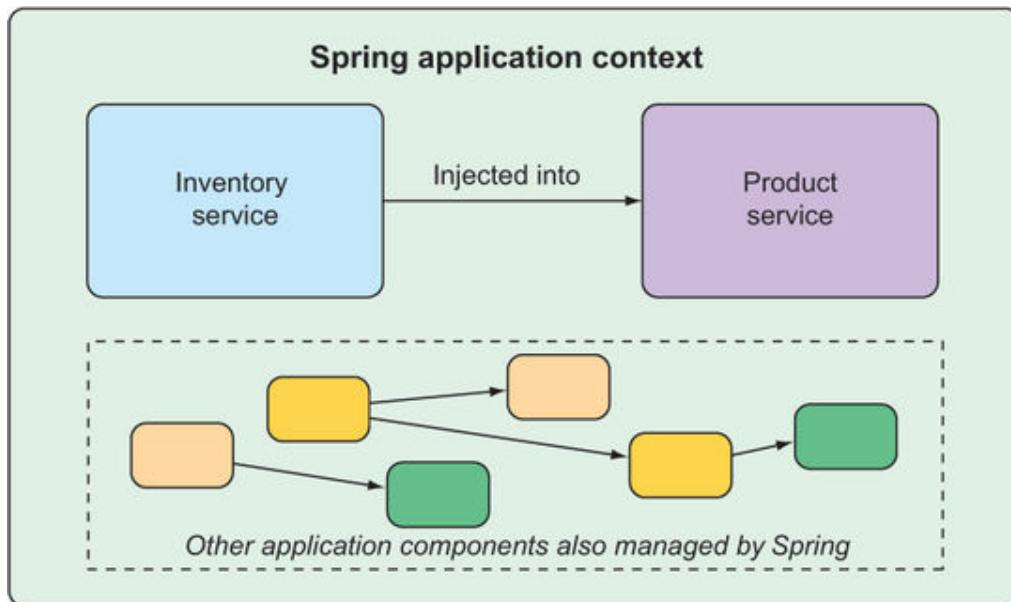


CODE LISTINGS AND FIGURES

CHAPTER 1

Figure 1.1 Application components are managed and injected into each other by the Spring application context.



```
1 <bean id="inventoryService"
2     class="com.example.InventoryService" />
3
4 <bean id="productService"
5     class="com.example.ProductService" />
6     <constructor-arg ref="inventoryService" />
7 </bean>
```

```
1  @Configuration
2  public class ServiceConfiguration {
3      @Bean
4      public InventoryService inventoryService() {
5          return new InventoryService();
6      }
7
8      @Bean
9      public ProductService productService() {
10         return new ProductService(inventoryService());
11     }
12 }
```

Figure 1.2 Starting a new project with the Initializr in Spring Tool Suite

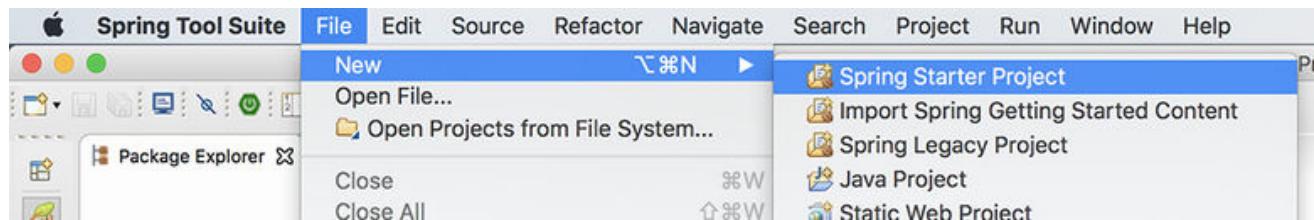


Figure 1.3 Specifying general project information for the Taco Cloud application

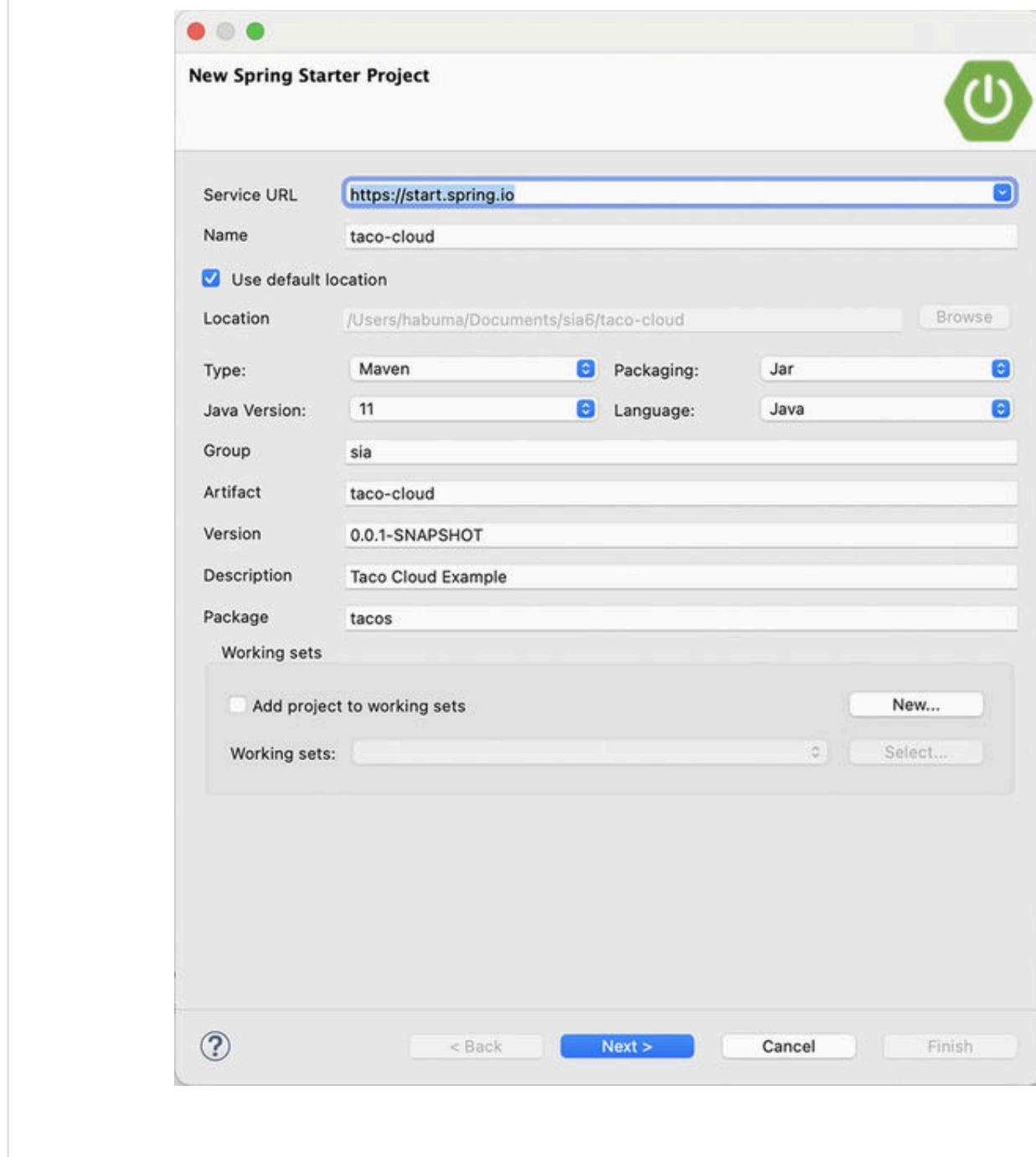


Figure 1.4 Choosing starter dependencies

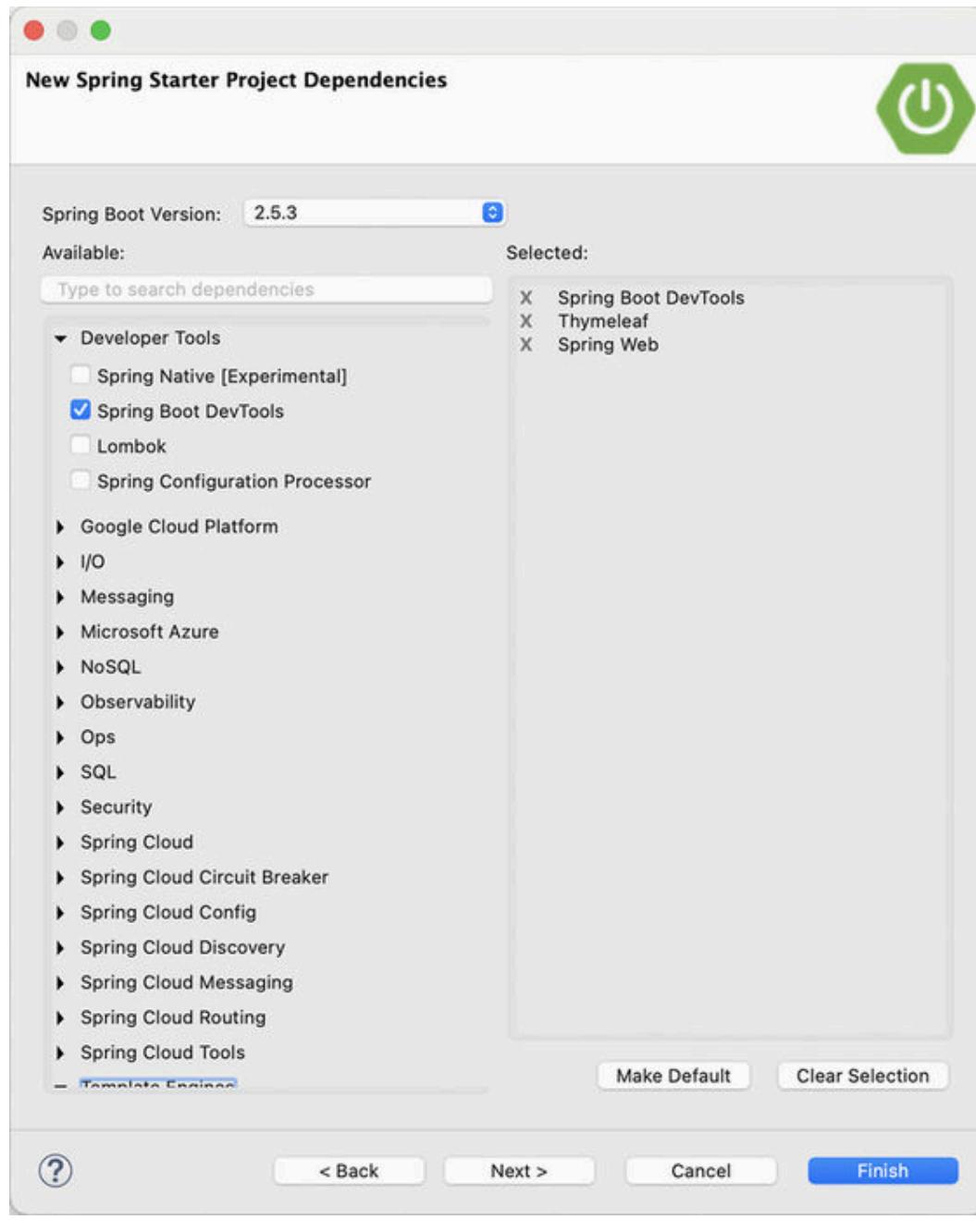


Figure 1.5 Optionally specifying an alternate Initializr address

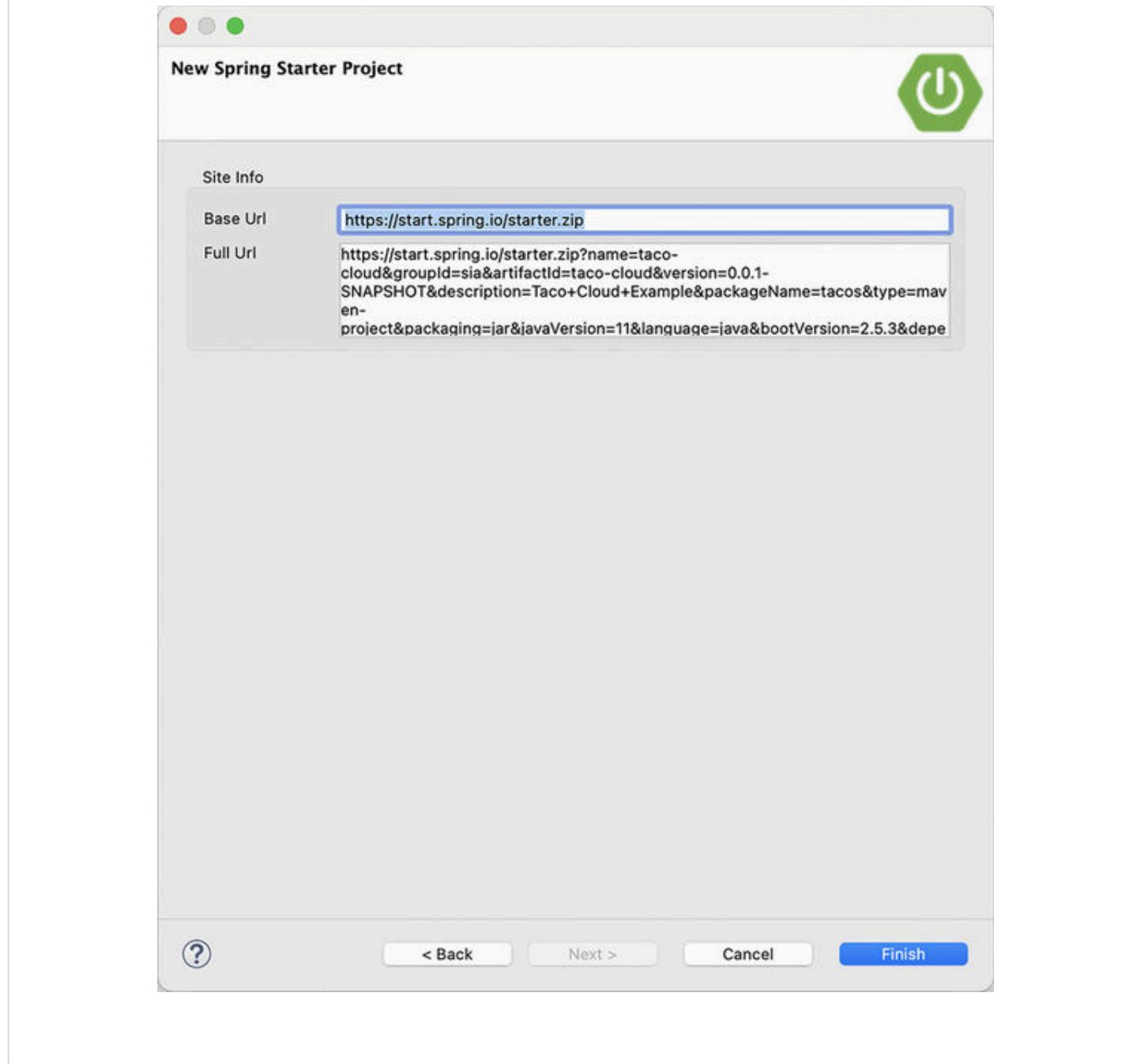
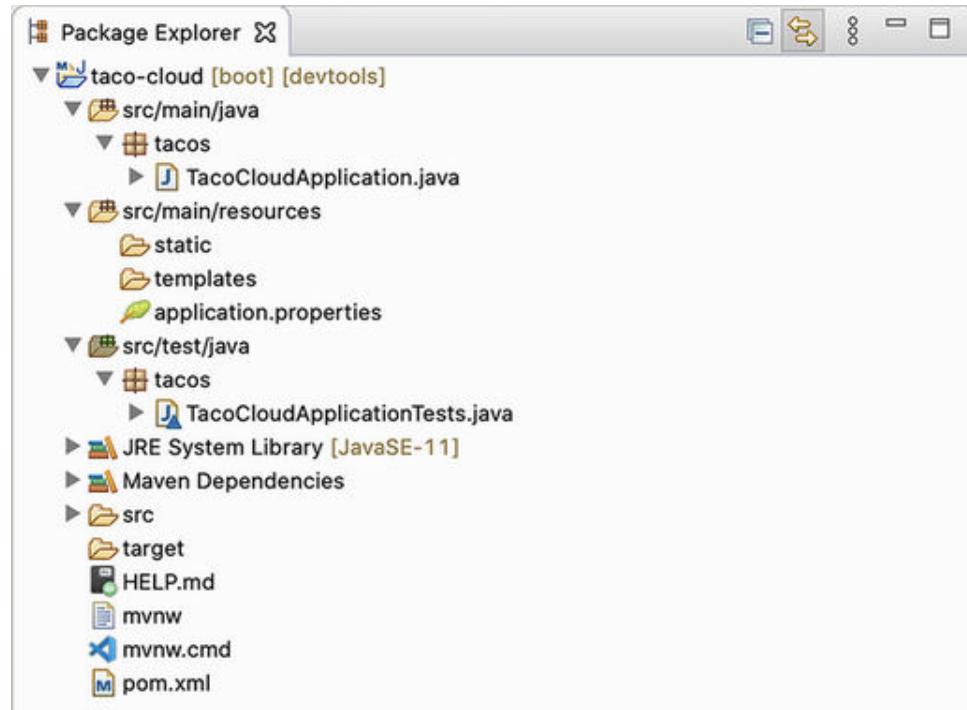


Figure 1.6 The initial Spring project structure as shown in Spring Tool Suite



Listing 1.1 The initial Maven build specification

```
1  <?xml version="1.0" encoding="UTF-8"?><project
2  xmlns="http://maven.apache.org/POM/4.0.0"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5      https://maven.apache.org/xsd/maven-4.0.0.xsd">
6  <modelVersion>4.0.0</modelVersion>
7  <parent>
8      <groupId>org.springframework.boot</groupId>
9      <artifactId>spring-boot-starter-parent</artifactId>
10     <version>2.5.3</version>
11     <relativePath />
12 </parent>
13 <groupId>sia</groupId>
14 <artifactId>taco-cloud</artifactId>
15 <version>0.0.1-SNAPSHOT</version>
16 <name>taco-cloud</name>
17 <description>Taco Cloud Example</description>
18
19 <properties>
20     <java.version>11</java.version>
21 </properties>
22
23 <dependencies>
24     <dependency>
25         <groupId>org.springframework.boot</groupId>
26         <artifactId>spring-boot-starter-thymeleaf</artifactId>
27     </dependency>
28
29     <dependency>
30         <groupId>org.springframework.boot</groupId>
31         <artifactId>spring-boot-starter-web</artifactId>
32     </dependency>
33
34     <dependency>
35         <groupId>org.springframework.boot</groupId>
36         <artifactId>spring-boot-devtools</artifactId>
37         <scope>runtime</scope>
38         <optional>true</optional>
39     </dependency>
40
41     <dependency>
42         <groupId>org.springframework.boot</groupId>
43         <artifactId>spring-boot-starter-test</artifactId>
44         <scope>test</scope>
45         <exclusions>
46             <exclusion>
47                 <groupId>org.junit.vintage</groupId>
48                 <artifactId>junit-vintage-engine</artifactId>
49             </exclusion>
50         </exclusions>
51     </dependency>
52
53 </dependencies>
54
55 <build>
56     <plugins>
57         <plugin>
58             <groupId>org.springframework.boot</groupId>
59             <artifactId>spring-boot-maven-plugin</artifactId>
60         </plugin>
61     </plugins>
62 </build>
63
```

1

2

3

```
64 <repositories>
65   <repository>
66     <id>spring-milestones</id>
67     <name>Spring Milestones</name>
68     <url>https://repo.spring.io/milestone</url>
69   </repository>
70 </repositories>
71 <pluginRepositories>
72   <pluginRepository>
73     <id>spring-milestones</id>
74     <name>Spring Milestones</name>
75     <url>https://repo.spring.io/milestone</url>
76   </pluginRepository>
77 </pluginRepositories>
78
79 </project>
```

Listing 1.2 The Taco Cloud bootstrap class

```
1 package tacos;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class TacoCloudApplication {
8
9   public static void main(String[] args) {
10     SpringApplication.run(TacoCloudApplication.class, args);
11   }
12
13 }
```

```
1 $ ./mvnw package
2 ...
3 $ java -jar target/taco-cloud-0.0.1-SNAPSHOT.jar
```

```
1 $ ./mvnw spring-boot:run
```

Listing 1.3 A baseline application test

```
1 package tacos;  
2  
3 import org.junit.jupiter.api.Test;  
4 import org.springframework.boot.test.context.SpringBootTest;  
5  
6 @SpringBootTest  
7 public class TacoCloudApplicationTests {  
8  
9     @Test  
10    public void contextLoads() {  
11    }  
12  
13 }
```

```
1 $ ./mvnw test
```

Listing 1.4 The home page controller

```
1 package tacos;  
2  
3 import org.springframework.stereotype.Controller;  
4 import org.springframework.web.bind.annotation.GetMapping;  
5  
6 @Controller  
7 public class HomeController {  
8  
9     @GetMapping("/")  
10    public String home() {  
11        return "home";  
12    }  
13  
14 }
```

Listing 1.5 The Taco Cloud home page template

```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3       xmlns:th="http://www.thymeleaf.org">
4   <head>
5     <title>Taco Cloud</title>
6   </head>
7
8   <body>
9     <h1>Welcome to...</h1>
10    
11  </body>
12 </html>
```

Listing 1.6 A test for the home page controller

```
1 package tacos;
2
3 import static org.hamcrest.Matchers.containsString;
4 import static org.springframework.test.web.servlet.request.MockMvcBuilders.ge
5 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.cont
6 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.stat
7 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view
8
9 import org.junit.jupiter.api.Test;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
12 import org.springframework.test.web.servlet.MockMvc;
13
14 @WebMvcTest(HomeController.class) 1
15 public class HomeControllerTest {
16
17   @Autowired
18   private MockMvc mockMvc; 2
19
20   @Test
21   public void testHomePage() throws Exception {
22     mockMvc.perform(get("/"))
23       .andExpect(status().isOk()) 3
24       .andExpect(view().name("home")) 4
25       .andExpect(content().string(
26         containsString("Welcome to..."))); 5
27   } 6
28
29 }
```

```
1 $ mvnw test
```

Figure 1.7 Highlights of the Spring Boot Dashboard

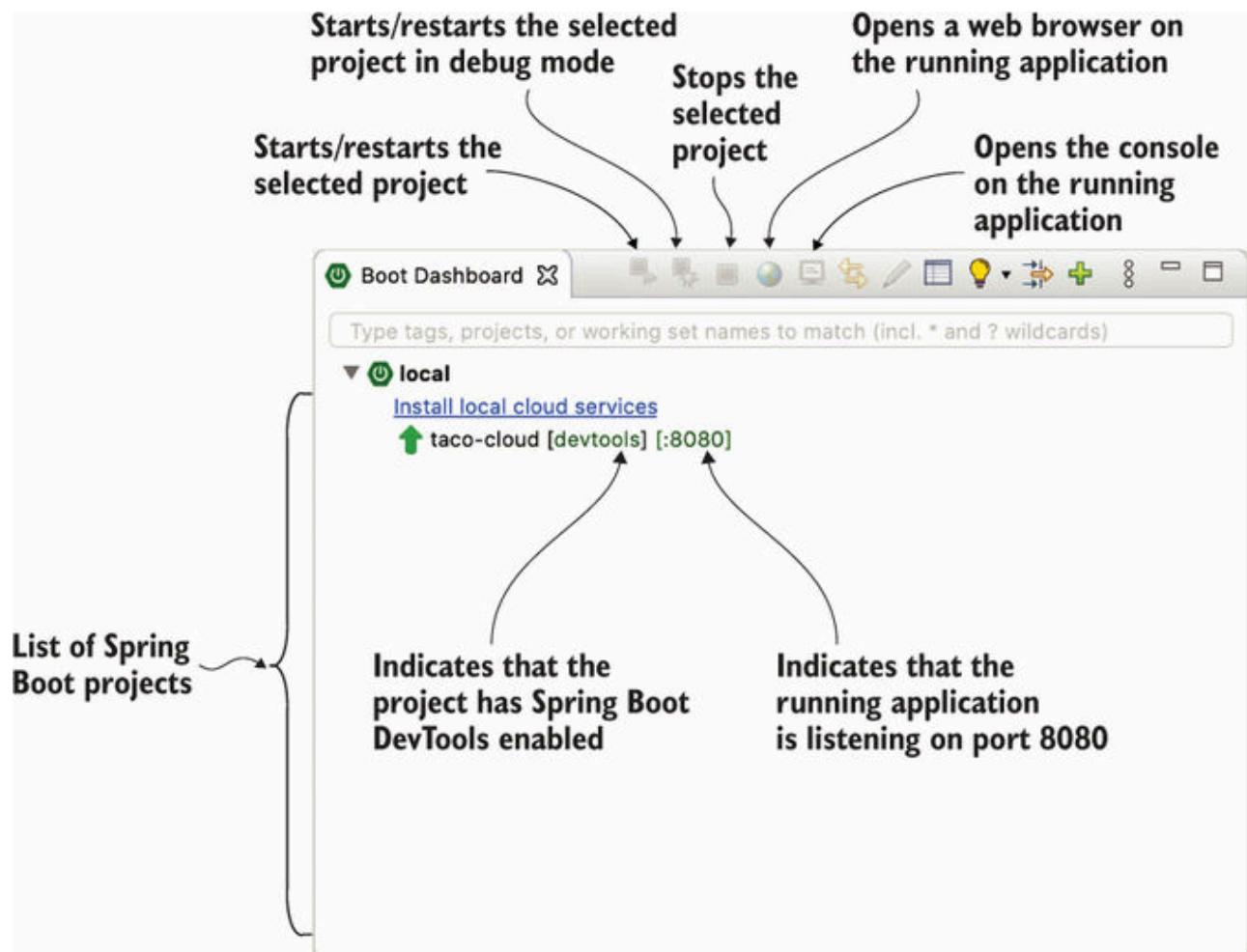
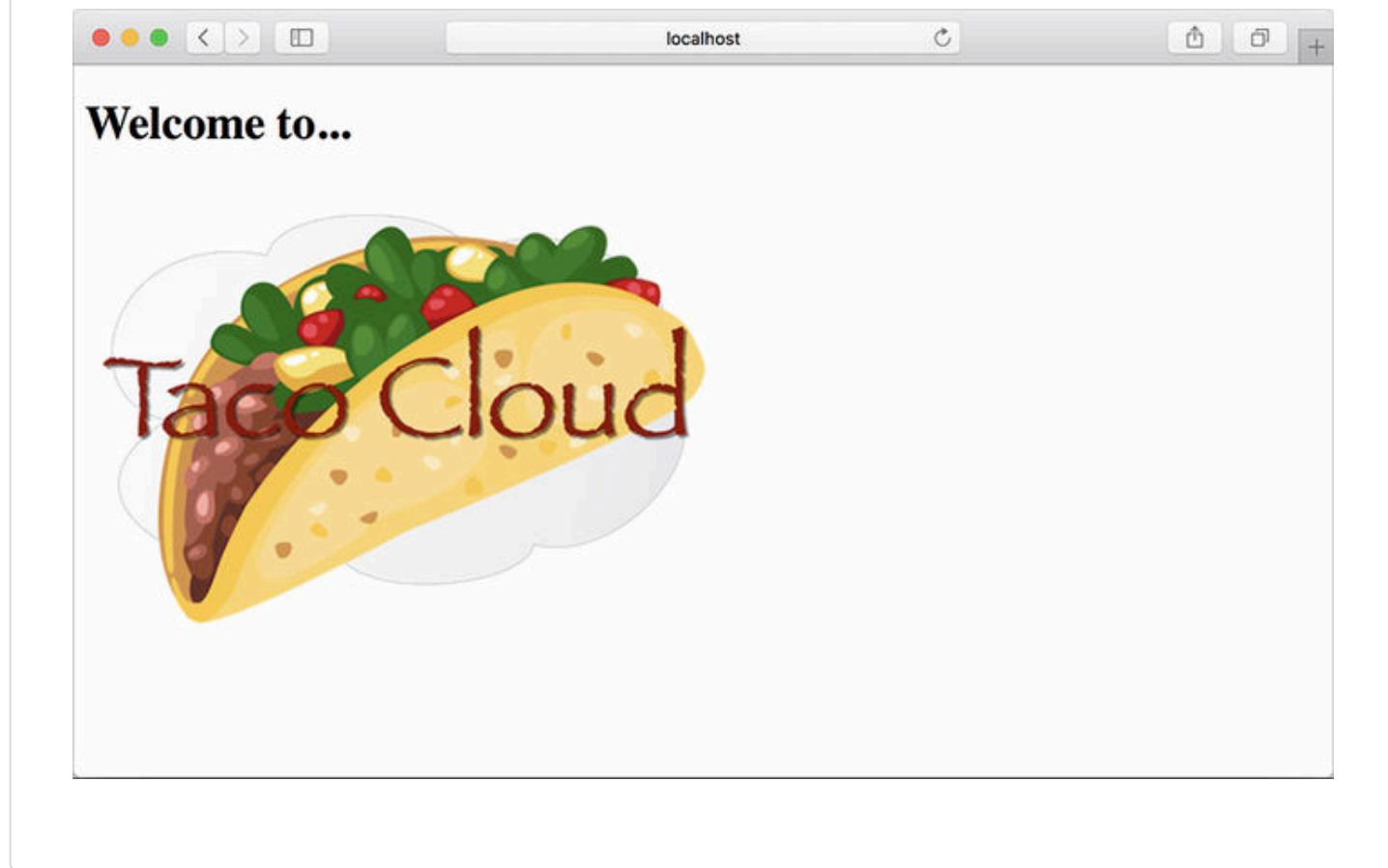


Figure 1.8 The Taco Cloud home page



CHAPTER 2

Figure 2.1 A typical Spring MVC request flow

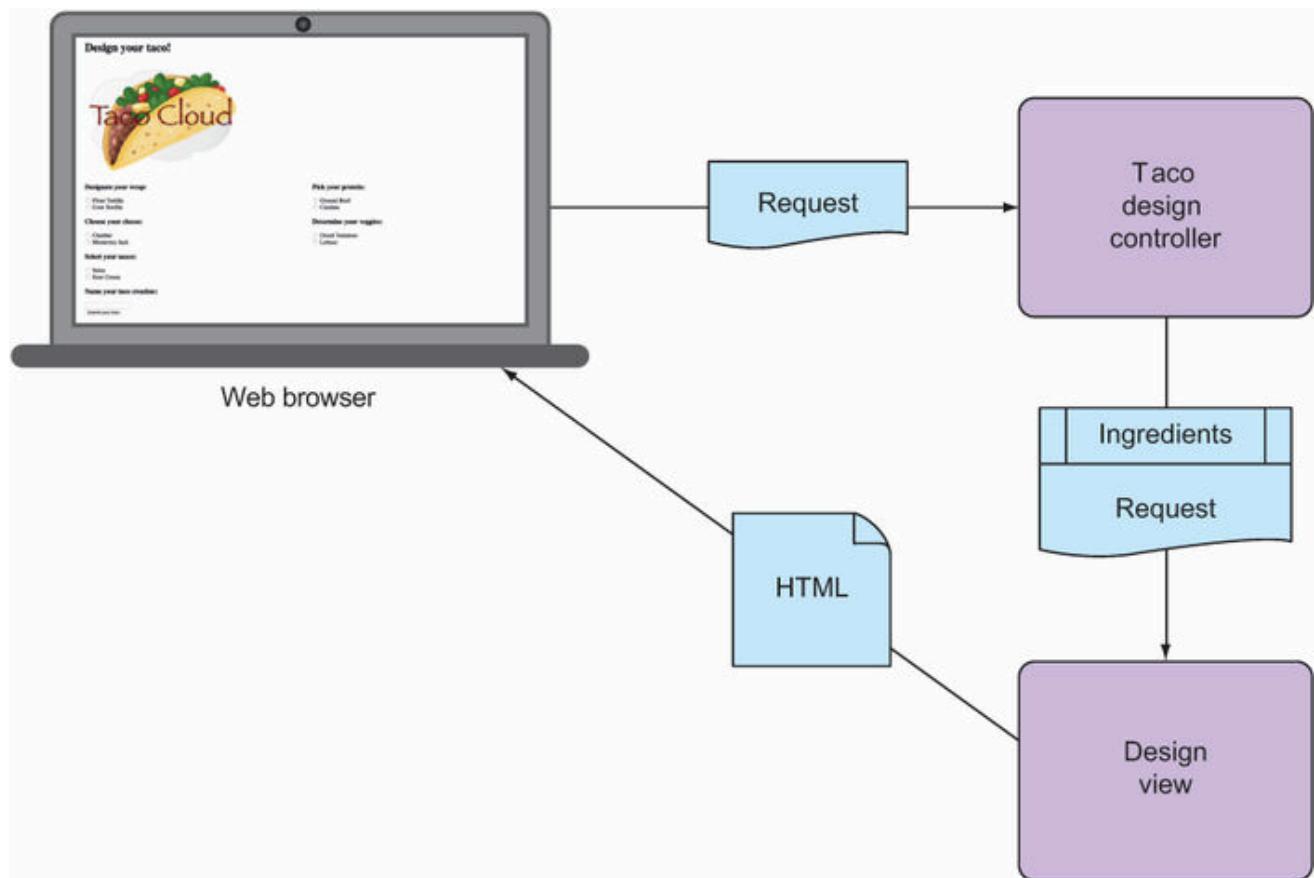
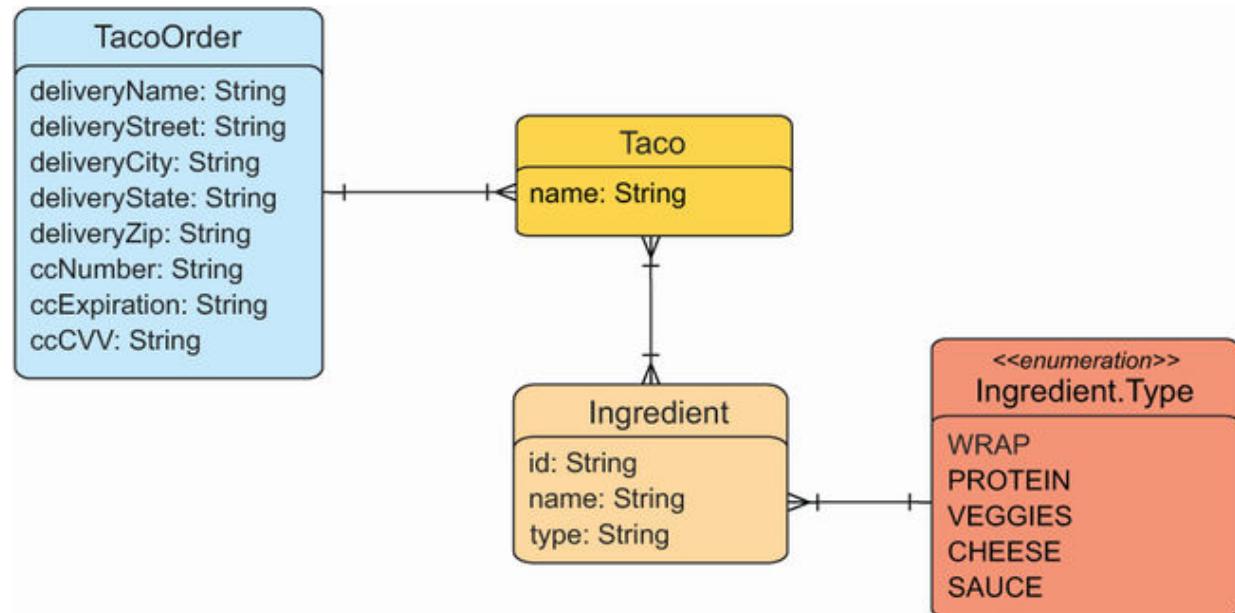


Figure 2.2 The Taco Cloud domain



Listing 2.1 Defining taco ingredients

```
package tacos;

import lombok.Data;

@Data
public class Ingredient {

    private final String id;
    private final String name;
    private final Type type;

    public enum Type {
        WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
    }
}
```

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
          </exclude>
        </excludes>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Listing 2.2 A domain object defining a taco design

```
package tacos;
import java.util.List;
import lombok.Data;

@Data
public class Taco {

  private String name;

  private List<Ingredient> ingredients;

}
```

Listing 2.3 A domain object for taco orders

```
package tacos;
import java.util.List;
import java.util.ArrayList;
import lombok.Data;

@Data
public class TacoOrder {

    private String deliveryName;
    private String deliveryStreet;
    private String deliveryCity;
    private String deliveryState;
    private String deliveryZip;
    private String ccNumber;
    private String ccExpiration;
    private String ccCVV;

    private List<Taco> tacos = new ArrayList<>();

    public void addTaco(Taco taco) {
        this.tacos.add(taco);
    }
}
```

Listing 2.4 The beginnings of a Spring controller class

```
package tacos.web;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.SessionAttributes;

import lombok.extern.slf4j.Slf4j;
import tacos.Ingredient;
import tacos.Ingredient.Type;
import tacos.Taco;

@Slf4j
@Controller
@RequestMapping("/design")
@SessionAttributes("tacoOrder")
public class DesignTacoController {

    @ModelAttribute
    public void addIngredientsToModel(Model model) {
        List<Ingredient> ingredients = Arrays.asList(
            new Ingredient("FLTO", "Flour Tortilla", Type.WRAP),
            new Ingredient("COTO", "Corn Tortilla", Type.WRAP),
            new Ingredient("GRBF", "Ground Beef", Type.PROTEIN),
            new Ingredient("CARN", "Carnitas", Type.PROTEIN),
            new Ingredient("TMTO", "Diced Tomatoes", Type.VEGGIES),
            new Ingredient("LETC", "Lettuce", Type.VEGGIES),
            new Ingredient("CHED", "Cheddar", Type.CHEESE),
            new Ingredient("JACK", "Monterrey Jack", Type.CHEESE),
            new Ingredient("SLSA", "Salsa", Type.SAUCE),
            new Ingredient("SRCR", "Sour Cream", Type.SAUCE)
        );
        Type[] types = Ingredient.Type.values();
        for (Type type : types) {
            model.addAttribute(type.toString().toLowerCase(),
                filterByType(ingredients, type));
        }
    }

    @ModelAttribute(name = "tacoOrder")
    public TacoOrder order() {
        return new TacoOrder();
    }

    @ModelAttribute(name = "taco")
    public Taco taco() {
        return new Taco();
    }

    @GetMapping
    public String showDesignForm() {
        return "design";
    }

    private Iterable<Ingredient> filterByType(
        List<Ingredient> ingredients, Type type) {
```

```
        return ingredients
            .stream()
            .filter(x -> x.getType().equals(type))
            .collect(Collectors.toList());
    }

}
```

```
private static final org.slf4j.Logger log =
org.slf4j.LoggerFactory.getLogger(DesignTacoController.class);
```

Table 2.1 Spring MVC request-mapping annotations (view table figure)

Annotation	Description
@RequestMapping	General-purpose request handling
@GetMapping	Handles HTTP GET requests
@PostMapping	Handles HTTP POST requests
@PutMapping	Handles HTTP PUT requests
@DeleteMapping	Handles HTTP DELETE requests
@PatchMapping	Handles HTTP PATCH requests

```
<p th:text="${message}">placeholder message</p>
```

```
<h3>Designate your wrap:</h3>
<div th:each="ingredient : ${wrap}">
    <input th:field="*{ingredients}" type="checkbox"
        th:value="${ingredient.id}"/>
    <span th:text="${ingredient.name}">INGREDIENT</span><br/>
</div>
```

```
<div>
    <input name="ingredients" type="checkbox" value="FLTO" />
    <span>Flour Tortilla</span><br/>
</div>
```

Listing 2.5 The complete design-a-taco page

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Taco Cloud</title>
    <link rel="stylesheet" th:href="@{/styles.css}" />
</head>

<body>
    <h1>Design your taco!</h1>
    

    <form method="POST" th:object="${taco}">
        <div class="grid">
            <div class="ingredient-group" id="wraps">
                <h3>Designate your wrap:</h3>
                <div th:each="ingredient : ${wrap}">
                    <input th:field="*{ingredients}" type="checkbox"
                           th:value="${ingredient.id}"/>
                    <span th:text="${ingredient.name}">INGREDIENT</span><br/>
                </div>
            </div>

            <div class="ingredient-group" id="proteins">
                <h3>Pick your protein:</h3>
                <div th:each="ingredient : ${protein}">
                    <input th:field="*{ingredients}" type="checkbox"
                           th:value="${ingredient.id}"/>
                    <span th:text="${ingredient.name}">INGREDIENT</span><br/>
                </div>
            </div>

            <div class="ingredient-group" id="cheeses">
                <h3>Choose your cheese:</h3>
                <div th:each="ingredient : ${cheese}">
                    <input th:field="*{ingredients}" type="checkbox"
                           th:value="${ingredient.id}"/>
                    <span th:text="${ingredient.name}">INGREDIENT</span><br/>
                </div>
            </div>

            <div class="ingredient-group" id="veggies">
                <h3>Determine your veggies:</h3>
                <div th:each="ingredient : ${veggies}">
                    <input th:field="*{ingredients}" type="checkbox"
                           th:value="${ingredient.id}"/>
                    <span th:text="${ingredient.name}">INGREDIENT</span><br/>
                </div>
            </div>

            <div class="ingredient-group" id="sauces">
                <h3>Select your sauce:</h3>
                <div th:each="ingredient : ${sauce}">
                    <input th:field="*{ingredients}" type="checkbox"
                           th:value="${ingredient.id}"/>
                    <span th:text="${ingredient.name}">INGREDIENT</span><br/>
                </div>
            </div>
        </div>
    </form>
```

```
<h3>Name your taco creation:</h3>
<input type="text" th:field="*{name}" />
<br/>

<button>Submit Your Taco</button>
</div>
</form>
</body>
</html>
```

Figure 2.3 The rendered taco design page

localhost

Design your taco!

Designate your wrap:

Flour Tortilla
 Corn Tortilla

Choose your cheese:

Cheddar
 Monterrey Jack

Select your sauce:

Salsa
 Sour Cream

Name your taco creation:

Pick your protein:

Ground Beef
 Carnitas

Determine your veggies:

Diced Tomatoes
 Lettuce

Submit your taco

Listing 2.6 Handling **POST** requests with **@PostMapping**

```
@PostMapping  
public String processTaco(Taco taco,  
                           @ModelAttribute TacoOrder tacoOrder) {  
    tacoOrder.addTaco(taco);  
    log.info("Processing taco: {}", taco);  
  
    return "redirect:/orders/current";  
}
```

Listing 2.7 Converting strings to ingredients

```
package tacos.web;

import java.util.HashMap;
import java.util.Map;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

import tacos.Ingredient;
import tacos.Ingredient.Type;

@Component
public class IngredientByIdConverter implements Converter<String, Ingredient> {

    private Map<String, Ingredient> ingredientMap = new HashMap<>();

    public IngredientByIdConverter() {
        ingredientMap.put("FLTO",
            new Ingredient("FLTO", "Flour Tortilla", Type.WRAP));
        ingredientMap.put("COTO",
            new Ingredient("COTO", "Corn Tortilla", Type.WRAP));
        ingredientMap.put("GRBF",
            new Ingredient("GRBF", "Ground Beef", Type.PROTEIN));
        ingredientMap.put("CARN",
            new Ingredient("CARN", "Carnitas", Type.PROTEIN));
        ingredientMap.put("TMTO",
            new Ingredient("TMTO", "Diced Tomatoes", Type.VEGGIES));
        ingredientMap.put("LETC",
            new Ingredient("LETC", "Lettuce", Type.VEGGIES));
        ingredientMap.put("CHED",
            new Ingredient("CHED", "Cheddar", Type.CHEESE));
        ingredientMap.put("JACK",
            new Ingredient("JACK", "Monterrey Jack", Type.CHEESE));
        ingredientMap.put("SLSA",
            new Ingredient("SLSA", "Salsa", Type.SAUCE));
        ingredientMap.put("SRCR",
            new Ingredient("SRCR", "Sour Cream", Type.SAUCE));
    }

    @Override
    public Ingredient convert(String id) {
        return ingredientMap.get(id);
    }
}
```

Listing 2.8 A controller to present a taco order form

```
package tacos.web;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.bind.support.SessionStatus;

import lombok.extern.slf4j.Slf4j;
import tacos.TacoOrder;

@Slf4j
@Controller
@RequestMapping("/orders")
@SessionAttributes("tacoOrder")
public class OrderController {

    @GetMapping("/current")
    public String orderForm() {
        return "orderForm";
    }

}
```

Listing 2.9 A taco order form view

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Taco Cloud</title>
    <link rel="stylesheet" th:href="@{/styles.css}" />
</head>

<body>

    <form method="POST" th:action="@{/orders}" th:object="${tacoOrder}">
        <h1>Order your taco creations!</h1>

        <h3>Your tacos in this order:</h3>
        <a th:href="@{/design}" id="another">Design another taco</a><br/>
        <ul>
            <li th:each="taco : ${tacoOrder.tacos}">
                <span th:text="${taco.name}">taco name</span></li>
        </ul>

        <h3>Deliver my taco masterpieces to...</h3>
        <label for="deliveryName">Name: </label>
        <input type="text" th:field="*{deliveryName}" />
        <br/>

        <label for="deliveryStreet">Street address: </label>
        <input type="text" th:field="*{deliveryStreet}" />
        <br/>

        <label for="deliveryCity">City: </label>
        <input type="text" th:field="*{deliveryCity}" />
        <br/>

        <label for="deliveryState">State: </label>
        <input type="text" th:field="*{deliveryState}" />
        <br/>

        <label for="deliveryZip">Zip code: </label>
        <input type="text" th:field="*{deliveryZip}" />
        <br/>

        <h3>Here's how I'll pay...</h3>
        <label for="ccNumber">Credit Card #: </label>
        <input type="text" th:field="*{ccNumber}" />
        <br/>

        <label for="ccExpiration">Expiration: </label>
        <input type="text" th:field="*{ccExpiration}" />
        <br/>

        <label for="ccCVV">CVV: </label>
        <input type="text" th:field="*{ccCVV}" />
        <br/>

        <input type="submit" value="Submit Order"/>
    </form>
</body>
</html>
```

Listing 2.10 Handling a taco order submission

```
@PostMapping  
public String processOrder(TacoOrder order,  
    SessionStatus sessionStatus) {  
    log.info("Order submitted: {}", order);  
    sessionStatus.setComplete();  
  
    return "redirect:/";  
}
```

Figure 2.4 The taco order form

The screenshot shows a web browser window with the address bar set to "localhost". The main content area features a large, stylized illustration of a taco filled with meat, cheese, and toppings, with the words "Taco Cloud" written across it in a red, outlined font. Above the illustration, the text "Order your taco creations!" is displayed. Below the illustration, there is a link "Design another taco".

Deliver my taco masterpieces to...

Name:

Street address:

City:

State:

Zip code:

Here's how I'll pay...

Credit Card #:

Expiration:

CVV:

```
Order submitted: TacoOrder(deliveryName=Craig Walls, deliveryStreet=1234 7th Street, deliveryCity=Somewhere, deliveryState=Who knows?, deliveryZip=zipzap, ccNumber=Who can guess?, ccExpiration=Some day, ccCVV=See-vee-vee, tacos=[Taco(name=Awesome Sauce, ingredients=[Ingredient(id=FLTO, name=Flour Tortilla, type=WRAP), Ingredient(id=GRBF, name=Ground Beef, type=PROTEIN), Ingredient(id=CHED, name=Cheddar, type=CHEESE), Ingredient(id=TMTO, name=Diced Tomatoes, type=VEGGIES), Ingredient(id=SLSA, name=Salsa, type=SAUCE), Ingredient(id=SRCR, name=Sour Cream, type=SAUCE)]), Taco(name=Quesoriffic, ingredients=[Ingredient(id=FLTO, name=Flour Tortilla, type=WRAP), Ingredient(id=CHED, name=Cheddar, type=CHEESE), Ingredient(id=JACK, name=Monterrey Jack, type=CHEESE), Ingredient(id=TMTO, name=Diced Tomatoes, type=VEGGIES), Ingredient(id=SRCR, name=Sour Cream, type=SAUCE)]])])
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

```
implementation 'org.springframework.boot:spring-boot-starter-validation'
```

Listing 2.11 Adding validation to the `Taco` domain class

```
package tacos;
import java.util.List;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import lombok.Data;

@Data
public class Taco {

    @NotNull
    @Size(min=5, message="Name must be at least 5 characters long")
    private String name;
    @NotNull
    @Size(min=1, message="You must choose at least 1 ingredient")
    private List<Ingredient> ingredients;

}
```

Listing 2.12 Validating order fields

```
package tacos;
import javax.validation.constraints.Digits;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;
import org.hibernate.validator.constraints.CreditCardNumber;
import java.util.List;
import java.util.ArrayList;
import lombok.Data;

@Data
public class TacoOrder {

    @NotBlank(message="Delivery name is required")
    private String deliveryName;

    @NotBlank(message="Street is required")
    private String deliveryStreet;

    @NotBlank(message="City is required")
    private String deliveryCity;

    @NotBlank(message="State is required")
    private String deliveryState;

    @NotBlank(message="Zip code is required")
    private String deliveryZip;

    @CreditCardNumber(message="Not a valid credit card number")
    private String ccNumber;

    @Pattern(regexp="^(0[1-9]|1[0-2])([\\/])([2-9][0-9])$",
              message="Must be formatted MM/YY")
    private String ccExpiration;

    @Digits(integer=3, fraction=0, message="Invalid CVV")
    private String ccCVV;

    private List<Taco> tacos = new ArrayList<>();

    public void addTaco(Taco taco) {
        this.tacos.add(taco);
    }
}
```

Listing 2.13 Validating a POST ed Taco

```
import javax.validation.Valid;
import org.springframework.validation.Errors;

...
@PostMapping
public String processTaco(
    @Valid Taco taco, Errors errors,
    @ModelAttribute TacoOrder tacoOrder) {

    if (errors.hasErrors()) {
        return "design";
    }

    tacoOrder.addTaco(taco);
    log.info("Processing taco: {}", taco);

    return "redirect:/orders/current";
}
```

Listing 2.14 Validating a POST ed TacoOrder

```
@PostMapping
public String processOrder(@Valid TacoOrder order, Errors errors,
    SessionStatus sessionStatus) {
    if (errors.hasErrors()) {
        return "orderForm";
    }

    log.info("Order submitted: {}", order);
    sessionStatus.setComplete();

    return "redirect:/";
}
```

Listing 2.15 Displaying validation errors

```
<label for="ccNumber">Credit Card #: </label>
<input type="text" th:field="*{ccNumber}"/>
<span class="validationError"
    th:if="${#fields.hasErrors('ccNumber')}"
    th:errors="*{ccNumber}">CC Num Error</span>
```

Figure 2.5 Validation errors displayed on the order form

The screenshot shows a web browser window with the URL "localhost" in the address bar. The main content is a landing page for "Taco Cloud" featuring a large graphic of a taco filled with meat, cheese, and toppings, with the word "Taco Cloud" written across it. Below the graphic, the text "Order your taco creations!" is displayed.

Please correct the problems below and resubmit.

[Design another taco](#)

Deliver my taco masterpieces to...

Name: Name is required

Street address: 1234 7th Street

City: City is required

State: VT

Zip code: Zip code is required

Here's how I'll pay...

Credit Card #: Who can guess? Not a valid credit card number

Expiration: Some day Must be formatted MM/YY

CVV: See-vee-vee Invalid CVV

Listing 2.16 Declaring a view controller

```
package tacos.web;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/").setViewName("home");
    }

}
```

```
@SpringBootApplication
public class TacoCloudApplication implements WebMvcConfigurer {

    public static void main(String[] args) {
        SpringApplication.run(TacoCloudApplication.class, args);
    }
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/").setViewName("home");
    }

}
```

Table 2.2 Supported template options ([view table figure](#))

Template	Spring Boot starter dependency
FreeMarker	<code>spring-boot-starter-freemarker</code>
Groovy templates	<code>spring-boot-starter-groovy-templates</code>
JavaServer Pages (JSP)	None (provided by Tomcat or Jetty)
Mustache	<code>spring-boot-starter-mustache</code>
Thymeleaf	<code>spring-boot-starter-thymeleaf</code>

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-mustache</artifactId>
</dependency>
```

```
<h3>Designate your wrap:</h3>
{{#wrap}}
<div>
  <input name="ingredients" type="checkbox" value="{{id}}"/>
  <span>{{name}}</span><br/>
</div>
{{/wrap}}
```

Table 2.3 Properties to enable/disable template caching ([view table figure](#))

Template	Cache-enable property
FreeMarker	<code>spring.freemarker.cache</code>
Groovy templates	<code>spring.groovy.template.cache</code>
Mustache	<code>spring.mustache.cache</code>
Thymeleaf	<code>spring.thymeleaf.cache</code>

```
spring.thymeleaf.cache=false
```

CHAPTER 3

Listing 3.1 Querying a database without **JdbcTemplate**

```
@Override
public Optional<Ingredient> findById(String id) {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        connection = dataSource.getConnection();
        statement = connection.prepareStatement(
            "select id, name, type from Ingredient where id=?");
        statement.setString(1, id);
        resultSet = statement.executeQuery();
        Ingredient ingredient = null;
        if(resultSet.next()) {
            ingredient = new Ingredient(
                resultSet.getString("id"),
                resultSet.getString("name"),
                Ingredient.Type.valueOf(resultSet.getString("type")));
        }
        return Optional.of(ingredient);
    } catch (SQLException e) {
        // ??? What should be done here ???
    } finally {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {}
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {}
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {}
        }
    }
    return Optional.empty();
}
```

Listing 3.2 Querying a database with `JdbcTemplate`

```
private JdbcTemplate jdbcTemplate;

public Optional<Ingredient> findById(String id) {
    List<Ingredient> results = jdbcTemplate.query(
        "select id, name, type from Ingredient where id=?",
        this::mapRowToIngredient,
        id);
    return results.size() == 0 ?
        Optional.empty() :
        Optional.of(results.get(0));
}

private Ingredient mapRowToIngredient(ResultSet row, int rowNum)
    throws SQLException {
    return new Ingredient(
        row.getString("id"),
        row.getString("name"),
        Ingredient.Type.valueOf(row.getString("type")));
}
```

Listing 3.3 Adding ID and timestamp fields to the `Taco` class

```
@Data
public class Taco {

    private Long id;

    private Date createdAt = new Date();

    // ...
}
```

```
@Data
public class TacoOrder implements Serializable {

    private static final long serialVersionUID = 1L;

    private Long id;

    private Date placedAt;
    // ...
}
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

```
spring.datasource.generate-unique-name=false
spring.datasource.name=tacocloud
```

```
spring:
  datasource:
    generate-unique-name: false
    name: tacocloud
```

```
package tacos.data;

import java.util.Optional;

import tacos.Ingredient;

public interface IngredientRepository {

    Iterable<Ingredient> findAll();

    Optional<Ingredient> findById(String id);

    Ingredient save(Ingredient ingredient);

}
```

Listing 3.4 Beginning an ingredient repository with **JdbcTemplate**

```
package tacos.data;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import tacos.Ingredient;

@Repository
public class JdbcIngredientRepository implements IngredientRepository {

    private JdbcTemplate jdbcTemplate;

    public JdbcIngredientRepository(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    // ...
}
```

```
@Autowired
public JdbcIngredientRepository(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}
```

Listing 3.5 Querying the database with **JdbcTemplate**

```
@Override
public Iterable<Ingredient> findAll() {
    return jdbcTemplate.query(
        "select id, name, type from Ingredient",
        this::mapRowToIngredient);
}

@Override
public Optional<Ingredient> findById(String id) {
    List<Ingredient> results = jdbcTemplate.query(
        "select id, name, type from Ingredient where id=?",
        this::mapRowToIngredient,
        id);
    return results.size() == 0 ?
        Optional.empty() :
        Optional.of(results.get(0));
}

private Ingredient mapRowToIngredient(ResultSet row, int rowNum)
    throws SQLException {
    return new Ingredient(
        row.getString("id"),
        row.getString("name"),
        Ingredient.Type.valueOf(row.getString("type")));
}
```

```
@Override
public Ingredient findById(String id) {
    return jdbcTemplate.queryForObject(
        "select id, name, type from Ingredient where id=?",
        new RowMapper<Ingredient>() {
            public Ingredient mapRow(ResultSet rs, int rowNum)
                throws SQLException {
                return new Ingredient(
                    rs.getString("id"),
                    rs.getString("name"),
                    Ingredient.Type.valueOf(rs.getString("type")));
            }
        }, id);
}
```

Listing 3.6 Inserting data with `JdbcTemplate`

```
@Override
public Ingredient save(Ingredient ingredient) {
    jdbcTemplate.update(
        "insert into Ingredient (id, name, type) values (?, ?, ?)",
        ingredient.getId(),
        ingredient.getName(),
        ingredient.getType().toString());
    return ingredient;
}
```

Listing 3.7 Injecting and using a repository in the controller

```
@Controller
@RequestMapping("/design")
@SessionAttributes("tacoOrder")
public class DesignTacoController {

    private final IngredientRepository ingredientRepo;

    @Autowired
    public DesignTacoController(
        IngredientRepository ingredientRepo) {
        this.ingredientRepo = ingredientRepo;
    }

    @ModelAttribute
    public void addIngredientsToModel(Model model) {
        Iterable<Ingredient> ingredients = ingredientRepo.findAll();
        Type[] types = Ingredient.Type.values();
        for (Type type : types) {
            model.addAttribute(type.toString().toLowerCase(),
                filterByType(ingredients, type));
        }
    }

    // ...
}
```

Listing 3.8 Simplifying IngredientByIdConverter

```
package tacos.web;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

import tacos.Ingredient;
import tacos.data.IngredientRepository;

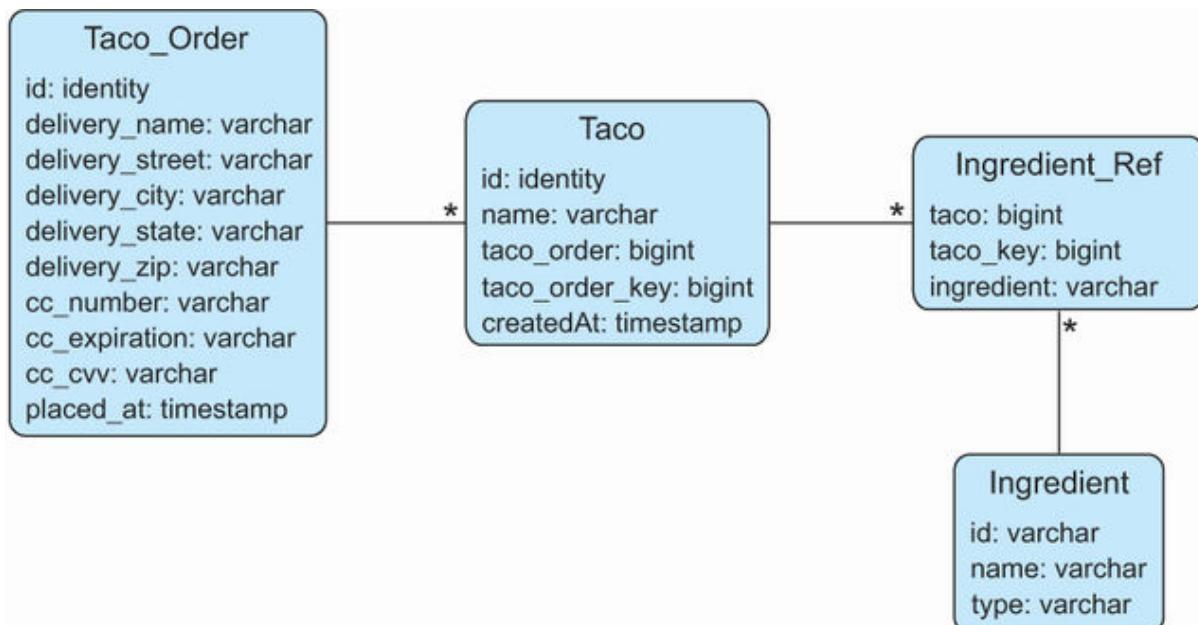
@Component
public class IngredientByIdConverter implements Converter<String, Ingredient> {

    private IngredientRepository ingredientRepo;

    @Autowired
    public IngredientByIdConverter(IngredientRepository ingredientRepo) {
        this.ingredientRepo = ingredientRepo;
    }

    @Override
    public Ingredient convert(String id) {
        return ingredientRepo.findById(id).orElse(null);
    }
}
```

Figure 3.1 The tables for the Taco Cloud schema



Listing 3.9 Defining the Taco Cloud schema

```
create table if not exists Taco_Order (
    id identity,
    delivery_Name varchar(50) not null,
    delivery_Street varchar(50) not null,
    delivery_City varchar(50) not null,
    delivery_State varchar(2) not null,
    delivery_Zip varchar(10) not null,
    cc_number varchar(16) not null,
    cc_expiration varchar(5) not null,
    cc_cvv varchar(3) not null,
    placed_at timestamp not null
);

create table if not exists Taco (
    id identity,
    name varchar(50) not null,
    taco_order bigint not null,
    taco_order_key bigint not null,
    created_at timestamp not null
);

create table if not exists Ingredient_Ref (
    ingredient varchar(4) not null,
    taco bigint not null,
    taco_key bigint not null
);

create table if not exists Ingredient (
    id varchar(4) not null,
    name varchar(25) not null,
    type varchar(10) not null
);

alter table Taco
    add foreign key (taco_order) references Taco_Order(id);
alter table Ingredient_Ref
    add foreign key (ingredient) references Ingredient(id);
```

Listing 3.10 Preloading the database with data.sql

```
delete from Ingredient_Ref;
delete from Taco;
delete from Taco_Order;

delete from Ingredient;
insert into Ingredient (id, name, type)
    values ('FLTO', 'Flour Tortilla', 'WRAP');
insert into Ingredient (id, name, type)
    values ('COTO', 'Corn Tortilla', 'WRAP');
insert into Ingredient (id, name, type)
    values ('GRBF', 'Ground Beef', 'PROTEIN');
insert into Ingredient (id, name, type)
    values ('CARN', 'Carnitas', 'PROTEIN');
insert into Ingredient (id, name, type)
    values ('TMTO', 'Diced Tomatoes', 'VEGGIES');
insert into Ingredient (id, name, type)
    values ('LETC', 'Lettuce', 'VEGGIES');
insert into Ingredient (id, name, type)
    values ('CHED', 'Cheddar', 'CHEESE');
insert into Ingredient (id, name, type)
    values ('JACK', 'Monterrey Jack', 'CHEESE');
insert into Ingredient (id, name, type)
    values ('SLSA', 'Salsa', 'SAUCE');
insert into Ingredient (id, name, type)
    values ('SRCR', 'Sour Cream', 'SAUCE');
```

```
package tacos.data;

import java.util.Optional;

import tacos.TacoOrder;

public interface OrderRepository {

    TacoOrder save(TacoOrder order);

}
```

```
package tacos;

import lombok.Data;

@Data
public class IngredientRef {

    private final String ingredient;

}
```

```
package tacos.data;

import java.sql.Types;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import java.util.Optional;

import org.springframework.asm.Type;
import org.springframework.jdbc.core.JdbcOperations;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.PreparedStatementCreatorFactory;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import tacos.IngredientRef;
import tacos.Taco;
import tacos.TacoOrder;

@Repository
public class JdbcOrderRepository implements OrderRepository {

    private JdbcOperations jdbcOperations;

    public JdbcOrderRepository(JdbcOperations jdbcOperations) {
        this.jdbcOperations = jdbcOperations;
    }

    @Override
    @Transactional
    public TacoOrder save(TacoOrder order) {
        PreparedStatementCreatorFactory pscf =
            new PreparedStatementCreatorFactory(
                "insert into Taco_Order "
                + "(delivery_name, delivery_street, delivery_city, "
                + "delivery_state, delivery_zip, cc_number, "
                + "cc_expiration, cc_cvv, placed_at) "
                + "values (?, ?, ?, ?, ?, ?, ?, ?, ?)",
                Types.VARCHAR, Types.VARCHAR, Types.VARCHAR,
                Types.VARCHAR, Types.VARCHAR, Types.VARCHAR,
                Types.VARCHAR, Types.VARCHAR, Types.TIMESTAMP
            );
        pscf.setReturnGeneratedKeys(true);

        order.setPlacedAt(new Date());
        PreparedStatementCreator psc =
            pscf.newPreparedStatementCreator(
                Arrays.asList(
                    order.getDeliveryName(),
                    order.getDeliveryStreet(),
                    order.getDeliveryCity(),
                    order.getDeliveryState(),
                    order.getDeliveryZip(),
                    order.getCcNumber(),
                    order.getCcExpiration(),
                    order.getCcCVV(),
                    order.getPlacedAt())));
        GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();
        jdbcOperations.update(psc, keyHolder);
        long orderId = keyHolder.getKey().longValue();
        order.setId(orderId);

        List<Taco> tacos = order.getTacos();
    }
}
```

```
        int i=0;
        for (Taco taco : tacos) {
            saveTaco(orderId, i++, taco);
        }

        return order;
    }
}
```

```
private long saveTaco(Long orderId, int orderKey, Taco taco) {
    taco.setCreatedAt(new Date());
    PreparedStatementCreatorFactory pscf =
        new PreparedStatementCreatorFactory(
            "insert into Taco "
            + "(name, created_at, taco_order, taco_order_key) "
            + "values (?, ?, ?, ?)",
            Types.VARCHAR, Types.TIMESTAMP, Type.LONG, Type.LONG
        );
    pscf.setReturnGeneratedKeys(true);

    PreparedStatementCreator psc =
        pscf.newPreparedStatementCreator(
            Arrays.asList(
                taco.getName(),
                taco.getCreatedAt(),
                orderId,
                orderKey));
}

GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();
jdbcOperations.update(psc, keyHolder);
long tacoId = keyHolder.getKey().longValue();
taco.setId(tacoId);

saveIngredientRefs(tacoId, taco.getIngredients());

return tacoId;
}
```

```

private void saveIngredientRefs(
    long tacoId, List<IngredientRef> ingredientRefs) {
    int key = 0;
    for (IngredientRef ingredientRef : ingredientRefs) {
        jdbcOperations.update(
            "insert into Ingredient_Ref (ingredient, taco, taco_key) "
            + "values (?, ?, ?)",
            ingredientRef.getIngredient(), tacoId, key++);
    }
}

```

Listing 3.11 Injecting and using OrderRepository

```

package tacos.web;
import javax.validation.Valid;

import org.springframework.stereotype.Controller;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.bind.support.SessionStatus;

import tacos.TacoOrder;
import tacos.data.OrderRepository;

@Controller
@RequestMapping("/orders")
@SessionAttributes("tacoOrder")
public class OrderController {

    private OrderRepository orderRepo;

    public OrderController(OrderRepository orderRepo) {
        this.orderRepo = orderRepo;
    }

    // ...

    @PostMapping
    public String processOrder(@Valid TacoOrder order, Errors errors, SessionStatus sessionStatus) {
        if (errors.hasErrors()) {
            return "orderForm";
        }
        orderRepo.save(order);
        sessionStatus.setComplete();

        return "redirect:/";
    }
}

```

Listing 3.12 Adding the Spring Data JDBC dependency to the build

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

```
package tacos.data;
import java.util.Optional;
import org.springframework.data.repository.Repository;
import tacos.Ingredient;

public interface IngredientRepository
    extends Repository<Ingredient, String> {

    Iterable<Ingredient> findAll();

    Optional<Ingredient> findById(String id);

    Ingredient save(Ingredient ingredient);

}
```

Listing 3.13 Defining a repository interface for persisting ingredients

```
package tacos.data;

import org.springframework.data.repository.CrudRepository;

import tacos.Ingredient;

public interface IngredientRepository
    extends CrudRepository<Ingredient, String> {

}
```

Listing 3.14 Defining a repository interface for persisting taco orders

```
package tacos.data;

import org.springframework.data.repository.CrudRepository;

import tacos.TacoOrder;

public interface OrderRepository
    extends CrudRepository<TacoOrder, Long> {

}
```

Listing 3.15 Preparing the `Taco` class for persistence

```
package tacos;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.validation.constraints.Digits;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;

import org.hibernate.validator.constraints.CreditCardNumber;
import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;

import lombok.Data;

@Data
@Table
public class TacoOrder implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    private Long id;

    // ...
}
```

```
@Table("Taco_Cloud_Order")
public class TacoOrder {
    ...
}
```

```
@Column("customer_name")
@NotBlank(message="Delivery name is required")
private String deliveryName;
```

Listing 3.16 Preparing the `Ingredient` class for persistence

```
package tacos;

import org.springframework.data.annotation.Id;
import org.springframework.data.domain.Persistable;
import org.springframework.data.relational.core.mapping.Table;

import lombok.AccessLevel;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Table
@AllArgsConstructor
@NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
public class Ingredient implements Persistable<String> {

    @Id
    private String id;

    // ...

}
```

Listing 3.17 Preparing the `Taco` class for persistence

```
package tacos;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;

import lombok.Data;

@Data
@Table
public class Taco {

    @Id
    private Long id;

    // ...

}
```

```
@Bean
public CommandLineRunner dataLoader(IngredientRepository repo) {
    return args -> {
        repo.save(new Ingredient("FLTO", "Flour Tortilla", Type.WRAP));
        repo.save(new Ingredient("COTO", "Corn Tortilla", Type.WRAP));
        repo.save(new Ingredient("GRBF", "Ground Beef", Type.PROTEIN));
        repo.save(new Ingredient("CARN", "Carnitas", Type.PROTEIN));
        repo.save(new Ingredient("TMTO", "Diced Tomatoes", Type.VEGGIES));
        repo.save(new Ingredient("LETC", "Lettuce", Type.VEGGIES));
        repo.save(new Ingredient("CHED", "Cheddar", Type.CHEESE));
        repo.save(new Ingredient("JACK", "Monterrey Jack", Type.CHEESE));
        repo.save(new Ingredient("SLSA", "Salsa", Type.SAUCE));
        repo.save(new Ingredient("SRCR", "Sour Cream", Type.SAUCE));
    };
}
```

```
@Bean
public ApplicationRunner dataLoader(IngredientRepository repo) {
    return args -> {
        repo.save(new Ingredient("FLTO", "Flour Tortilla", Type.WRAP));
        repo.save(new Ingredient("COTO", "Corn Tortilla", Type.WRAP));
        repo.save(new Ingredient("GRBF", "Ground Beef", Type.PROTEIN));
        repo.save(new Ingredient("CARN", "Carnitas", Type.PROTEIN));
        repo.save(new Ingredient("TMTO", "Diced Tomatoes", Type.VEGGIES));
        repo.save(new Ingredient("LETC", "Lettuce", Type.VEGGIES));
        repo.save(new Ingredient("CHED", "Cheddar", Type.CHEESE));
        repo.save(new Ingredient("JACK", "Monterrey Jack", Type.CHEESE));
        repo.save(new Ingredient("SLSA", "Salsa", Type.SAUCE));
        repo.save(new Ingredient("SRCR", "Sour Cream", Type.SAUCE));
    };
}
```

```
public ApplicationRunner dataLoader(IngredientRepository repo) {
    return args -> {
        List<String> version = args.getOptionValues("version");
        ...
    };
}
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa</artifactId>
    <version>2.7.6</version>
</dependency>
```

Listing 3.18 Annotating **Ingredient** for JPA persistence

```
package tacos;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.AccessLevel;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
public class Ingredient {

    @Id
    private String id;
    private String name;
    private Type type;
    public enum Type {
        WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
    }

}
```

Listing 3.19 Annotating **Taco** as an entity

```
package tacos;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import lombok.Data;

@Entity
@Data
public class Taco {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotNull
    @Size(min=5, message="Name must be at least 5 characters long")
    private String name;

    private Date createdAt = new Date();

    @Size(min=1, message="You must choose at least 1 ingredient")
    @ManyToOne()
    private List<Ingredient> ingredients = new ArrayList<>();

    public void addIngredient(Ingredient ingredient) {
        this.ingredients.add(ingredient);
    }

}
```

Listing 3.20 Annotating `TacoOrder` as a JPA entity

```
package tacos;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.validation.constraints.Digits;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;

import org.hibernate.validator.constraints.CreditCardNumber;

import lombok.Data;

@Data
@Entity
public class TacoOrder implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private Date placedAt = new Date();

    ...

    @OneToMany(cascade = CascadeType.ALL)
    private List<Taco> tacos = new ArrayList<>();

    public void addTaco(Taco taco) {
        this.tacos.add(taco);
    }

}
```

```
package tacos.data;

import org.springframework.data.repository.CrudRepository;

import tacos.Ingredient;

public interface IngredientRepository
    extends CrudRepository<Ingredient, String> {

}
```

```
package tacos.data;

import org.springframework.data.repository.CrudRepository;

import tacos.TacoOrder;

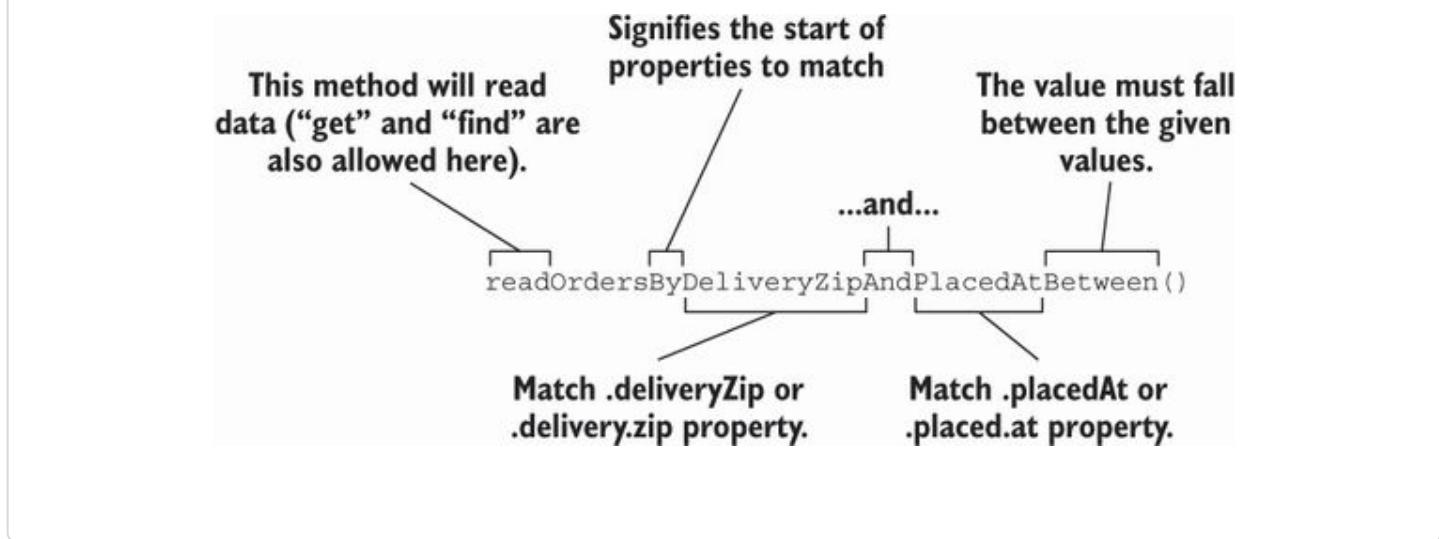
public interface OrderRepository
    extends CrudRepository<TacoOrder, Long> {

}
```

```
List<TacoOrder> findByDeliveryZip(String deliveryZip);
```

```
List<TacoOrder> readOrdersByDeliveryZipAndPlacedAtBetween(
    String deliveryZip, Date startDate, Date endDate);
```

Figure 3.2 Spring Data parses repository method signatures to determine the query that should be performed.



```
List<TacoOrder> findByDeliveryToAndDeliveryCityAllIgnoresCase(  
    String deliveryTo, String deliveryCity);
```

```
List<TacoOrder> findByDeliveryCityOrderByDeliveryTo(String city);
```

```
@Query("Order o where o.deliveryCity='Seattle'")  
List<TacoOrder> readOrdersDelivered();
```

CHAPTER 4

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-cassandra</artifactId>
4 </dependency>
```

```
1 $ docker network create cassandra-net
2 $ docker run --name my-cassandra \
3           --network cassandra-net \
4           -p 9042:9042 \
5           -d cassandra:latest
```

```
1 $ docker run -it --network cassandra-net --rm cassandra cqlsh my-cassandra
```

```
cqlsh> create keyspace tacocloud
... with replication={'class':'SimpleStrategy', 'replication_factor':1}
... and durable_writes=true;
```

```
spring:
  data:
    cassandra:
      keyspace-name: taco_cloud
      schema-action: recreate
      local-datacenter: datacenter1
```

```
spring:
  data:
    cassandra:
      keyspace-name: tacocloud
      local-datacenter: datacenter1
      contact-points:
        - casshost-1.tacocloud.com
        - casshost-2.tacocloud.com
        - casshost-3.tacocloud.com
      port: 9043
```

```
spring:
  data:
    cassandra:
      ...
      username: tacocloud
      password: s3cr3tP455w0rd
```

```
1 package tacos;
2
3 import org.springframework.data.cassandra.core.mapping.PrimaryKey;
4 import org.springframework.data.cassandra.core.mapping.Table;
5
6 import lombok.AccessLevel;
7 import lombok.AllArgsConstructor;
8 import lombok.Data;
9 import lombok.NoArgsConstructor;
10 import lombok.RequiredArgsConstructor;
11
12 @Data
13@AllArgsConstructor
14@NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
15@Table("ingredients")
16public class Ingredient {
17
18  @PrimaryKey
19  private String id;
20  private String name;
21  private Type type;
22
23  public enum Type {
24    WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
25  }
26
27 }
```

Listing 4.1 Annotating the `Taco` class for Cassandra persistence

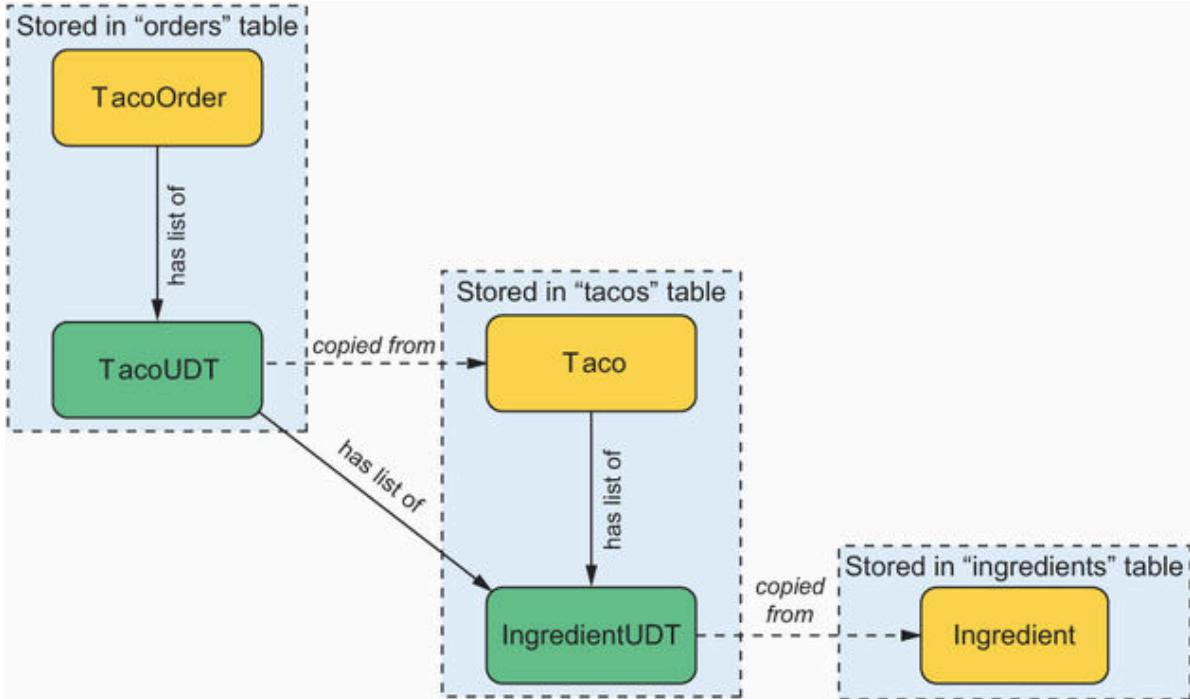
```
1 package tacos;
2 import java.util.ArrayList;
3 import java.util.Date;
4 import java.util.List;
5 import java.util.UUID;
6
7 import javax.validation.constraints.NotNull;
8 import javax.validation.constraints.Size;
9
10 import org.springframework.data.cassandra.core.cql.Ordering;
11 import org.springframework.data.cassandra.core.cql.PrimaryKeyType;
12 import org.springframework.data.cassandra.core.mapping.Column;
13 import org.springframework.data.cassandra.core.mapping.PrimaryKeyColumn;
14 import org.springframework.data.cassandra.core.mapping.Table;
15
16 import com.datastax.oss.driver.api.core.uuid.Uuids;
17
18 import lombok.Data;
19
20 @Data
21 @Table("tacos")
22 public class Taco {
23
24     @PrimaryKeyColumn(type=PrimaryKeyType.PARTITIONED)
25     private UUID id = Uuids.timeBased(); 1
26
27     @NotNull
28     @Size(min = 5, message = "Name must be at least 5 characters long")
29     private String name; 2
30
31     @PrimaryKeyColumn(type=PrimaryKeyType.CLUSTERED,
32                         ordering=Ordering.DESCENDING)
33     private Date createdAt = new Date(); 3
34
35     @Size(min=1, message="You must choose at least 1 ingredient")
36     @Column("ingredients")
37     private List<IngredientUDT> ingredients = new ArrayList<>(); 4
38
39     public void addIngredient(Ingredient ingredient) {
40         this.ingredients.add(TacoUDRUtils.toIngredientUDT(ingredient));
41     }
42 }
```

```

1 package tacos;
2
3 import org.springframework.data.cassandra.core.mapping.UserDefinedType;
4
5 import lombok.AccessLevel;
6 import lombok.Data;
7 import lombok.NoArgsConstructor;
8 import lombok.RequiredArgsConstructor;
9
10 @Data
11 @RequiredArgsConstructor
12 @NoArgsConstructor(access = AccessLevel.PRIVATE, force = true)
13 @UserDefinedType("ingredient")
14 public class IngredientUDT {
15
16     private final String name;
17
18     private final Ingredient.Type type;
19
20 }

```

Figure 4.1 Instead of using foreign keys and joins, Cassandra tables are denormalized, with user-defined types containing data copied from related tables.



```
1 cqlsh:tacocloud> select id, name, createdAt, ingredients from tacos;
2
3   id        | name      | createdat | ingredients
4 -----
5   827390... | Carnivore | 2018-04... | [{name: 'Flour Tortilla', type: 'WRAP'},
6                               {name: 'Carnitas', type: 'PROTEIN'},
7                               {name: 'Sour Cream', type: 'SAUCE'},
8                               {name: 'Salsa', type: 'SAUCE'},
9                               {name: 'Cheddar', type: 'CHEESE'}]
10
11 (1 rows)
```

Listing 4.2 Mapping the `TacoOrder` class to a Cassandra `orders` table

```
1 package tacos;
2 import java.io.Serializable;
3 import java.util.ArrayList;
4 import java.util.Date;
5 import java.util.List;
6 import java.util.UUID;
7
8 import javax.validation.constraints.Digits;
9 import javax.validation.constraints.NotBlank;
10 import javax.validation.constraints.Pattern;
11
12 import org.hibernate.validator.constraints.CreditCardNumber;
13 import org.springframework.data.cassandra.core.mapping.Column;
14 import org.springframework.data.cassandra.core.mapping.PrimaryKey;
15 import org.springframework.data.cassandra.core.mapping.Table;
16
17 import com.datastax.oss.driver.api.core.uuid.Uuids;
18
19 import lombok.Data; 1
20
21 @Data
22 @Table("orders")
23 public class TacoOrder implements Serializable { 2
24
25     private static final long serialVersionUID = 1L;
26
27     @PrimaryKey
28     private UUID id = Uuids.timeBased();
29
30     private Date placedAt = new Date();
31
32     // delivery and credit card properties omitted for brevity's sake
33
34     @Column("tacos") 3
35     private List<TacoUDT> tacos = new ArrayList<>();
36
37     public void addTaco(TacoUDT taco) {
38         this.tacos.add(taco);
39     }
40
41 }
```

```
1 package tacos;
2
3 import java.util.List;
4 import org.springframework.data.cassandra.core.mapping.UserDefinedType;
5 import lombok.Data;
6
7 @Data
8 @UserDefinedType("taco")
9 public class TacoUDT {
10
11     private final String name;
12     private final List<IngredientUDT> ingredients;
13
14 }
```

```
1 package tacos.data;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import tacos.Ingredient;
6
7 public interface IngredientRepository
8     extends CrudRepository<Ingredient, String> {
9
10 }
```

```
1 package tacos.data;
2
3 import java.util.UUID;
4
5 import org.springframework.data.repository.CrudRepository;
6
7 import tacos.TacoOrder;
8
9 public interface OrderRepository
10    extends CrudRepository<TacoOrder, UUID> {
11
12 }
```

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>
4     spring-boot-starter-data-mongodb
5   </artifactId>
6 </dependency>
```

```
1 $ docker run -p 27017:27017 -d mongo:latest
```

```
1 <dependency>
2   <groupId>de.flapdoodle.embed</groupId>
3   <artifactId>de.flapdoodle.embed.mongo</artifactId>
4   <!-- <scope>test</scope> -->
5 </dependency>
```

```
1 spring:
2   data:
3     mongodb:
4       host: mongodb.tacocloud.com
5       port: 27017
6       username: tacocloud
7       password: s3cr3tp455w0rd
8       database: tacoclouddb
```

```
1 package tacos;
2
3 import org.springframework.data.annotation.Id;
4 import org.springframework.data.mongodb.core.mapping.Document;
5
6 import lombok.AccessLevel;
7 import lombok.AllArgsConstructor;
8 import lombok.Data;
9 import lombok.NoArgsConstructor;
10
11 @Data
12 @Document
13 @NoArgsConstructor
14 @NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
15 public class Ingredient {
16
17     @Id
18     private String id;
19     private String name;
20     private Type type;
21
22     public enum Type {
23         WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
24     }
25
26 }
```

```
1 @Data
2 @NoArgsConstructor
3 @NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
4 @Document(collection="ingredients")
5 public class Ingredient {
6 ...
7 }
```

```
1 package tacos;
2 import java.util.ArrayList;
3 import java.util.Date;
4 import java.util.List;
5
6 import javax.validation.constraints.NotNull;
7 import javax.validation.constraints.Size;
8
9 import lombok.Data;
10
11 @Data
12 public class Taco {
13
14     @NotNull
15     @Size(min=5, message="Name must be at least 5 characters long")
16     private String name;
17
18     private Date createdAt = new Date();
19
20     @Size(min=1, message="You must choose at least 1 ingredient")
21     private List<Ingredient> ingredients = new ArrayList<>();
22
23     public void addIngredient(Ingredient ingredient) {
24         this.ingredients.add(ingredient);
25     }
26
27 }
```

```
1 package tacos;
2 import java.io.Serializable;
3 import java.util.ArrayList;
4 import java.util.Date;
5 import java.util.List;
6
7 import javax.validation.constraints.Digits;
8 import javax.validation.constraints.NotBlank;
9 import javax.validation.constraints.Pattern;
10
11 import org.hibernate.validator.constraints.CreditCardNumber;
12 import org.springframework.data.annotation.Id;
13 import org.springframework.data.mongodb.core.mapping.Document;
14
15 import lombok.Data;
16
17 @Data
18 @Document
19 public class TacoOrder implements Serializable {
20
21     private static final long serialVersionUID = 1L;
22
23     @Id
24     private String id;
25
26     private Date placedAt = new Date();
27
28     // other properties omitted for brevity's sake
29
30     private List<Taco> tacos = new ArrayList<>();
31
32     public void addTaco(Taco taco) {
33         this.tacos.add(taco);
34     }
35
36 }
```

```
1 package tacos.data;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import tacos.Ingredient;
6
7 public interface IngredientRepository
8     extends CrudRepository<Ingredient, String> {
9
10 }
```

```
1 package tacos.data;  
2  
3 import org.springframework.data.repository.CrudRepository;  
4  
5 import tacos.TacoOrder;  
6  
7 public interface OrderRepository  
8     extends CrudRepository<TacoOrder, String> {  
9  
10 }
```

CHAPTER 5

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Figure 5.1 Adding the security starter with Spring Tool Suite

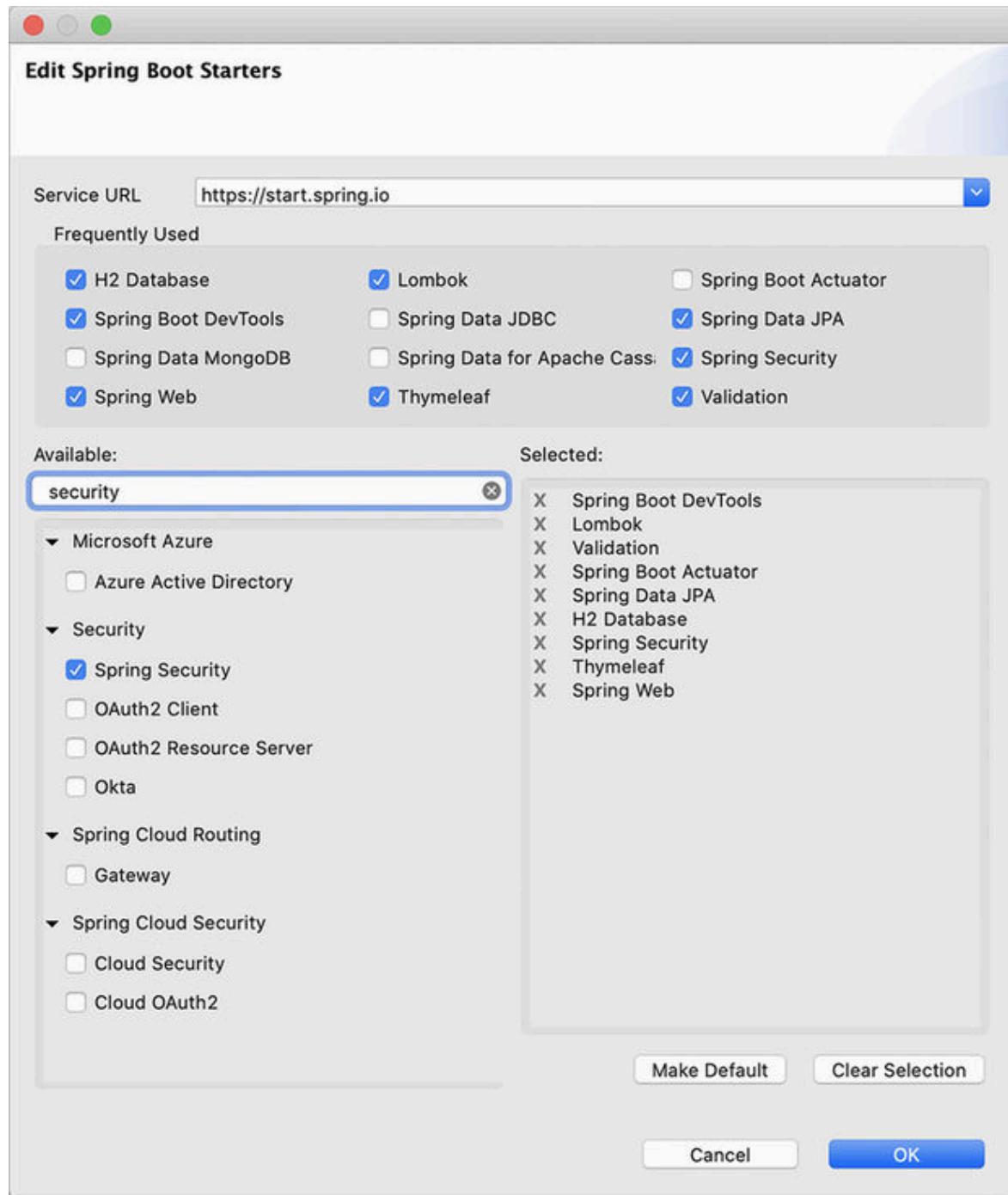
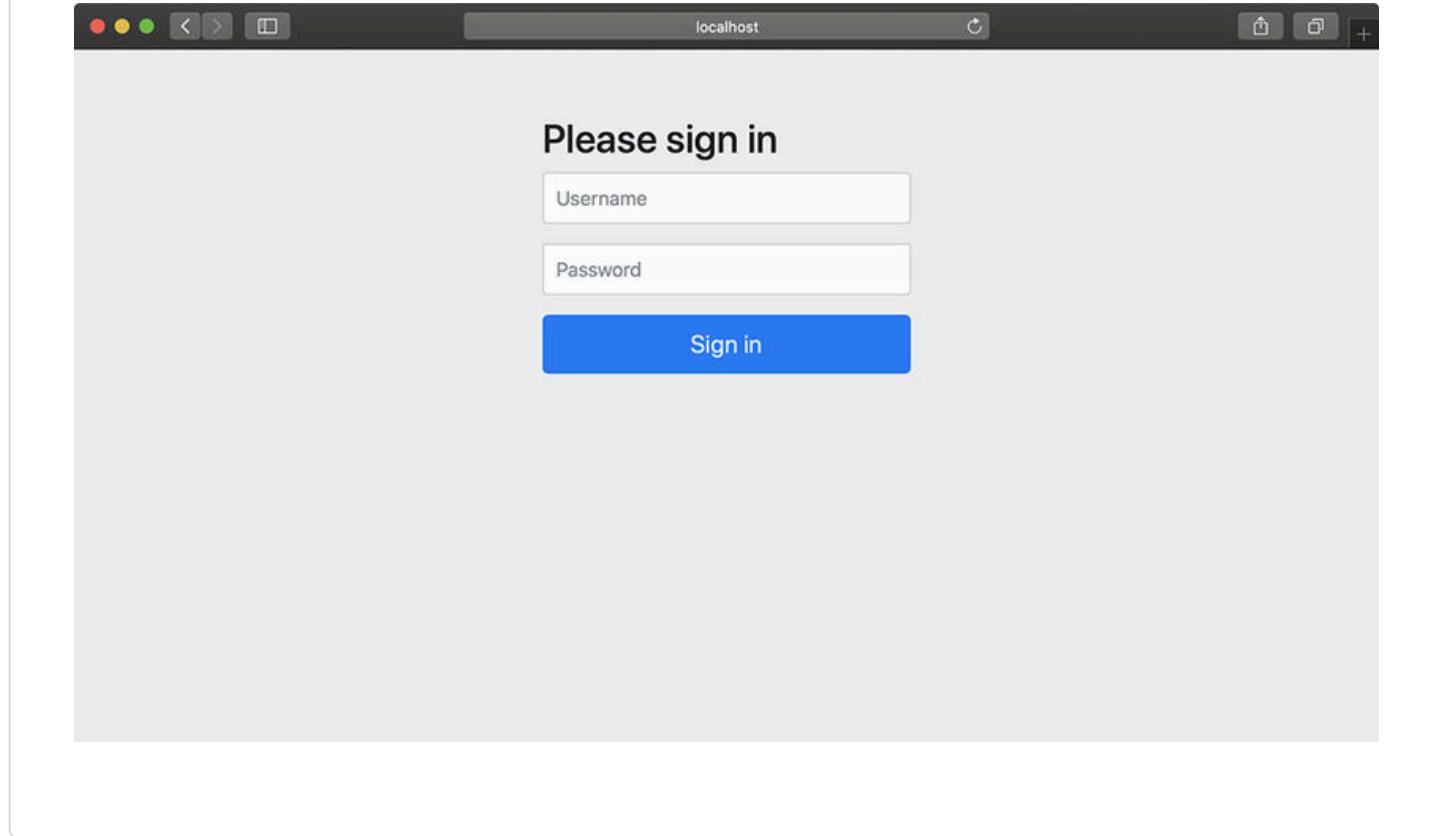


Figure 5.2 Spring Security gives you a plain login page for free.



Using generated security password: 087cf6a-027d-44bc-95d7-cbb3a798alea

Listing 5.1 A barebones configuration class for Spring Security

```
package tacos.security;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

```
public interface UserDetailsService {  
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;  
}
```

Listing 5.2 Declaring users in an in-memory user details service bean

```
@Bean  
public UserDetailsService userDetailsService(PasswordEncoder encoder) {  
    List<UserDetails> usersList = new ArrayList<>();  
    usersList.add(new User(  
        "buzz", encoder.encode("password"),  
        Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"))));  
    usersList.add(new User(  
        "woody", encoder.encode("password"),  
        Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"))));  
    return new InMemoryUserDetailsManager(usersList);  
}
```

Listing 5.3 Defining a user entity

```
package tacos;
import java.util.Arrays;
import java.util.Collection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.
SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import lombok.AccessLevel;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.RequiredArgsConstructor;
@Entity
@Data
@NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
@RequiredArgsConstructor
public class User implements UserDetails {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    private final String username;
    private final String password;
    private final String fullname;
    private final String street;
    private final String city;
    private final String state;
    private final String zip;
    private final String phoneNumber;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"));
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

```
package tacos.data;
import org.springframework.data.repository.CrudRepository;
import tacos.User;

public interface UserRepository extends CrudRepository<User, Long> {
    User findByUsername(String username);
}
```

Listing 5.4 Defining a custom user details service bean

```
@Bean
public UserDetailsService userDetailsService(UserRepository userRepo) {
    return username -> {
        User user = userRepo.findByUsername(username);
        if (user != null) return user;

        throw new UsernameNotFoundException("User '" + username + "' not found");
    };
}
```

Listing 5.5 A user registration controller

```
package tacos.security;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import tacos.data.UserRepository;

@Controller
@RequestMapping("/register")
public class RegistrationController {

    private UserRepository userRepo;
    private PasswordEncoder passwordEncoder;

    public RegistrationController(
        UserRepository userRepo, PasswordEncoder passwordEncoder) {
        this.userRepo = userRepo;
        this.passwordEncoder = passwordEncoder;
    }

    @GetMapping
    public String registerForm() {
        return "registration";
    }

    @PostMapping
    public String processRegistration(RegistrationForm form) {
        userRepo.save(form.toUser(passwordEncoder));
        return "redirect:/login";
    }
}
```

Listing 5.6 A Thymeleaf registration form view

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Taco Cloud</title>
</head>

<body>
    <h1>Register</h1>

    <form method="POST" th:action="@{/register}" id="registerForm">

        <label for="username">Username: </label>
        <input type="text" name="username"/><br/>

        <label for="password">Password: </label>
        <input type="password" name="password"/><br/>

        <label for="confirm">Confirm password: </label>
        <input type="password" name="confirm"/><br/>

        <label for="fullname">Full name: </label>
        <input type="text" name="fullname"/><br/>

        <label for="street">Street: </label>
        <input type="text" name="street"/><br/>

        <label for="city">City: </label>
        <input type="text" name="city"/><br/>

        <label for="state">State: </label>
        <input type="text" name="state"/><br/>

        <label for="zip">Zip: </label>
        <input type="text" name="zip"/><br/>

        <label for="phone">Phone: </label>
        <input type="text" name="phone"/><br/>

        <input type="submit" value="Register"/>
    </form>
</body>
</html>
```

```
package tacos.security;
import org.springframework.security.crypto.password.PasswordEncoder;
import lombok.Data;
import tacos.User;

@Data
public class RegistrationForm {

    private String username;
    private String password;
    private String fullname;
    private String street;
    private String city;
    private String state;
    private String zip;
    private String phone;

    public User toUser(PasswordEncoder passwordEncoder) {
        return new User(
            username, passwordEncoder.encode(password),
            fullname, street, city, state, zip, phone);
    }
}
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http.build();
}
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
        .authorizeRequests()
        .antMatchers("/design", "/orders").hasRole("USER")
        .antMatchers("/", "/**").permitAll()

        .and()
        .build();
}
```

Table 5.1 Configuration methods to define how a path is to be secured ([view table figure](#))

Method	What it does
<code>access(String)</code>	Allows access if the given Spring Expression Language (SpEL) expression evaluates to <code>true</code>
<code>anonymous()</code>	Allows access to anonymous users
<code>authenticated()</code>	Allows access to authenticated users
<code>denyAll()</code>	Denies access unconditionally
<code>fullyAuthenticated()</code>	Allows access if the user is fully authenticated (not remembered)
<code>hasAnyAuthority(String...)</code>	Allows access if the user has any of the given authorities
<code>hasAnyRole(String...)</code>	Allows access if the user has any of the given roles
<code>hasAuthority(String)</code>	Allows access if the user has the given authority
<code>hasIpAddress(String)</code>	Allows access if the request comes from the given IP address
<code>hasRole(String)</code>	Allows access if the user has the given role
<code>not()</code>	Negates the effect of any of the other access methods
<code>permitAll()</code>	Allows access unconditionally
<code>rememberMe()</code>	Allows access for users who are authenticated via <code>remember-me</code>

Table 5.2 Spring Security extensions to the Spring Expression Language ([view table figure](#))

Security expression	What it evaluates to
<code>authentication</code>	The user's authentication object
<code>denyAll</code>	Always evaluates to <code>false</code>
<code>hasAnyAuthority(String... authorities)</code>	<code>true</code> if the user has been granted any of the given authorities
<code>hasAnyRole(String... roles)</code>	<code>true</code> if the user has any of the given roles
<code>hasAuthority(String authority)</code>	<code>true</code> if the user has been granted the specified authority
<code>hasPermission(Object target, Object permission)</code>	<code>true</code> if the user has access to the specified target object for the given permission
<code>hasPermission(Serializable targetId, String targetType, Object permission)</code>	<code>true</code> if the user has access to the object specified by <code>targetId</code> and the specified <code>targetType</code> for the given permission
<code>hasRole(String role)</code>	<code>true</code> if the user has the given role
<code>hasIpAddress(String ipAddress)</code>	<code>true</code> if the request comes from the given IP address
<code>isAnonymous()</code>	<code>true</code> if the user is anonymous
<code>isAuthenticated()</code>	<code>true</code> if the user is authenticated
<code>isFullyAuthenticated()</code>	<code>true</code> if the user is fully authenticated (not authenticated with <code>remember-me</code>)
<code>isRememberMe()</code>	<code>true</code> if the user is authenticated via <code>remember-me</code>
<code>permitAll</code>	Always evaluates to <code>true</code>
<code>principal</code>	The user's principal object

Listing 5.7 Using Spring expressions to define authorization rules

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
        .authorizeRequests()
            .antMatchers("/design", "/orders").access("hasRole('USER')")
            .antMatchers("/", "/**").access("permitAll()")

        .and()
        .build();
}
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
        .authorizeRequests()
            .antMatchers("/design", "/orders")
                .access("hasRole('USER') && " +
                    "T(java.util.Calendar).getInstance().get(" +
                    "T(java.util.Calendar).DAY_OF_WEEK) == " +
                    "T(java.util.Calendar).TUESDAY")
            .antMatchers("/", "/**").access("permitAll")

        .and()
        .build();
}
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
        .authorizeRequests()
            .antMatchers("/design", "/orders").access("hasRole('USER')")
            .antMatchers("/", "/**").access("permitAll()")

        .and()
            .formLogin()
                .loginPage("/login")

        .and()
        .build();
}
```

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/").setViewName("home");
    registry.addViewController("/login");
}
```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Taco Cloud</title>
</head>

<body>
    <h1>Login</h1>
    

    <div th:if="${error}">
        Unable to login. Check your username and password.
    </div>

    <p>New here? Click
        <a th:href="@{/register}">here</a> to register.</p>

    <form method="POST" th:action="@{/login}" id="loginForm">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username" /><br/>

        <label for="password">Password:</label>
        <input type="password" name="password" id="password" /><br/>

        <input type="submit" value="Login"/>
    </form>
</body>
</html>
```

```
.and()
.formLogin()
.loginPage("/login")
.loginProcessingUrl("/authenticate")
.usernameParameter("user")
.passwordParameter("pwd")
```

```
.and()
.formLogin()
.loginPage("/login")
.defaultSuccessUrl("/design")
```

```
.and()
.formLogin()
.loginPage("/login")
.defaultSuccessUrl("/design", true)
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

```
spring:
  security:
    oauth2:
      client:
        registration:
          <oauth2 or openid provider name>:
            clientId: <client id>
            clientSecret: <client secret>
            scope: <comma-separated list of requested scopes>
```

```
spring:  
  security:  
    oauth2:  
      client:  
        registration:  
          facebook:  
            clientId: <facebook client id>  
            clientSecret: <facebook client secret>  
            scope: email, public_profile
```

```
@Bean  
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
  return http  
    .authorizeRequests()  
      .mvcMatchers("/design", "/orders").hasRole("USER")  
      .anyRequest().permitAll()  
  
    .and()  
      .formLogin()  
        .loginPage("/login")  
  
    .and()  
      .oauth2Login()  
  
    ...  
  
    .and()  
      .build();  
}
```

```
.and()  
.oauth2Login()  
.loginPage("/login")
```

```
<a th:href="/oauth2/authorization/facebook">Sign in with Facebook</a>
```

```
.and()  
.logout()
```

```
<form method="POST" th:action="@{/logout}">  
    <input type="submit" value="Logout"/>  
</form>
```

```
.and()  
.logout()  
.logoutSuccessUrl("/")
```

```
<input type="hidden" name="_csrf" th:value="${_csrf.token}"/>
```

```
<form method="POST" th:action="@{/login}" id="loginForm">
```

```
.and()  
.csrf()  
.disable()
```

```
public void deleteAllOrders() {  
    orderRepository.deleteAll();  
}
```

```
@Controller  
@RequestMapping("/admin")  
public class AdminController {  
  
    private OrderAdminService adminService;  
  
    public AdminController(OrderAdminService adminService) {  
        this.adminService = adminService;  
    }  
  
    @PostMapping("/deleteOrders")  
    public String deleteAllOrders() {  
        adminService.deleteAllOrders();  
        return "redirect:/admin";  
    }  
}
```

```
.authorizeRequests()  
    ...  
    .antMatchers(HttpMethod.POST, "/admin/**")  
        .access("hasRole('ADMIN')")  
    ....
```

```
@PreAuthorize("hasRole('ADMIN')")  
public void deleteAllOrders() {  
    orderRepository.deleteAll();  
}
```

```
@Configuration  
@EnableGlobalMethodSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
    ...  
}
```

```
@PostAuthorize("hasRole('ADMIN') || " +  
    "returnObject.user.username == authentication.name")  
public TacoOrder getOrder(long id) {  
    ...  
}
```

```
@Data  
@Entity  
@Table(name="Taco_Order")  
public class TacoOrder implements Serializable {  
    ...  
    @ManyToOne  
    private User user;  
    ...  
}
```

```
@PostMapping
public String processOrder(@Valid TacoOrder order, Errors errors,
    SessionStatus sessionStatus,
    Principal principal) {

    ...

    User user = userRepository.findByUsername(
        principal.getName());

    order.setUser(user);

    ...
}
```

```
@PostMapping
public String processOrder(@Valid TacoOrder order, Errors errors,
    SessionStatus sessionStatus,
    Authentication authentication) {

    ...

    User user = (User) authentication.getPrincipal();
    order.setUser(user);

    ...
}
```

```
@PostMapping
public String processOrder(@Valid TacoOrder order, Errors errors,
    SessionStatus sessionStatus,
    @AuthenticationPrincipal User user) {

    if (errors.hasErrors()) {
        return "orderForm";
    }

    order.setUser(user);

    orderRepo.save(order);
    sessionStatus.setComplete();

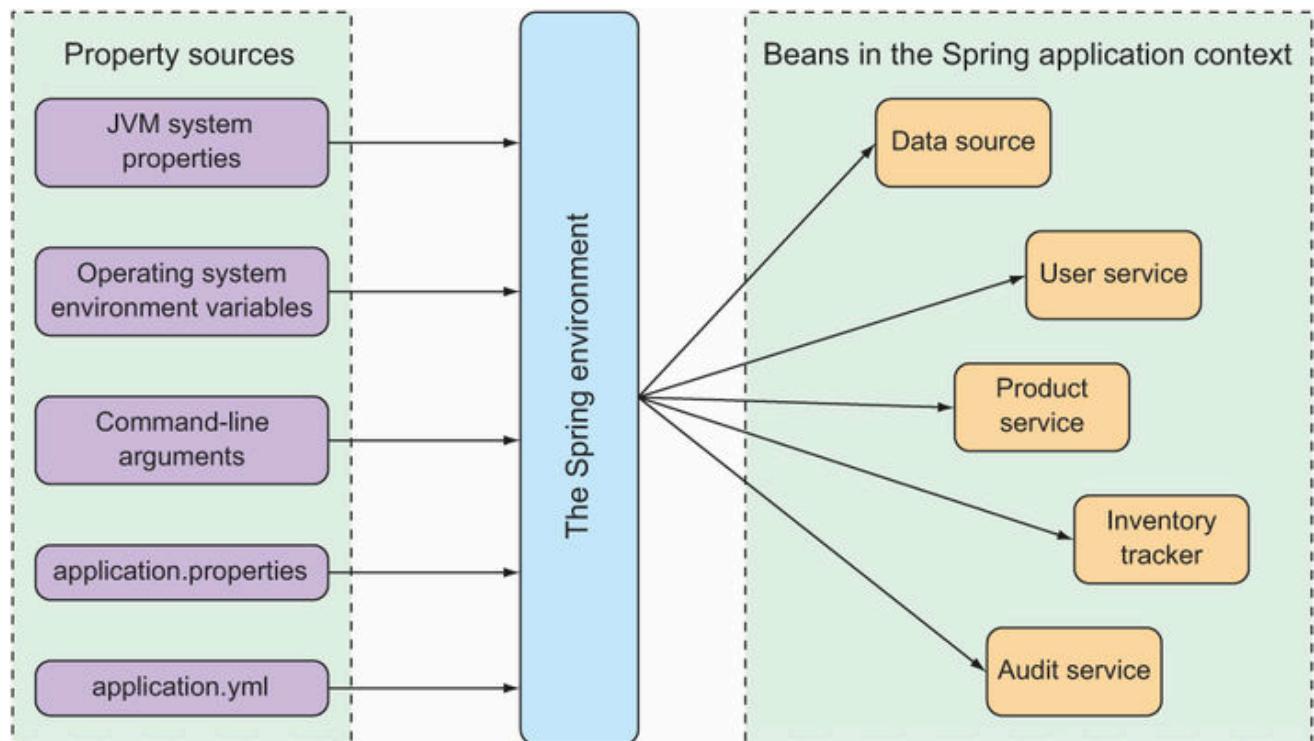
    return "redirect:/";
}
```

```
Authentication authentication =  
    SecurityContextHolder.getContext().getAuthentication();  
User user = (User) authentication.getPrincipal();
```

CHAPTER 6

```
@Bean  
public DataSource dataSource() {  
    return new EmbeddedDatabaseBuilder()  
        .setType(H2)  
        .addScript("taco_schema.sql")  
        .addScripts("user_data.sql", "ingredient_data.sql")  
        .build();  
}
```

Figure 6.1 The Spring environment pulls properties from property sources and makes them available to beans in the application context.



```
server.port=9090
```

```
server:  
port: 9090
```

```
$ java -jar tacocloud-0.0.5-SNAPSHOT.jar --server.port=9090
```

```
$ export SERVER_PORT=9090
```

```
spring:  
  datasource:  
    url: jdbc:mysql:/ /localhost/tacocloud  
    username: tacouser  
    password: tacopassword
```

```
spring:  
  datasource:  
    url: jdbc:mysql:/ /localhost/tacocloud  
    username: tacouser  
    password: tacopassword  
    driver-class-name: com.mysql.jdbc.Driver
```

```
spring:  
  datasource:  
    schema:  
      - order-schema.sql  
      - ingredient-schema.sql  
      - taco-schema.sql  
      - user-schema.sql  
    data:  
      - ingredients.sql
```

```
spring:  
  datasource:  
    jndi-name: java:/comp/env/jdbc/tacoCloudDS
```

```
server:  
  port: 0
```

```
$ keytool -keystore mykeys.jks -genkey -alias tomcat -keyalg RSA
```

```
server:  
  port: 8443  
  ssl:  
    key-store: file:/ /path/to/mykeys.jks  
    key-store-password: letmein  
    key-password: letmein
```

```
2021-07-29 17:24:24.187 INFO 52240 --- [nio-8080-exec-1]
↳ com.example.demo.Hello           Here's a log entry.
2021-07-29 17:24:24.187 INFO 52240 --- [nio-8080-exec-1]
↳ com.example.demo.Hello           Here's another log entry.
2021-07-29 17:24:24.187 INFO 52240 --- [nio-8080-exec-1]
↳ com.example.demo.Hello           And here's one more.
```

```
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>
        %d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
      </pattern>
    </encoder>
  </appender>
  <logger name="root" level="INFO"/>
  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

```
17:25:09.088 [http-nio-8080-exec-1] INFO  com.example.demo.Hello - 
                                         Here's a log entry.
17:25:09.088 [http-nio-8080-exec-1] INFO  com.example.demo.Hello - 
                                         Here's another log entry.
17:25:09.088 [http-nio-8080-exec-1] INFO  com.example.demo.Hello - 
                                         And here's one more.
```

```
logging:
  level:
    root: WARN
    org:
      springframework:
        security: DEBUG
```

```
logging:
  level:
    root: WARN
    org.springframework.security: DEBUG
```

```
logging:
  file:
    path: /var/logs/
    file: TacoCloud.log
  level:
    root: WARN
    org:
      springframework:
        security: DEBUG
```

```
greeting:
  welcome: ${spring.application.name}
```

```
greeting:
  welcome: You are using ${spring.application.name}.
```

```
@GetMapping
public String ordersForUser(
  @AuthenticationPrincipal User user, Model model) {

  model.addAttribute("orders",
    orderRepo.findByUserOrderByPlacedAtDesc(user));

  return "orderList";
}
```

```
List<Order> findByUserOrderByPlacedAtDesc(User user);
```

```
@GetMapping  
public String ordersForUser(  
    @AuthenticationPrincipal User user, Model model) {  
  
    Pageable pageable = PageRequest.of(0, 20);  
    model.addAttribute("orders",  
        orderRepo.findByUserOrderByPlacedAtDesc(user, pageable));  
  
    return "orderList";  
}
```

```
List<TacoOrder> findByUserOrderByPlacedAtDesc(  
    User user, Pageable pageable);
```

Listing 6.1 Enabling configuration properties in `OrderController`

```
@Controller
@RequestMapping("/orders")
@SessionAttributes("order")
@ConfigurationProperties(prefix="taco.orders")
public class OrderController {

    private int pageSize = 20;

    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }

    ...
    @GetMapping
    public String ordersForUser(
        @AuthenticationPrincipal User user, Model model) {

        Pageable pageable = PageRequest.of(0, pageSize);
        model.addAttribute("orders",
            orderRepo.findByUserOrderByPlacedAtDesc(user, pageable));
        return "orderList";
    }

}
```

```
taco:
  orders:
    pageSize: 10
```

```
$ export TACO_ORDERS_PAGESIZE=10
```

Listing 6.2 Extracting `pageSize` to a holder class

```
package tacos.web;
import org.springframework.boot.context.properties.
    ConfigurationProperties;
import org.springframework.stereotype.Component;
import lombok.Data;

@Component
@ConfigurationProperties(prefix="taco.orders")
@Data
public class OrderProps {

    private int pageSize = 20;

}
```

```
private OrderProps props;

public OrderController(OrderRepository orderRepo,
    OrderProps props) {
    this.orderRepo = orderRepo;
    this.props = props;
}

...

@GetMapping
public String ordersForUser(
    @AuthenticationPrincipal User user, Model model) {

    Pageable pageable = PageRequest.of(0, props.getPageSize());
    model.addAttribute("orders",
        orderRepo.findByUserOrderByPlacedAtDesc(user, pageable));

    return "orderList";
}
```

```

package tacos.web;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;
import org.springframework.validation.annotation.Validated;

import lombok.Data;

@Component
@ConfigurationProperties(prefix="taco.orders")
@Data
@Validated
public class OrderProps {

    @Min(value=5, message="must be between 5 and 25")
    @Max(value=25, message="must be between 5 and 25")
    private int pageSize = 20;

}

```

Figure 6.2 A warning for missing configuration property metadata

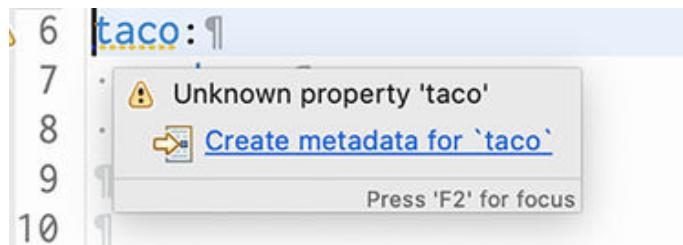


Figure 6.3 Hover documentation for configuration properties in the Spring Tool Suite

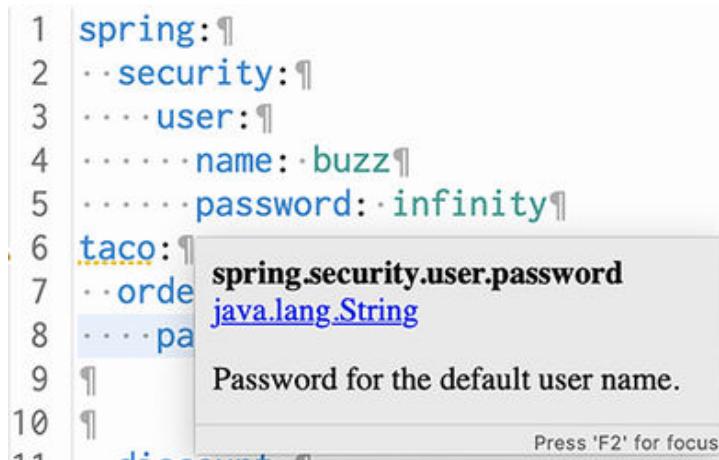
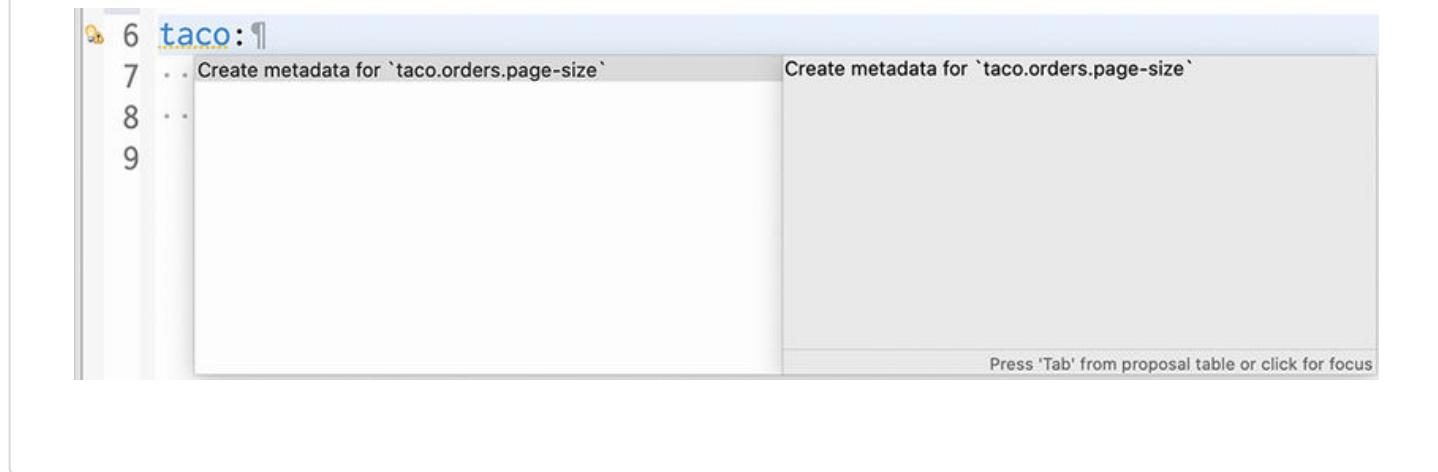


Figure 6.4 Creating configuration property metadata with the quick-fix pop-up in Spring Tool Suite



```
{"properties": [
  "name": "taco.orders.page-size",
  "type": "java.lang.String",
  "description": "A description for 'taco.orders.page-size'"
}]}
```

```
{"properties": [
  "name": "taco.orders.page-size",
  "type": "java.lang.Integer",
  "description": "Sets the maximum number of orders to display in a list."
}]}
```

Figure 6.5 Hover help for custom configuration properties

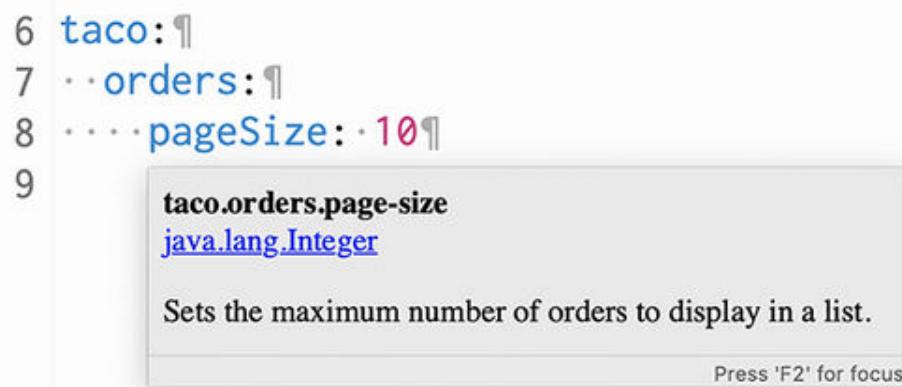


Figure 6.6 Configuration property metadata enables autocompletion of properties.

```
10 taco
11     taco.orders.page-size : String
12     spring.mustache.content-type : org.springframework.util.MimeType
13     spring.data.couchbase.auto-index : boolean
14     spring.data.couchbase.consistency : org.springframework.data.co
spring.jta.narayana.one-phase-commit : boolean
spring.jta.narayana.recovery-db-pass : String
spring.jta.narayana.recovery-db-user : String
spring.jta.narayana.recovery-modules : List<String>
spring.jta.narayana.recovery-imc-base : String
```

taco.orders.page-size
java.lang.String
Sets the maximum number of orders to display in a list.

```
% export SPRING_DATASOURCE_URL=jdbc:mysql:/ /localhost/tacocloud
% export SPRING_DATASOURCE_USERNAME=tacouser
% export SPRING_DATASOURCE_PASSWORD=tacopassword
```

```
logging:
  level:
    tacos: DEBUG
```

```
spring:
  datasource:
    url: jdbc:mysql:/ /localhost/tacocloud
    username: tacouser
    password: tacopassword
  logging:
    level:
      tacos: WARN
```

```
logging:
  level:
    tacos: DEBUG

---
spring:
  profiles: prod

  datasource:
    url: jdbc:mysql:/localhost/tacocloud
    username: tacouser
    password: tacopassword

logging:
  level:
    tacos: WARN
```

```
spring:
  profiles:
    active:
      - prod
```

```
% export SPRING_PROFILES_ACTIVE=prod
```

```
% java -jar taco-cloud.jar --spring.profiles.active=prod
```

```
% export SPRING_PROFILES_ACTIVE=prod,audit,ha
```

```
spring:  
  profiles:  
    active:  
      - prod  
      - audit  
      - ha
```

```
@Bean  
@Profile("dev")  
public CommandLineRunner dataLoader(IngredientRepository repo,  
                                  UserRepository userRepo, PasswordEncoder encoder) {  
  
  ...  
}
```

```
@Bean  
@Profile({"dev", "qa"})  
public CommandLineRunner dataLoader(IngredientRepository repo,  
                                  UserRepository userRepo, PasswordEncoder encoder) {  
  
  ...  
}
```

```
@Bean  
@Profile("!prod")  
public CommandLineRunner dataLoader(IngredientRepository repo,  
                                  UserRepository userRepo, PasswordEncoder encoder) {  
  
  ...  
}
```

```
@Profile({"!prod", "!qa"})
@Configuration
public class DevelopmentConfig {

    @Bean
    public CommandLineRunner dataLoader(IngredientRepository repo,
        UserRepository userRepo, PasswordEncoder encoder) {
        ...
    }
}
```

CHAPTER 7

Table 7.1 Spring MVC's HTTP request-handling annotations (view table figure)

Annotation	HTTP method	Typical use ¹
@GetMapping	HTTP GET requests	Reading resource data
@PostMapping	HTTP POST requests	Creating a resource
@PutMapping	HTTP PUT requests	Updating a resource
@PatchMapping	HTTP PATCH requests	Updating a resource
@DeleteMapping	HTTP DELETE requests	Deleting a resource
@RequestMapping	General-purpose request handling; HTTP method specified in the method attribute	

Listing 7.1 A RESTful controller for taco design API requests

```
1 package tacos.web.api;
2
3 import org.springframework.data.domain.PageRequest;
4 import org.springframework.data.domain.Sort;
5 import org.springframework.web.bind.annotation.CrossOrigin;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import tacos.Taco;
11 import tacos.data.TacoRepository;
12
13 @RestController
14 @RequestMapping(path="/api/tacos",
15                  produces="application/json")
16 @CrossOrigin(origins="http://tacocloud:8080")
17 public class TacoController {
18     private TacoRepository tacoRepo;
19
20     public TacoController(TacoRepository tacoRepo) {
21         this.tacoRepo = tacoRepo;
22     }
23
24     @GetMapping(params="recent")
25     public Iterable<Taco> recentTacos() {
26         PageRequest page = PageRequest.of(
27             0, 12, Sort.by("createdAt").descending());
28         return tacoRepo.findAll(page).getContent();
29     }
30 }
```

```
@RequestMapping(path="/api/tacos",
    produces={"application/json", "text/xml"})
```

```
@RestController
@RequestMapping(path="/api/tacos",
    produces="application/json")
@CrossOrigin(origins={"http://tacocloud:8080", "http://tacocloud.com"})
public class TacoController {
    ...
}
```

```
$ curl localhost:8080/api/tacos?recent
```

```
1 $ http :8080/api/tacos?recent
```

```
1  @Bean
2  public CommandLineRunner dataLoader(
3      IngredientRepository repo,
4      UserRepository userRepo,
5      PasswordEncoder encoder,
6      TacoRepository tacoRepo) {
7      return args -> {
8          Ingredient flourTortilla = new Ingredient(
9              "FLTO", "Flour Tortilla", Type.WRAP);
10         Ingredient cornTortilla = new Ingredient(
11             "COTO", "Corn Tortilla", Type.WRAP);
12         Ingredient groundBeef = new Ingredient(
13             "GRBF", "Ground Beef", Type.PROTEIN);
14         Ingredient carnitas = new Ingredient(
15             "CARN", "Carnitas", Type.PROTEIN);
16         Ingredient tomatoes = new Ingredient(
17             "TMTO", "Diced Tomatoes", Type.VEGGIES);
18         Ingredient lettuce = new Ingredient(
19             "LETC", "Lettuce", Type.VEGGIES);
20         Ingredient cheddar = new Ingredient(
21             "CHED", "Cheddar", Type.CHEESE);
22         Ingredient jack = new Ingredient(
23             "JACK", "Monterrey Jack", Type.CHEESE);
24         Ingredient salsa = new Ingredient(
25             "SLSA", "Salsa", Type.SAUCE);
26         Ingredient sourCream = new Ingredient(
27             "SRCR", "Sour Cream", Type.SAUCE);
28         repo.save(flourTortilla);
29         repo.save(cornTortilla);
30         repo.save(groundBeef);
31         repo.save(carnitas);
32         repo.save(tomatoes);
33         repo.save(lettuce);
34         repo.save(cheddar);
35         repo.save(jack);
36         repo.save(salsa);
37         repo.save(sourCream);
38
39         Taco tacol = new Taco();
40         tacol.setName("Carnivore");
41         tacol.setIngredients(Arrays.asList(
42             flourTortilla, groundBeef, carnitas,
43             sourCream, salsa, cheddar));
44         tacoRepo.save(tacol);
45
46         Taco taco2 = new Taco();
47         taco2.setName("Bovine Bounty");
48         taco2.setIngredients(Arrays.asList(
49             cornTortilla, groundBeef, cheddar,
50             jack, sourCream));
51         tacoRepo.save(taco2);
52
53         Taco taco3 = new Taco();
54         taco3.setName("Veg-Out");
55         taco3.setIngredients(Arrays.asList(
56             flourTortilla, cornTortilla, tomatoes,
57             lettuce, salsa));
58         tacoRepo.save(taco3);
59     };
60 }
```

```
1 $ curl localhost:8080/api/tacos?recent
2 [
3   {
4     "id": 4,
5     "name": "Veg-Out",
6     "createdAt": "2021-08-02T00:47:09.624+00:00",
7     "ingredients": [
8       { "id": "FLTO", "name": "Flour Tortilla", "type": "WRAP" },
9       { "id": "COTO", "name": "Corn Tortilla", "type": "WRAP" },
10      { "id": "TMTO", "name": "Diced Tomatoes", "type": "VEGGIES" },
11      { "id": "LETC", "name": "Lettuce", "type": "VEGGIES" },
12      { "id": "SLSA", "name": "Salsa", "type": "SAUCE" }
13    ]
14  },
15  {
16    "id": 3,
17    "name": "Bovine Bounty",
18    "createdAt": "2021-08-02T00:47:09.621+00:00",
19    "ingredients": [
20      { "id": "COTO", "name": "Corn Tortilla", "type": "WRAP" },
21      { "id": "GRBF", "name": "Ground Beef", "type": "PROTEIN" },
22      { "id": "CHED", "name": "Cheddar", "type": "CHEESE" },
23      { "id": "JACK", "name": "Monterrey Jack", "type": "CHEESE" },
24      { "id": "SRCR", "name": "Sour Cream", "type": "SAUCE" }
25    ]
26  },
27  {
28    "id": 2,
29    "name": "Carnivore",
30    "createdAt": "2021-08-02T00:47:09.520+00:00",
31    "ingredients": [
32      { "id": "FLTO", "name": "Flour Tortilla", "type": "WRAP" },
33      { "id": "GRBF", "name": "Ground Beef", "type": "PROTEIN" },
34      { "id": "CARN", "name": "Carnitas", "type": "PROTEIN" },
35      { "id": "SRCR", "name": "Sour Cream", "type": "SAUCE" },
36      { "id": "SLSA", "name": "Salsa", "type": "SAUCE" },
37      { "id": "CHED", "name": "Cheddar", "type": "CHEESE" }
38    ]
39  }
40 ]
```

```
1 @GetMapping("/{id}")
2 public Optional<Taco> tacoById(@PathVariable("id") Long id) {
3   return tacoRepo.findById(id);
4 }
```

```
1 @GetMapping("/{id}")
2 public ResponseEntity<Taco> tacoById(@PathVariable("id") Long id) {
3     Optional<Taco> optTaco = tacoRepo.findById(id);
4     if (optTaco.isPresent()) {
5         return new ResponseEntity<>(optTaco.get(), HttpStatus.OK);
6     }
7     return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
8 }
```

```
1 @PostMapping(consumes="application/json")
2 @ResponseStatus(HttpStatus.CREATED)
3 public Taco postTaco(@RequestBody Taco taco) {
4     return tacoRepo.save(taco);
5 }
```

```
1 @PutMapping(path="/{orderId}" , consumes="application/json")
2 public TacoOrder putOrder(
3                 @PathVariable("orderId") Long orderId,
4                 @RequestBody TacoOrder order) {
5     order.setId(orderId);
6     return repo.save(order);
7 }
```

```
1 @PatchMapping(path="/{orderId}", consumes="application/json")
2 public TacoOrder patchOrder(@PathVariable("orderId") Long orderId,
3                             @RequestBody TacoOrder patch) {
4
5     TacoOrder order = repo.findById(orderId).get();
6     if (patch.getDeliveryName() != null) {
7         order.setDeliveryName(patch.getDeliveryName());
8     }
9     if (patch.getDeliveryStreet() != null) {
10        order.setDeliveryStreet(patch.getDeliveryStreet());
11    }
12    if (patch.getDeliveryCity() != null) {
13        order.setDeliveryCity(patch.getDeliveryCity());
14    }
15    if (patch.getDeliveryState() != null) {
16        order.setDeliveryState(patch.getDeliveryState());
17    }
18    if (patch.getDeliveryZip() != null) {
19        order.setDeliveryZip(patch.getDeliveryZip());
20    }
21    if (patch.getCcNumber() != null) {
22        order.setCcNumber(patch.getCcNumber());
23    }
24    if (patch.getCcExpiration() != null) {
25        order.setCcExpiration(patch.getCcExpiration());
26    }
27    if (patch.getCcCVV() != null) {
28        order.setCcCVV(patch.getCcCVV());
29    }
30    return repo.save(order);
31 }
```

```
1 @DeleteMapping("/{orderId}")
2 @ResponseStatus(HttpStatus.NO_CONTENT)
3 public void deleteOrder(@PathVariable("orderId") Long orderId) {
4     try {
5         repo.deleteById(orderId);
6     } catch (EmptyResultDataAccessException e) {}
7 }
```

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-rest</artifactId>
4 </dependency>
```

```
1 $ curl localhost:8080/ingredients
2 {
3     "_embedded" : {
4         "ingredients" : [ {
5             "name" : "Flour Tortilla",
6             "type" : "WRAP",
7             "_links" : {
8                 "self" : {
9                     "href" : "http://localhost:8080/ingredients/FLTO"
10                },
11                "ingredient" : {
12                    "href" : "http://localhost:8080/ingredients/FLTO"
13                }
14            }
15        },
16        ...
17    ]
18 },
19 "_links" : {
20     "self" : {
21         "href" : "http://localhost:8080/ingredients"
22     },
23     "profile" : {
24         "href" : "http://localhost:8080/profile/ingredients"
25     }
26 }
27 }
```

```
1 $ curl http://localhost:8080/ingredients/FLTO
2 {
3     "name" : "Flour Tortilla",
4     "type" : "WRAP",
5     "_links" : {
6         "self" : {
7             "href" : "http://localhost:8080/ingredients/FLTO"
8         },
9         "ingredient" : {
10             "href" : "http://localhost:8080/ingredients/FLTO"
11         }
12     }
13 }
```

```
1 spring:
2   data:
3     rest:
4       base-path: /data-api
```

```
1 $ curl http://localhost:8080/data-api/tacos
2 {
3     "timestamp": "2018-02-11T16:22:12.381+0000",
4     "status": 404,
5     "error": "Not Found",
6     "message": "No message available",
7     "path": "/api/tacos"
8 }
```

```
1 $ curl localhost:8080/data-api/tacoes
2 {
3     "_embedded" : {
4         "tacoes" : [ {
5             "name" : "Carnivore",
6             "createdAt" : "2018-02-11T17:01:32.999+0000",
7             "_links" : {
8                 "self" : {
9                     "href" : "http://localhost:8080/data-api/tacoes/2"
10                },
11                "taco" : {
12                    "href" : "http://localhost:8080/data-api/tacoes/2"
13                },
14                "ingredients" : {
15                    "href" : "http://localhost:8080/data-api/tacoes/2/ingredients"
16                }
17            }
18        }]
19    },
20    "page" : {
21        "size" : 20,
22        "totalElements" : 3,
23        "totalPages" : 1,
24        "number" : 0
25    }
26 }
```

```
1 $ curl localhost:8080/api
2 {
3     "_links" : {
4         "orders" : {
5             "href" : "http://localhost:8080/data-api/orders"
6         },
7         "ingredients" : {
8             "href" : "http://localhost:8080/data-api/ingredients"
9         },
10        "tacoes" : {
11            "href" : "http://localhost:8080/data-api/tacoes{?page,size,sort}",
12            "templated" : true
13        },
14        "users" : {
15            "href" : "http://localhost:8080/data-api/users"
16        },
17        "profile" : {
18            "href" : "http://localhost:8080/data-api/profile"
19        }
20    }
21 }
```

```
1 @Data
2 @Entity
3 @RestResource(rel="tacos", path="tacos")
4 public class Taco {
5     ...
6 }
```

```
1 "tacos" : {
2     "href" : "http://localhost:8080/data-api/tacos{?page,size,sort}",
3     "templated" : true
4 },
```

```
1 $ curl "localhost:8080/data-api/tacos?size=5"
```

```
1 $ curl "localhost:8080/data-api/tacos?size=5&page=1"
```

```
1 $ curl "localhost:8080/data-api/tacos?sort=createdAt,desc&page=0&size=12"
```

Table 7.2 `RestTemplate` defines 12 unique operations, each of which is overloaded, providing a total of 41 methods. (view table figure)

Method	Description
<code>delete(...)</code>	Performs an HTTP <code>DELETE</code> request on a resource at a specified URL
<code>exchange(...)</code>	Executes a specified HTTP method against a URL, returning a <code>ResponseEntity</code> containing an object mapped from the response body
<code>execute(...)</code>	Executes a specified HTTP method against a URL, returning an object mapped from the response body
<code>getForEntity(...)</code>	Sends an HTTP <code>GET</code> request, returning a <code>ResponseEntity</code> containing an object mapped from the response body
<code>getForObject(...)</code>	Sends an HTTP <code>GET</code> request, returning an object mapped from a response body
<code>headForHeaders(...)</code>	Sends an HTTP <code>HEAD</code> request, returning the HTTP headers for the specified resource URL
<code>optionsForAllow(...)</code>	Sends an HTTP <code>OPTIONS</code> request, returning the <code>Allow</code> header for the specified URL
<code>patchForObject(...)</code>	Sends an HTTP <code>PATCH</code> request, returning the resulting object mapped from the response body
<code>postForEntity(...)</code>	<code>POST</code> s data to a URL, returning a <code>ResponseEntity</code> containing an object mapped from the response body
<code>postForLocation(...)</code>	<code>POST</code> s data to a URL, returning the URL of the newly created resource
<code>postForObject(...)</code>	<code>POST</code> s data to a URL, returning an object mapped from the response body
<code>put(...)</code>	<code>PUT</code> s resource data to the specified URL

```
1 RestTemplate rest = new RestTemplate();
```

```
1 @Bean
2 public RestTemplate restTemplate() {
3     return new RestTemplate();
4 }
```

```
1 public Ingredient getIngredientById(String ingredientId) {
2     return rest.getForObject("http://localhost:8080/ingredients/{id}",
3                             Ingredient.class, ingredientId);
4 }
```

```
1 public Ingredient getIngredientById(String ingredientId) {
2     Map<String, String> urlVariables = new HashMap<>();
3     urlVariables.put("id", ingredientId);
4     return rest.getForObject("http://localhost:8080/ingredients/{id}",
5                             Ingredient.class, urlVariables);
6 }
```

```
1 public Ingredient getIngredientById(String ingredientId) {
2     Map<String, String> urlVariables = new HashMap<>();
3     urlVariables.put("id", ingredientId);
4     URI url = UriComponentsBuilder
5             .fromHttpUrl("http://localhost:8080/ingredients/{id}")
6             .build(urlVariables);
7     return rest.getForObject(url, Ingredient.class);
8 }
```

```
1 public Ingredient getIngredientById(String ingredientId) {  
2     ResponseEntity<Ingredient> responseEntity =  
3         rest.getForEntity("http://localhost:8080/ingredients/{id}",  
4             Ingredient.class, ingredientId);  
5     log.info("Fetched time: {}",  
6             responseEntity.getHeaders().getDate());  
7     return responseEntity.getBody();  
8 }
```

```
1 public void updateIngredient(Ingredient ingredient) {  
2     rest.put("http://localhost:8080/ingredients/{id}",  
3             ingredient, ingredient.getId());  
4 }
```

```
1 public void deleteIngredient(Ingredient ingredient) {  
2     rest.delete("http://localhost:8080/ingredients/{id}",  
3             ingredient.getId());  
4 }
```

```
1 public Ingredient createIngredient(Ingredient ingredient) {  
2     return rest.postForObject("http://localhost:8080/ingredients",  
3             ingredient, Ingredient.class);  
4 }
```

```
1 public java.net.URI createIngredient(Ingredient ingredient) {  
2     return rest.postForLocation("http://localhost:8080/ingredients",  
3             ingredient);  
4 }
```

```
1 public Ingredient createIngredient(Ingredient ingredient) {  
2     ResponseEntity<Ingredient> responseEntity =  
3         rest.postForEntity("http://localhost:8080/ingredients",  
4                             ingredient,  
5                             Ingredient.class);  
6     log.info("New resource created at {}",  
7             responseEntity.getHeaders().getLocation());  
8     return responseEntity.getBody();  
9 }
```

CHAPTER 8

Listing 8.1 A controller to manage available ingredients

```
package tacos.web.api;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import tacos.Ingredient;
import tacos.data.IngredientRepository;

@RestController
@RequestMapping(path="/api/ingredients", produces="application/json")
@CrossOrigin(origins="http://localhost:8080")
public class IngredientController {

    private IngredientRepository repo;

    @Autowired
    public IngredientController(IngredientRepository repo) {
        this.repo = repo;
    }

    @GetMapping
    public Iterable<Ingredient> allIngredients() {
        return repo.findAll();
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Ingredient saveIngredient(@RequestBody Ingredient ingredient) {
        return repo.save(ingredient);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteIngredient(@PathVariable("id") String ingredientId) {
        repo.deleteById(ingredientId);
    }
}
```

```
$ curl localhost:8080/ingredients \
-H"Content-type: application/json" \
-d'{"id":"FISH", "name":"Stinky Fish", "type":"PROTEIN"}'
```

```
$ curl localhost:8080/ingredients/GRBF -X DELETE
```

```
@PostMapping
@PreAuthorize("#${hasRole('ADMIN')}")
public Ingredient saveIngredient(@RequestBody Ingredient ingredient) {
    return repo.save(ingredient);
}

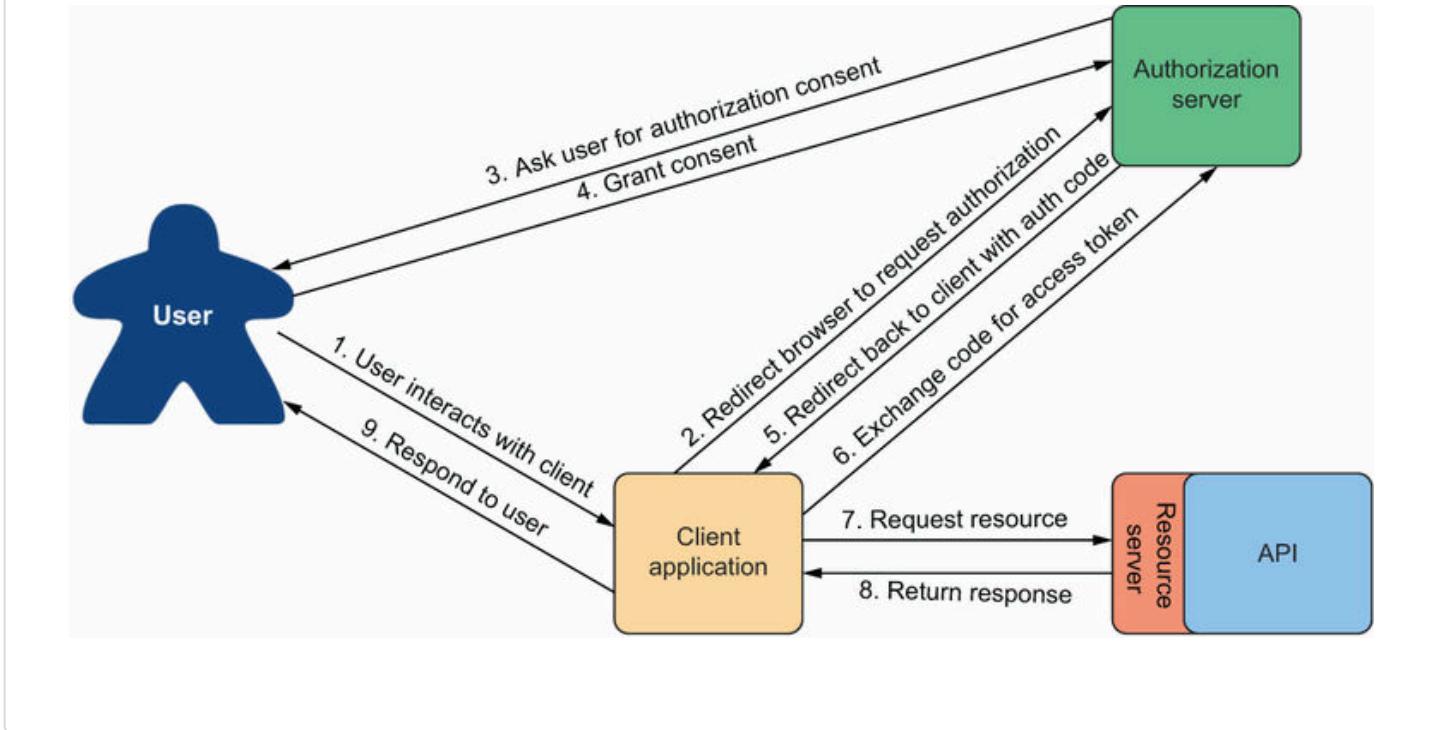
@DeleteMapping("/{id}")
@PreAuthorize("#${hasRole('ADMIN')}")
public void deleteIngredient(@PathVariable("id") String ingredientId) {
    repo.deleteById(ingredientId);
}
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .antMatchers(HttpMethod.POST, "/ingredients").hasRole("ADMIN")
        .antMatchers(HttpMethod.DELETE, "/ingredients/**").hasRole("ADMIN")

        ...
}
```

```
$ curl localhost:8080/ingredients \
-H"Content-type: application/json" \
-d'{"id":"FISH", "name":"Stinky Fish", "type":"PROTEIN"}' \
-u admin:l3tm31n
```

Figure 8.1 The OAuth 2 authorization code flow



```
<dependency>
    <groupId>org.springframework.security.experimental</groupId>
    <artifactId>spring-security-oauth2-authorization-server</artifactId>
    <version>0.1.2</version>
</dependency>
```

```
server:
  port: 9000
```

Listing 8.2 Essential security configuration for form-based login

```
package tacos.authorization;
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.web.builders.
    HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
    EnableWebSecurity;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

import tacos.authorization.users.UserRepository;

@EnableWebSecurity
public class SecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
        throws Exception {
        return http
            .authorizeRequests(authorizeRequests ->
                authorizeRequests.anyRequest().authenticated()
            )

            .formLogin()

            .and().build();
    }

    @Bean
    UserDetailsService userDetailsService(UserRepository userRepo) {
        return username -> userRepo.findByUsername(username);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

```
@Bean
public ApplicationRunner dataLoader(
    UserRepository repo, PasswordEncoder encoder) {
    return args -> {
        repo.save(
            new User("habuma", encoder.encode("password"), "ROLE_ADMIN"));
        repo.save(
            new User("tacochef", encoder.encode("password"), "ROLE_ADMIN"));
    };
}
```

```
@Configuration(proxyBeanMethods = false)
public class AuthorizationServerConfig {

    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http)
        OAuth2AuthorizationServerConfiguration
            .applyDefaultSecurity(http);
        return http
            .formLogin(Customizer.withDefaults())
            .build();
}

...
}
```

```
public interface RegisteredClientRepository {

    @Nullable
    RegisteredClient findById(String id);

    @Nullable
    RegisteredClient findByClientId(String clientId);

}
```

```

@Bean
public RegisteredClientRepository registeredClientRepository(
    PasswordEncoder passwordEncoder) {
    RegisteredClient registeredClient =
        RegisteredClient.withId(UUID.randomUUID().toString())
            .clientId("taco-admin-client")
            .clientSecret(passwordEncoder.encode("secret"))
            .clientAuthenticationMethod(
                ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
            .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
            .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
            .redirectUri(
                "http://127.0.0.1:9090/login/oauth2/code/taco-admin-client")
            .scope("writeIngredients")
            .scope("deleteIngredients")
            .scope(OidcScopes.OPENID)
            .clientSettings(
                clientSettings -> clientSettings.requireUserConsent(true))
            .build();
    return new InMemoryRegisteredClientRepository(registeredClient);
}

```

```

@Bean
public JWKSource<SecurityContext> jwkSource()
    throws NoSuchAlgorithmException {
    RSAKey rsaKey = generateRsa();
    JWKSet jwkSet = new JWKSet(rsaKey);
    return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
}

private static RSAKey generateRsa() throws NoSuchAlgorithmException {
    KeyPair keyPair = generateRsaKey();
    RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
    RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
    return new RSAKey.Builder(publicKey)
        .privateKey(privateKey)
        .keyID(UUID.randomUUID().toString())
        .build();
}

private static KeyPair generateRsaKey() throws NoSuchAlgorithmException {
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(2048);
    return keyPairGenerator.generateKeyPair();
}

@Bean
public JwtDecoder jwtDecoder(JWKSource<SecurityContext> jwkSource) {
    return OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
}

```

Figure 8.2 The authorization server login page

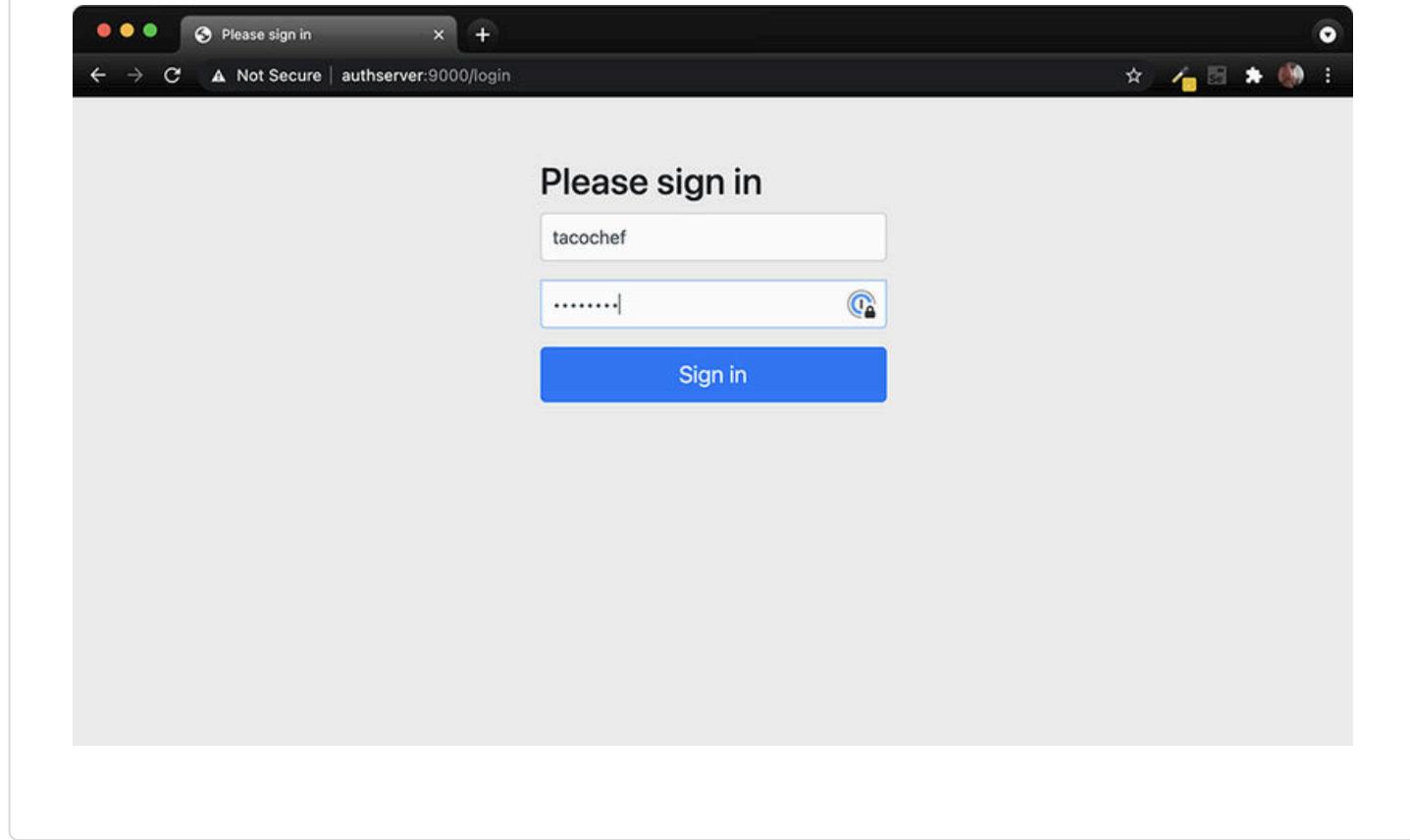
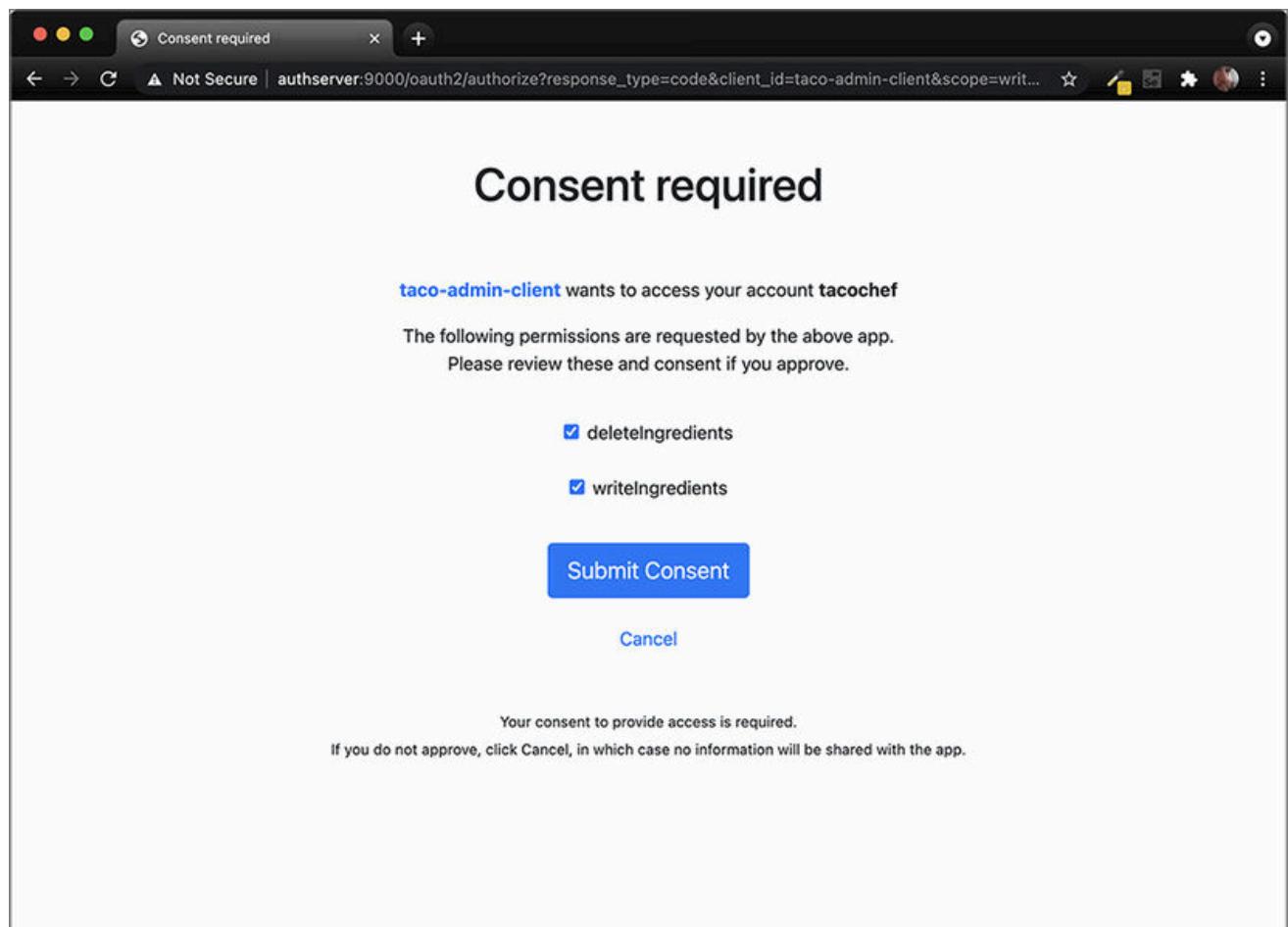


Figure 8.3 The authorization server consent page



```
$ curl localhost:9000/oauth2/token \
-H"Content-type: application/x-www-form-urlencoded" \
-d"grant_type=authorization_code" \
-d"redirect_uri=http://127.0.0.1:9090/login/oauth2/code/taco-admin-client" \
-d"code=$code" \
-u taco-admin-client:secret
```

```
{  
  "access_token": "eyJraWQ... ",  
  "refresh_token": "HOzHA5s... ",  
  "scope": "deleteIngredients writeIngredients",  
  "token_type": "Bearer",  
  "expires_in": "299"  
}
```

```
$ curl localhost:8080/ingredients \
-H"Content-type: application/json" \
-H"Authorization: Bearer eyJraWQ... " \
-d'{"id": "FISH", "name": "Stinky Fish", "type": "PROTEIN"}'
```

Figure 8.4 Decoding a JWT token at jwt.io

The screenshot shows the jwt.io website interface. On the left, under 'Encoded' (highlighted in red), is a large block of encoded JWT data. On the right, under 'Decoded' (highlighted in red), is the decoded JSON object. The 'Header' section contains:

```
{  
  "kid": "adfe4c94-9dfb-4bff-b528-2bea6c312370",  
  "typ": "JWT",  
  "alg": "RS256"  
}
```

The 'Payload' section contains:

```
{  
  "sub": "tacochef",  
  "aud": "taco-admin-client",  
  "nbf": 1621999618,  
  "scope": [  
    "deleteIngredients",  
    "writeIngredients",  
    "openid"  
  ],  
  "iss": "http://authserver:9000",  
  "exp": 1621999918,  
  "iat": 1621999618,  
  "jti": "abd90e99-af81-4f4f-8658-df322cf37d1b"  
}
```

The 'Verify Signature' section contains:

```
RSASHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  Public Key or Certificate. Enter it in plain text only if yo
```

```
$ curl localhost:9000/oauth2/token \  
-H "Content-type: application/x-www-form-urlencoded" \  
-d "grant_type=refresh_token&refresh_token=HOzHA5s..." \  
-u taco-admin-client:secret
```

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>  
</dependency>
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        ...
        .antMatchers(HttpMethod.POST, "/api/ingredients")
            .hasAuthority("SCOPE_writeIngredients")
        .antMatchers(HttpMethod.DELETE, "/api/ingredients")
            .hasAuthority("SCOPE_deleteIngredients")
        ...
}
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        ...
        .and()
            .oauth2ResourceServer(oauth2 -> oauth2.jwt())
        ...
}
```

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          jwk-set-uri: http://localhost:9000/oauth2/jwks
```

```
$ curl localhost:8080/ingredients \
  -H"Content-type: application/json" \
  -d'{"id":"CRKT", "name":"Legless Crickets", "type":"PROTEIN"}'
```

```
$ curl localhost:8080/ingredients \
-H"Content-type: application/json" \
-d'{"id":"SHMP", "name":"Coconut Shrimp", "type":"PROTEIN"}' \
-H"Authorization: Bearer $token"
```

```
$ curl localhost:8080/ingredients
[
  {
    "id": "FLTO",
    "name": "Flour Tortilla",
    "type": "WRAP"
  },
  ...
  {
    "id": "SHMP",
    "name": "Coconut Shrimp",
    "type": "PROTEIN"
  }
]
```

```
$ curl localhost:9000/oauth2/token \
-H"Content-type: application/x-www-form-urlencoded" \
-d"grant_type=refresh_token&refresh_token=$refreshToken" \
-u taco-admin-client:secret
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
    http
        .authorizeRequests(
            authorizeRequests -> authorizeRequests.anyRequest().authenticated()
        )
        .oauth2Login(
            oauth2Login ->
                oauth2Login.loginPage("/oauth2/authorization/taco-admin-client"))
        .oauth2Client(withDefaults());
    return http.build();
}
```

```
spring:
  security:
    oauth2:
      client:
        registration:
          taco-admin-client:
            provider: tacocloud
            client-id: taco-admin-client
            client-secret: secret
            authorization-grant-type: authorization_code
            redirect-uri:
              - "http://127.0.0.1:9090/login/oauth2/code/{registrationId}"
            scope: writeIngredients,deleteIngredients,openid
```

```
spring:
  security:
    oauth2:
      client:
      ...
      provider:
        tacocloud:
          issuer-uri: http://authserver:9000
```

```
127.0.0.1 authserver
```

```
spring:
  security:
    oauth2:
      client:
        provider:
          tacocloud:
            issuer-uri: http://authserver:9000
            authorization-uri: http://authserver:9000/oauth2/authorize
            token-uri: http://authserver:9000/oauth2/token
            jwk-set-uri: http://authserver:9000/oauth2/jwks
            user-info-uri: http://authserver:9000/userinfo
            user-name-attribute: sub
```

```
package tacos;

import java.util.Arrays;
import org.springframework.web.client.RestTemplate;
public class RestIngredientService implements IngredientService {

  private RestTemplate restTemplate;

  public RestIngredientService() {
    this.restTemplate = new RestTemplate();
  }

  @Override
  public Iterable<Ingredient> findAll() {
    return Arrays.asList(restTemplate.getForObject(
        "http://localhost:8080/api/ingredients",
        Ingredient[].class));
  }
  @Override
  public Ingredient addIngredient(Ingredient ingredient) {
    return restTemplate.postForObject(
        "http://localhost:8080/api/ingredients",
        ingredient,
        Ingredient.class);
  }
}
```

```

public RestIngredientService(String accessToken) {
    this.restTemplate = new RestTemplate();
    if (accessToken != null) {
        this.restTemplate
            .getInterceptors()
            .add(getBearerTokenInterceptor(accessToken));
    }
}

private ClientHttpRequestInterceptor
    getBearerTokenInterceptor(String accessToken) {
    ClientHttpRequestInterceptor interceptor =
        new ClientHttpRequestInterceptor() {
    @Override
    public ClientHttpResponse intercept(
        HttpRequest request, byte[] bytes,
        ClientHttpRequestExecution execution) throws IOException {
        request.getHeaders().add("Authorization", "Bearer " + accessToken);
        return execution.execute(request, bytes);
    }
};

    return interceptor;
}

```

```

@Bean
@RequestScope
public IngredientService ingredientService(
    OAuth2AuthorizedClientService clientService) {
    Authentication authentication =
        SecurityContextHolder.getContext().getAuthentication();

    String accessToken = null;

    if (authentication.getClass()
        .isAssignableFrom(OAuth2AuthenticationToken.class)) {
        OAuth2AuthenticationToken oauthToken =
            (OAuth2AuthenticationToken) authentication;
        String clientRegistrationId =
            oauthToken.getAuthorizedClientRegistrationId();
        if (clientRegistrationId.equals("taco-admin-client")) {
            OAuth2AuthorizedClient client =
                clientService.loadAuthorizedClient(
                    clientRegistrationId, oauthToken.getName());
            accessToken = client.getAccessToken().getTokenValue();
        }
    }
    return new RestIngredientService(accessToken);
}

```

CHAPTER 9

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-artemis</artifactId>
</dependency>
```

Figure 9.1 ActiveMQ and Artemis choices available in the Spring Initializr

JMS

Press ⌘ for multiple adds

Spring for Apache ActiveMQ 5 MESSAGING

Spring JMS support with Apache ActiveMQ 'Classic'.

Spring for Apache ActiveMQ Artemis MESSAGING

Spring JMS support with Apache ActiveMQ Artemis.

Table 9.1 Properties for configuring the location and credentials of an Artemis broker ([view table figure](#))

Property	Description
spring.artemis.host	The broker's host
spring.artemis.port	The broker's port
spring.artemis.user	The user for accessing the broker (optional)
spring.artemis.password	The password for accessing the broker (optional)

```
spring:
  artemis:
    host: artemis.tacocloud.com
    port: 61617
    user: tacoweb
    password: l3tm31n
```

Table 9.2 Properties for configuring the location and credentials of an ActiveMQ broker ([view table figure](#))

Property	Description
spring.activemq.broker-url	The URL of the broker
spring.activemq.user	The user for accessing the broker (optional)
spring.activemq.password	The password for accessing the broker (optional)
spring.activemq.in-memory	Whether to start an in-memory broker (default: true)

```
spring:  
  activemq:  
    broker-url: tcp://activemq.tacocloud.com  
    user: tacoweb  
    password: l3tm31n
```

```
// Send raw messages  
void send(MessageCreator messageCreator) throws JmsException;  
void send(Destination destination, MessageCreator messageCreator)  
    throws JmsException;  
void send(String destinationName, MessageCreator messageCreator)  
    throws JmsException;  
// Send messages converted from objects  
void convertAndSend(Object message) throws JmsException;  
void convertAndSend(Destination destination, Object message)  
    throws JmsException;  
void convertAndSend(String destinationName, Object message)  
    throws JmsException;  
  
// Send messages converted from objects with post-processing  
void convertAndSend(Object message,  
    MessagePostProcessor postProcessor) throws JmsException;  
void convertAndSend(Destination destination, Object message,  
    MessagePostProcessor postProcessor) throws JmsException;  
void convertAndSend(String destinationName, Object message,  
    MessagePostProcessor postProcessor) throws JmsException;
```

Listing 9.1 Sending an order with `.send()` to a default destination

```
package tacos.messaging;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.Session;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Service;

@Service
public class JmsOrderMessagingService implements OrderMessagingService {
    private JmsTemplate jms;

    @Autowired
    public JmsOrderMessagingService(JmsTemplate jms) {
        this.jms = jms;
    }

    @Override
    public void sendOrder(TacoOrder order) {
        jms.send(new MessageCreator() {
            @Override
            public Message createMessage(Session session)
                    throws JMSEException {
                return session.createObjectMessage(order);
            }
        });
    }
}
```

```
@RestController
@RequestMapping(path="/api/orders",
                 produces="application/json")
@CrossOrigin(origins="http://localhost:8080")
public class OrderApiController {

    private OrderRepository repo;
    private OrderMessagingService messageService;

    public OrderApiController(
        OrderRepository repo,
        OrderMessagingService messageService) {
        this.repo = repo;
        this.messageService = messageService;
    }

    @PostMapping(consumes="application/json")
    @ResponseStatus(HttpStatus.CREATED)
    public TacoOrder postOrder(@RequestBody TacoOrder order) {
        messageService.sendOrder(order);
        return repo.save(order);
    }

    ...
}
```

```
@Override
public void sendOrder(TacoOrder order) {
    jms.send(session -> session.createObjectMessage(order));
}
```

```
spring:
  jms:
    template:
      default-destination: tacocloud.order.queue
```

```
@Bean  
public Destination orderQueue() {  
    return new ActiveMQQueue("tacocloud.order.queue");  
}
```

```
private Destination orderQueue;  
  
@Autowired  
public JmsOrderMessagingService(JmsTemplate jms,  
                                 Destination orderQueue) {  
    this.jms = jms;  
    this.orderQueue = orderQueue;  
}  
  
...  
  
@Override  
public void sendOrder(TacoOrder order) {  
    jms.send(  
        orderQueue,  
        session -> session.createObjectMessage(order));  
}
```

```
@Override  
public void sendOrder(TacoOrder order) {  
    jms.send(  
        "tacocloud.order.queue",  
        session -> session.createObjectMessage(order));  
}
```

```
@Override  
public void sendOrder(TacoOrder order) {  
    jms.convertAndSend("tacocloud.order.queue", order);  
}
```

```

public interface MessageConverter {
    Message toMessage(Object object, Session session)
        throws JMSEException, MessageConversionException;
    Object fromMessage(Message message)
}

```

Table 9.3 Spring message converters for common conversion tasks (all in the `org.springframework.jms.support.converter` package) (view table figure)

Message converter	What it does
MappingJackson2MessageConverter	Uses the Jackson 2 JSON library to convert messages to and from JSON
MarshallingMessageConverter	Uses JAXB to convert messages to and from XML
MessagingMessageConverter	Converts a <code>Message</code> from the messaging abstraction to and from a <code>Message</code> using an underlying <code>MessageConverter</code> for the payload and a <code>JmsHeaderMapper</code> to map the JMS headers to and from standard message headers
SimpleMessageConverter	Converts a <code>String</code> to and from a <code>TextMessage</code> , <code>byte</code> arrays to and from a <code>BytesMessage</code> , a <code>Map</code> to and from a <code>MapMessage</code> , and a <code>Serializable</code> to and from an <code>ObjectMessage</code>

```

@Bean
public MappingJackson2MessageConverter messageConverter() {
    MappingJackson2MessageConverter messageConverter =
        new MappingJackson2MessageConverter();
    messageConverter.setTypeIdPropertyName("_ typeId");
    return messageConverter;
}

```

```
@Bean
public MappingJackson2MessageConverter messageConverter() {
    MappingJackson2MessageConverter messageConverter =
        new MappingJackson2MessageConverter();
    messageConverter.setTypeIdPropertyName("_ typeId");

    Map<String, Class<?>> typeIdMappings = new HashMap<String, Class<?>>();
    typeIdMappings.put("order", TacoOrder.class);
    messageConverter.setTypeIdMappings(typeIdMappings);

    return messageConverter;
}
```

```
jms.send("tacocloud.order.queue",
    session -> {
        Message message = session.createObjectMessage(order);
        message.setStringProperty("X_ORDER_SOURCE", "WEB");
    });

```

```
jms.convertAndSend("tacocloud.order.queue", order, new MessagePostProcessor() {
    @Override
    public Message postProcessMessage(Message message) throws JMSEException {
        message.setStringProperty("X_ORDER_SOURCE", "WEB");
        return message;
    }
});
```

```
jms.convertAndSend("tacocloud.order.queue", order,
    message -> {
        message.setStringProperty("X_ORDER_SOURCE", "WEB");
        return message;
    });

```

```

@GetMapping("/convertAndSend/order")
public String convertAndSendOrder() {
    TacoOrder order = buildOrder();
    jms.convertAndSend("tacocloud.order.queue", order,
        this::addOrderSource);
    return "Convert and sent order";
}

private Message addOrderSource(Message message) throws JMSEException {
    message.setStringProperty("X_ORDER_SOURCE", "WEB");
    return message;
}

```

```

Message receive() throws JmsException;
Message receive(Destination destination) throws JmsException;
Message receive(String destinationName) throws JmsException;

Object receiveAndConvert() throws JmsException;
Object receiveAndConvert(Destination destination) throws JmsException;
Object receiveAndConvert(String destinationName) throws JmsException;

```

Listing 9.2 Pulling orders from a queue

```

package tacos.kitchen.messaging.jms;
import javax.jms.Message;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.support.converter.MessageConverter;
import org.springframework.stereotype.Component;

@Component
public class JmsOrderReceiver implements OrderReceiver {
    private JmsTemplate jms;
    private MessageConverter converter;

    @Autowired
    public JmsOrderReceiver(JmsTemplate jms, MessageConverter converter) {
        this.jms = jms;
        this.converter = converter;
    }
    public TacoOrder receiveOrder() {
        Message message = jms.receive("tacocloud.order.queue");
        return (TacoOrder) converter.fromMessage(message);
    }
}

```

Listing 9.3 Receiving an already-converted `TacoOrder` object

```
package tacos.kitchen.messaging.jms;

import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Component;
import tacos.TacoOrder;
import tacos.kitchen.OrderReceiver;

@Component
public class JmsOrderReceiver implements OrderReceiver {

    private JmsTemplate jms;

    public JmsOrderReceiver(JmsTemplate jms) {
        this.jms = jms;
    }

    @Override
    public TacoOrder receiveOrder() {
        return (TacoOrder) jms.receiveAndConvert("tacocloud.order.queue");
    }

}
```

Listing 9.4 An `OrderListener` component that listens for orders

```
package tacos.kitchen.messaging.jms.listener;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Profile;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

import tacos.TacoOrder;
import tacos.kitchen.KitchenUI;

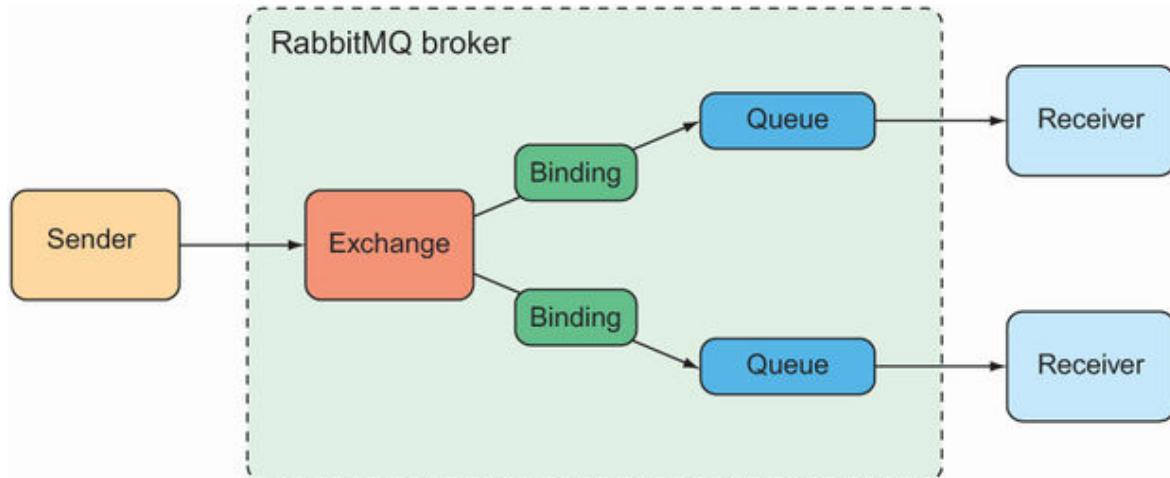
@Component
@Profile("jms-listener")
public class OrderListener {

    private KitchenUI ui;

    @Autowired
    public OrderListener(KitchenUI ui) {
        this.ui = ui;
    }

    @JmsListener(destination = "tacocloud.order.queue")
    public void receiveOrder(TacoOrder order) {
        ui.displayOrder(order);
    }
}
```

Figure 9.2 Messages sent to a RabbitMQ exchange are routed to one or more queues, based on routing keys and bindings.



```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>

```

Table 9.4 Properties for configuring the location and credentials of a RabbitMQ broker ([view table figure](#))

Property	Description
spring.rabbitmq.addresses	A comma-separated list of RabbitMQ broker addresses
spring.rabbitmq.host	The broker's host (defaults to localhost)
spring.rabbitmq.port	The broker's port (defaults to 5672)
spring.rabbitmq.username	The username for accessing the broker (optional)
spring.rabbitmq.password	The password for accessing the broker (optional)

```

spring:
  profiles: prod
  rabbitmq:
    host: rabbit.tacocloud.com
    port: 5673
    username: tacoweb
    password: 13tm31n

```

```

// Send raw messages
void send(Message message) throws AmqpException;
void send(String routingKey, Message message) throws AmqpException;
void send(String exchange, String routingKey, Message message)
    throws AmqpException;

// Send messages converted from objects
void convertAndSend(Object message) throws AmqpException;
void convertAndSend(String routingKey, Object message)
    throws AmqpException;
void convertAndSend(String exchange, String routingKey,
    Object message) throws AmqpException;

// Send messages converted from objects with post-processing
void convertAndSend(Object message, MessagePostProcessor mPP)
    throws AmqpException;
void convertAndSend(String routingKey, Object message,
    MessagePostProcessor messagePostProcessor)
    throws AmqpException;
void convertAndSend(String exchange, String routingKey,
    Object message,
    MessagePostProcessor messagePostProcessor)
    throws AmqpException;

```

Listing 9.5 Sending a message with `RabbitTemplate.send()`

```

package tacos.messaging;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import
    org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import tacos.Order;

@Service
public class RabbitOrderMessagingService
    implements OrderMessagingService {
    private RabbitTemplate rabbit;

    @Autowired
    public RabbitOrderMessagingService(RabbitTemplate rabbit) {
        this.rabbit = rabbit;
    }

    public void sendOrder(TacoOrder order) {
        MessageConverter converter = rabbit.getMessageConverter();
        MessageProperties props = new MessageProperties();
        Message message = converter.toMessage(order, props);
        rabbit.send("tacocloud.order", message);
    }
}

```

```
spring:  
  rabbitmq:  
    template:  
      exchange: tacocloud.order  
      routing-key: kitchens.central
```

```
public void sendOrder(TacoOrder order) {  
    rabbit.convertAndSend("tacocloud.order", order);  
}
```

```
@Bean  
public Jackson2JsonMessageConverter messageConverter() {  
    return new Jackson2JsonMessageConverter();  
}
```

```
public void sendOrder(TacoOrder order) {  
    MessageConverter converter = rabbit.getMessageConverter();  
    MessageProperties props = new MessageProperties();  
    props.setHeader("X_ORDER_SOURCE", "WEB");  
    Message message = converter.toMessage(order, props);  
    rabbit.send("tacocloud.order", message);  
}
```

```
public void sendOrder(TacoOrder order) {
    rabbit.convertAndSend("tacocloud.order.queue", order,
        new MessagePostProcessor() {
            @Override
            public Message postProcessMessage(Message message)
                throws AmqpException {
                MessageProperties props = message.getMessageProperties();
                props.setHeader("X_ORDER_SOURCE", "WEB");
                return message;
            }
        });
}
```

```
// Receive messages
Message receive() throws AmqpException;
Message receive(String queueName) throws AmqpException;
Message receive(long timeoutMillis) throws AmqpException;
Message receive(String queueName, long timeoutMillis) throws AmqpException;

// Receive objects converted from messages
Object receiveAndConvert() throws AmqpException;
Object receiveAndConvert(String queueName) throws AmqpException;
Object receiveAndConvert(long timeoutMillis) throws AmqpException;
Object receiveAndConvert(String queueName, long timeoutMillis)
    throws AmqpException;

// Receive type-safe objects converted from messages
<T> T receiveAndConvert(ParameterizedTypeReference<T> type)
    throws AmqpException;
<T> T receiveAndConvert(
    String queueName, ParameterizedTypeReference<T> type)
    throws AmqpException;
<T> T receiveAndConvert(
    long timeoutMillis, ParameterizedTypeReference<T> type)
    throws AmqpException;
<T> T receiveAndConvert(String queueName, long timeoutMillis,
    ParameterizedTypeReference<T> type)
    throws AmqpException;
```

Listing 9.6 Pulling orders from RabbitMQ with **RabbitTemplate**

```
package tacos.kitchen.messaging.rabbit;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.support.converter.MessageConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class RabbitOrderReceiver {
    private RabbitTemplate rabbit;
    private MessageConverter converter;

    @Autowired
    public RabbitOrderReceiver(RabbitTemplate rabbit) {
        this.rabbit = rabbit;
        this.converter = rabbit.getMessageConverter();
    }

    public TacoOrder receiveOrder() {
        Message message = rabbit.receive("tacocloud.order");
        return message != null
            ? (TacoOrder) converter.fromMessage(message)
            : null;
    }
}
```

```
public TacoOrder receiveOrder() {
    Message message = rabbit.receive("tacocloud.order.queue", 30000);
    return message != null
        ? (TacoOrder) converter.fromMessage(message)
        : null;
}
```

```
spring:
  rabbitmq:
    template:
      receive-timeout: 30000
```

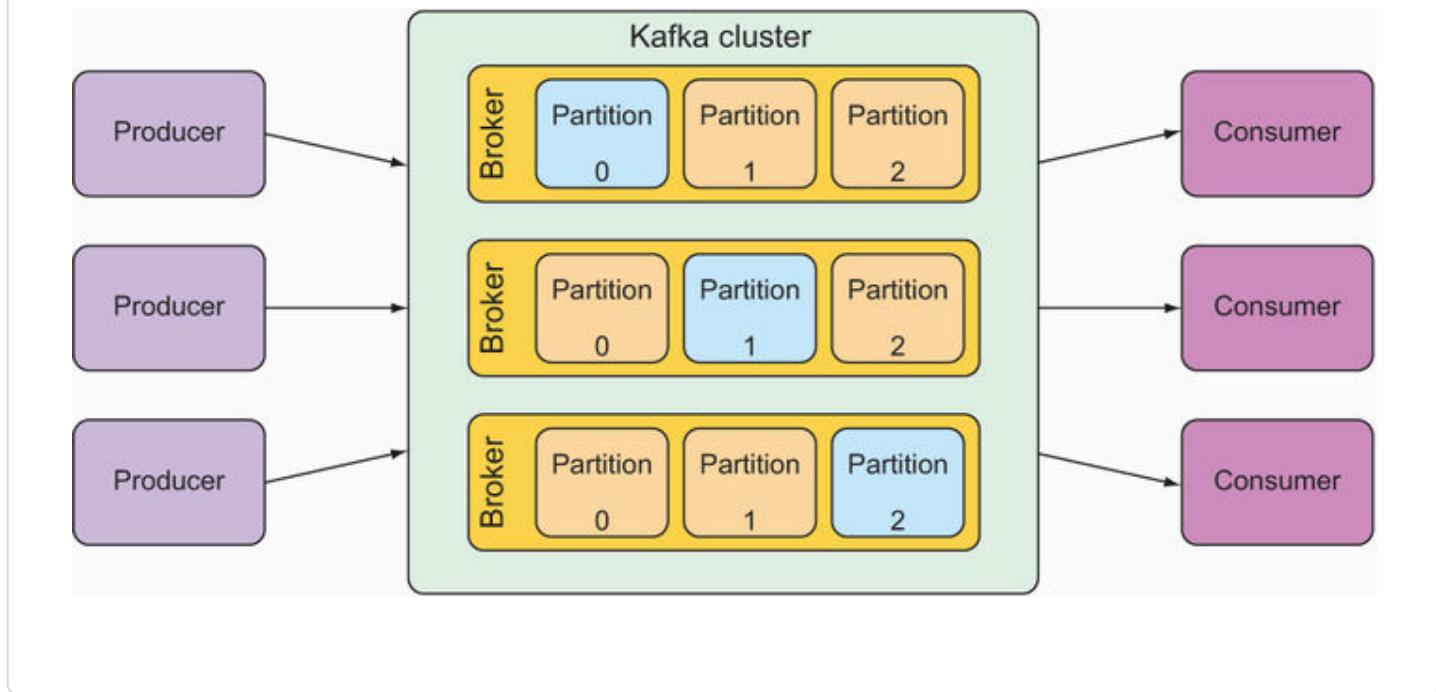
```
public TacoOrder receiveOrder() {  
    return (TacoOrder) rabbit.receiveAndConvert("tacocloud.order.queue");  
}
```

```
public TacoOrder receiveOrder() {  
    return rabbit.receiveAndConvert("tacocloud.order.queue",  
        new ParameterizedTypeReference<Order>() {});  
}
```

Listing 9.7 Declaring a method as a RabbitMQ message listener

```
package tacos.kitchen.messaging.rabbit.listener;  
  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
import tacos.TacoOrder;  
import tacos.kitchen.KitchenUI;  
  
@Component  
public class OrderListener {  
  
    private KitchenUI ui;  
  
    @Autowired  
    public OrderListener(KitchenUI ui) {  
        this.ui = ui;  
    }  
  
    @RabbitListener(queues = "tacocloud.order.queue")  
    public void receiveOrder(TacoOrder order) {  
        ui.displayOrder(order);  
    }  
}
```

Figure 9.3 A Kafka cluster is composed of multiple brokers, each acting as a leader for partitions of the topics.



```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

```
spring:
  kafka:
    bootstrap-servers:
      - kafka.tacocloud.com:9092
```

```
spring:
  kafka:
    bootstrap-servers:
      - kafka.tacocloud.com:9092
      - kafka.tacocloud.com:9093
      - kafka.tacocloud.com:9094
```

```
spring:  
  kafka:  
    bootstrap-servers:  
      - localhost:9092
```

Listing 9.8 Sending orders with **KafkaTemplate**

```
package tacos.messaging;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
import tacos.TacoOrder;
@Service
public class KafkaOrderMessagingService
    implements OrderMessagingService {

    private KafkaTemplate<String, TacoOrder> kafkaTemplate;

    @Autowired
    public KafkaOrderMessagingService(
        KafkaTemplate<String, TacoOrder> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    @Override
    public void sendOrder(TacoOrder order) {
        kafkaTemplate.send("tacocloud.orders.topic", order);
    }

}
```

```
spring:
  kafka:
    bootstrap-servers:
      - localhost:9092
    template:
      default-topic: tacocloud.orders.topic
```

```
@Override
public void sendOrder(TacoOrder order) {
    kafkaTemplate.sendDefault(order);
}
```

Listing 9.9 Receiving orders with `@KafkaListener`

```
package tacos.kitchen.messaging.kafka.listener;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
import tacos.Order;
import tacos.kitchen.KitchenUI;

@Component
public class OrderListener {

    private KitchenUI ui;

    @Autowired
    public OrderListener(KitchenUI ui) {
        this.ui = ui;
    }

    @KafkaListener(topics="tacocloud.orders.topic")
    public void handle(TacoOrder order) {
        ui.displayOrder(order);
    }

}
```

```
@KafkaListener(topics="tacocloud.orders.topic")
public void handle(
    TacoOrder order, ConsumerRecord<String, TacoOrder> record) {
    log.info("Received from partition {} with timestamp {}", 
        record.partition(), record.timestamp());

    ui.displayOrder(order);
}
```

```
@KafkaListener(topics="tacocloud.orders.topic")
public void handle(Order order, Message<Order> message) {
    MessageHeaders headers = message.getHeaders();
    log.info("Received from partition {} with timestamp {}", 
        headers.get(KafkaHeaders.RECEIVED_PARTITION_ID),
        headers.get(KafkaHeaders.RECEIVED_TIMESTAMP));
    ui.displayOrder(order);
}
```


CHAPTER 10

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-integration</artifactId>
4 </dependency>
5 <dependency>
6     <groupId>org.springframework.integration</groupId>
7     <artifactId>spring-integration-file</artifactId>
8 </dependency>
```

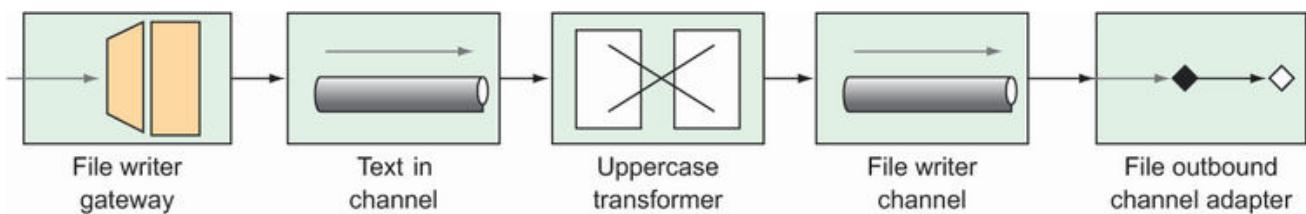
Listing 10.1 Message gateway interface to transform method invocations into messages

```
1 package sia6;
2
3 import org.springframework.integration.annotation.MessagingGateway;
4 import org.springframework.integration.file.FileHeaders;
5 import org.springframework.messaging.handler.annotation.Header;
6
7 @MessagingGateway(defaultRequestChannel="textInChannel")1
8 public interface FileWriterGateway {
9
10    void writeToFile(
11        @Header(FileHeaders.FILENAME) String filename,2
12        String data);
13
14 }
```

Listing 10.2 Defining an integration flow with Spring XML configuration

```
1 <?xml version="1.0" encoding="UTF-8"?><beans
2   xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:int="http://www.springframework.org/schema/integration"
5   xmlns:int-file="http://www.springframework.org/schema/integration/file"
6   xsi:schemaLocation="http://www.springframework.org/schema/beans
7       http://www.springframework.org/schema/beans/spring-beans.xsd
8       http://www.springframework.org/schema/integration
9       http://www.springframework.org/schema/integration/spring-integration.xsd
10      http://www.springframework.org/schema/integration/file
11      http://www.springframework.org/schema/integration/file/spring-integration-file.x
12
13    <int:channel id="textInChannel" /> 1
14
15    <int:transformer id="upperCase"
16      input-channel="textInChannel"
17      output-channel="fileWriterChannel"
18      expression="payload.toUpperCase()" /> 2
19
20    <int:channel id="fileWriterChannel" /> 3
21
22    <int-file:outbound-channel-adapter id="writer"
23      channel="fileWriterChannel"
24      directory="/tmp/sia6/files"
25      mode="APPEND"
26      append-new-line="true" /> 4
27
28 </beans>
```

Figure 10.1 The file writer integration flow



```
@Configuration
@ImportResource("classpath:/filewriter-config.xml")
public class FileWriterIntegrationConfig { ... }
```

Listing 10.3 Using Java configuration to define an integration flow

```
1 package sia6;
2
3 import java.io.File;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.integration.annotation.ServiceActivator;
7 import org.springframework.integration.annotation.Transformer;
8 import org.springframework.integration.file.FileWritingMessageHandler;
9 import org.springframework.integration.file.support.FileExistsMode;
10 import org.springframework.integration.transformer.GenericTransformer;
11
12 @Configuration
13 public class FileWriterIntegrationConfig {
14
15     @Bean
16     @Transformer(inputChannel="textInChannel",
17                  outputChannel="fileWriterChannel")
18     public GenericTransformer<String, String> uppercaseTransformer() {
19         return text -> text.toUpperCase();
20     }
21
22     @Bean
23     @ServiceActivator(inputChannel="fileWriterChannel")
24     public FileWritingMessageHandler fileWriter() {
25         FileWritingMessageHandler handler =
26             new FileWritingMessageHandler(new File("/tmp/sia6/files"));
27         handler.setExpectReply(false);
28         handler.setFileExistsMode(FileExistsMode.APPEND);
29         handler.setAppendNewLine(true);
30         return handler;
31     }
32
33 }
```

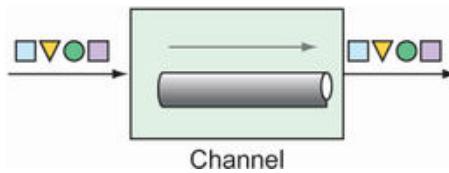
```
1 @Bean
2 public MessageChannel textInChannel() {
3     return new DirectChannel();
4 }
5 ...
6 @Bean
7 public MessageChannel fileWriterChannel() {
8     return new DirectChannel();
9 }
```

Listing 10.4 Providing a fluent API for designing integration flows

```
1 package sia6;
2
3 import java.io.File;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.integration.dsl.IntegrationFlow;
7 import org.springframework.integration.dsl.IntegrationFlows;
8 import org.springframework.integration.dsl.MessageChannels;
9 import org.springframework.integration.file.dsl.Files;
10 import org.springframework.integration.file.support.FileExistsMode;
11
12 @Configuration
13 public class FileWriterIntegrationConfig {
14
15     @Bean
16     public IntegrationFlow fileWriterFlow() {
17         return IntegrationFlows
18             .from(MessageChannels.direct("textInChannel"))
19             .<String, String>transform(t -> t.toUpperCase())
20             .handle(Files
21                 .outboundAdapter(new File("/tmp/sia6/files"))
22                 .fileExistsMode(FileExistsMode.APPEND)
23                 .appendNewLine(true))
24             .get();
25     }
26
27 }
```

```
1 @Bean
2 public IntegrationFlow fileWriterFlow() {
3     return IntegrationFlows
4         .from(MessageChannels.direct("textInChannel"))
5         .<String, String>transform(t -> t.toUpperCase())
6         .channel(MessageChannels.direct("FileWriterChannel"))
7         .handle(Files
8             .outboundAdapter(new File("/tmp/sia6/files"))
9             .fileExistsMode(FileExistsMode.APPEND)
10            .appendNewLine(true))
11     .get();
12 }
```

Figure 10.2 Message channels are conduits through which data flows between other components in an integration flow.



```
1 @Bean
2 public MessageChannel orderChannel() {
3     return new PublishSubscribeChannel();
4 }
```

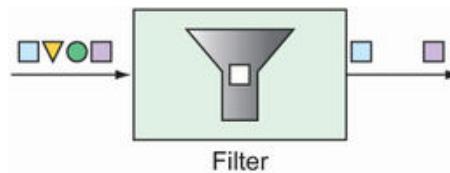
```
1 @ServiceActivator(inputChannel="orderChannel")
```

```
1 @Bean
2 public IntegrationFlow orderFlow() {
3     return IntegrationFlows
4         ...
5         .channel("orderChannel")
6         ...
7         .get();
8 }
```

```
1 @Bean
2 public MessageChannel orderChannel() {
3     return new QueueChannel();
4 }
```

```
1 @ServiceActivator(inputChannel="orderChannel",
2                     poller=@Poller(fixedRate="1000"))
```

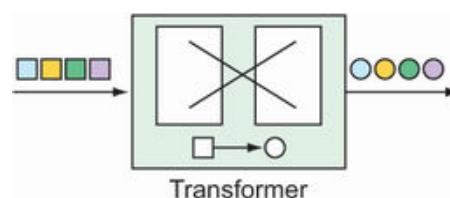
Figure 10.3 Filters based on some criteria allow or disallow messages from proceeding in the pipeline.



```
1 @Filter(inputChannel="numberChannel",
2          outputChannel="evenNumberChannel")
3 public boolean evenNumberFilter(Integer number) {
4     return number % 2 == 0;
5 }
```

```
1 @Bean
2 public IntegrationFlow evenNumberFlow(AtomicInteger integerSource) {
3     return IntegrationFlows
4         ...
5         .<Integer>filter((p) -> p % 2 == 0)
6         ...
7         .get();
8 }
```

Figure 10.4 Transformers morph messages as they flow through an integration flow.



```

1 @Bean
2 @Transformer(inputChannel="numberChannel",
3                 outputChannel="romanNumberChannel")
4 public GenericTransformer<Integer, String> romanNumTransformer() {
5     return RomanNumbers::toRoman;
6 }

```

```

1 @Bean
2 public IntegrationFlow transformerFlow() {
3     return IntegrationFlows
4         ...
5         .transform(RomanNumbers::toRoman)
6         ...
7         .get();
8 }

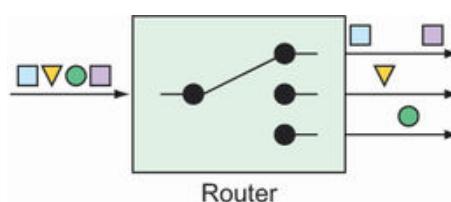
```

```

1 @Bean
2 public RomanNumberTransformer romanNumberTransformer() {
3     return new RomanNumberTransformer();
4 }
5
6 @Bean
7 public IntegrationFlow transformerFlow(
8         RomanNumberTransformer romanNumberTransformer) {
9     return IntegrationFlows
10        ...
11        .transform(romanNumberTransformer)
12        ...
13        .get();
14 }

```

Figure 10.5 Routers direct messages to different channels, based on some criteria applied to the messages.



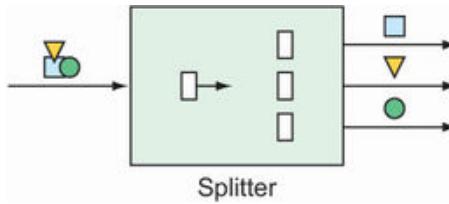
```

1  @Bean
2  @Router(inputChannel="numberChannel")
3  public AbstractMessageRouter evenOddRouter() {
4      return new AbstractMessageRouter() {
5          @Override
6          protected Collection<MessageChannel>
7              determineTargetChannels(Message<?> message) {
8              Integer number = (Integer) message.getPayload();
9              if (number % 2 == 0) {
10                  return Collections.singleton(evenChannel());
11              }
12              return Collections.singleton(oddChannel());
13          }
14      };
15  }
16
17 @Bean
18 public MessageChannel evenChannel() {
19     return new DirectChannel();
20 }
21
22 @Bean
23 public MessageChannel oddChannel() {
24     return new DirectChannel();
25 }
```

```

1  @Bean
2  public IntegrationFlow numberRoutingFlow(AtomicInteger source) {
3      return IntegrationFlows
4          ...
5          .<Integer, String>route(n -> n%2==0 ? "EVEN":"ODD", mapping -> mapping
6              .subFlowMapping("EVEN", sf -> sf
7                  .<Integer, Integer>transform(n -> n * 10)
8                  .handle((i,h) -> { ... })
9                  )
10                 .subFlowMapping("ODD", sf -> sf
11                     .transform(RomanNumbers::toRoman)
12                     .handle((i,h) -> { ... })
13                     )
14                 )
15             .get();
16 }
```

Figure 10.6 Splitters break down messages into two or more separate messages that can be handled by separate subflows.



```
1 public class OrderSplitter {  
2     public Collection<Object> splitOrderIntoParts(PurchaseOrder po) {  
3         ArrayList<Object> parts = new ArrayList<>();  
4         parts.add(po.getBillingInfo());  
5         parts.add(po.getLineItems());  
6         return parts;  
7     }  
8 }
```

```
1 @Bean  
2 @Splitter(inputChannel="poChannel",  
3             outputChannel="splitOrderChannel")  
4 public OrderSplitter orderSplitter() {  
5     return new OrderSplitter();  
6 }
```

```
1 @Bean  
2 @Router(inputChannel="splitOrderChannel")  
3 public MessageRouter splitOrderRouter() {  
4     PayloadTypeRouter router = new PayloadTypeRouter();  
5     router.setChannelMapping(  
6         BillingInfo.class.getName(), "billingInfoChannel");  
7     router.setChannelMapping(  
8         List.class.getName(), "lineItemsChannel");  
9     return router;  
10 }
```

```
1 @Splitter(inputChannel="lineItemsChannel", outputChannel="lineItemChannel")
2 public List<LineItem> lineItemSplitter(List<LineItem> lineItems) {
3     return lineItems;
4 }
```

```
1 return IntegrationFlows
2 ...
3     .split(orderSplitter())
4     .<Object, String> route(
5         p -> {
6             if (p.getClass().isAssignableFrom(BillingInfo.class)) {
7                 return "BILLING_INFO";
8             } else {
9                 return "LINE_ITEMS";
10            }
11        }, mapping -> mapping
12        .subFlowMapping("BILLING_INFO", sf -> sf
13            .<BillingInfo> handle((billingInfo, h) -> {
14                ...
15            }))
16        .subFlowMapping("LINE_ITEMS", sf -> sf
17            .split()
18            .<LineItem> handle((lineItem, h) -> {
19                ...
20            }))
21
22    )
23    .get();
```

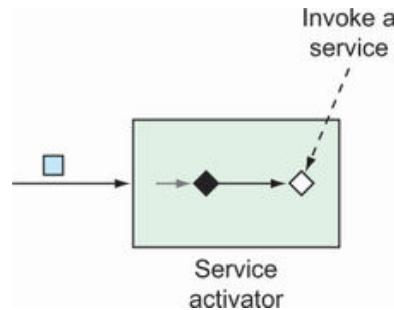
```
1 private String route(Object p) {
2     return p.getClass().isAssignableFrom(BillingInfo.class)
3         ? "BILLING_INFO"
4         : "LINE_ITEMS";
5 }
6
7 private BillingInfo handleBillingInfo(
8     BillingInfo billingInfo, MessageHeaders h) {
9     // ...
10 }
11
12 private LineItem handleLineItems(
13     LineItem lineItem, MessageHeaders h) {
14     // ...
15 }
```

```

1 return IntegrationFlows
2 ...
3     .split()
4     .route(
5         this::route,
6         mapping -> mapping
7             .subFlowMapping("BILLING_INFO", sf -> sf
8                 .<BillingInfo> handle(this::handleBillingInfo))
9             .subFlowMapping("LINE_ITEMS", sf -> sf
10                .split()
11                    .<LineItem> handle(this::handleLineItems)));

```

Figure 10.7 Service activators invoke some service by way of a `MessageHandler` on receipt of a message.



```

1 @Bean
2 @ServiceActivator(inputChannel="someChannel")
3 public MessageHandler sysoutHandler() {
4     return message -> {
5         System.out.println("Message payload: " + message.getPayload());
6     };
7 }

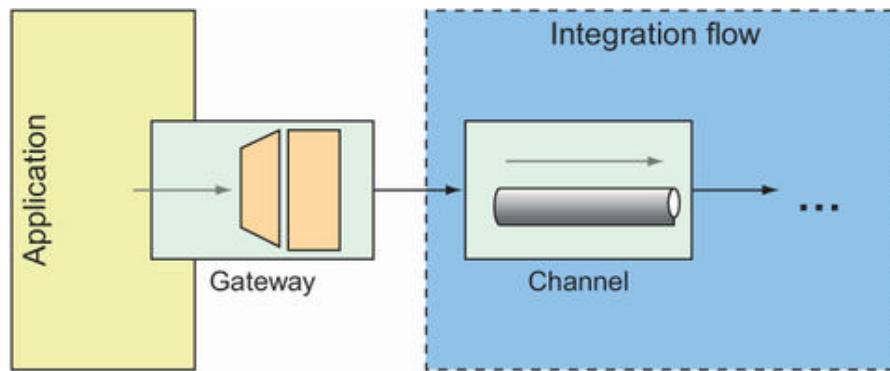
```

```
1 @Bean
2 @ServiceActivator(inputChannel="orderChannel",
3                     outputChannel="completeChannel")
4 public GenericHandler<EmailOrder> orderHandler(
5                                         OrderRepository orderRepo) {
6     return (payload, headers) -> {
7         return orderRepo.save(payload);
8     };
9 }
```

```
1 public IntegrationFlow someFlow() {
2     return IntegrationFlows
3         ...
4         .handle(msg -> {
5             System.out.println("Message payload: " + msg.getPayload());
6         })
7         .get();
8 }
```

```
1 public IntegrationFlow orderFlow(OrderRepository orderRepo) {
2     return IntegrationFlows
3         ...
4         .<EmailOrder>handle((payload, headers) -> {
5             return orderRepo.save(payload);
6         })
7         ...
8         .get();
9 }
```

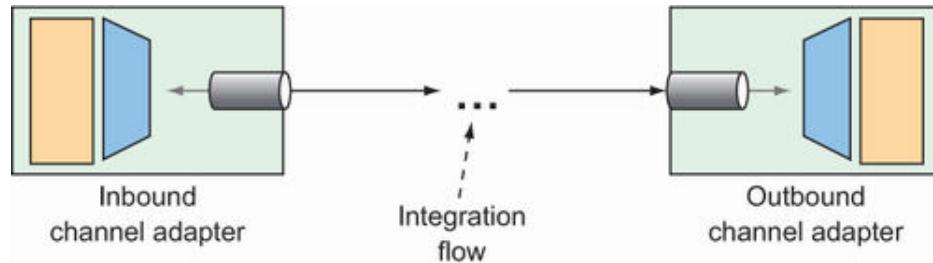
Figure 10.8 Service gateways are interfaces through which an application can submit messages to an integration flow.



```
1 package sia6;
2 import org.springframework.integration.annotation.MessagingGateway;
3 import org.springframework.stereotype.Component;
4
5 @Component
6 @MessagingGateway(defaultRequestChannel="inChannel",
7                     defaultReplyChannel="outChannel")
8 public interface UpperCaseGateway {
9     String uppercase(String in);
10 }
```

```
1 @Bean
2 public IntegrationFlow uppercaseFlow() {
3     return IntegrationFlows
4         .from("inChannel")
5         .<String, String> transform(s -> s.toUpperCase())
6         .channel("outChannel")
7         .get();
8 }
```

Figure 10.9 Channel adapters are the entry and exit points of an integration flow.



```
1 @Bean
2 @InboundChannelAdapter(
3     poller=@Poller(fixedRate="1000"), channel="numberChannel")
4 public MessageSource<Integer> numberSource(AtomicInteger source) {
5     return () -> {
6         return new GenericMessage<>(source.getAndIncrement());
7     };
8 }
```

```
1 @Bean
2 public IntegrationFlow someFlow(AtomicInteger integerSource) {
3     return IntegrationFlows
4         .from(integerSource, "getAndIncrement",
5             c -> c.poller(Pollers.fixedRate(1000)))
6         ...
7         .get();
8 }
```

```
1 @Bean
2 @InboundChannelAdapter(channel="file-channel",
3     poller=@Poller(fixedDelay="1000"))
4 public MessageSource<File> fileReadingMessageSource() {
5     FileReadingMessageSource sourceReader = new FileReadingMessageSource();
6     sourceReader.setDirectory(new File(INPUT_DIR));
7     sourceReader.setFilter(new SimplePatternFileListFilter(FILE_PATTERN));
8     return sourceReader;
9 }
```

```
1 @Bean
2 public IntegrationFlow fileReaderFlow() {
3     return IntegrationFlows
4         .from(Files.inboundAdapter(new File(INPUT_DIR)))
5             .patternFilter(FILE_PATTERN))
6         .get();
7 }
```

Table 10.1 Spring Integration provides more than two dozen endpoint modules for integration with external systems. ([view table figure](#))

Module	Dependency artifact ID (Group ID: <code>org.springframework.integration</code>)
AMQP	<code>spring-integration-amqp</code>
Application events	<code>spring-integration-event</code>
Atom and RSS	<code>spring-integration-feed</code>
Email	<code>spring-integration-mail</code>
Filesystem	<code>spring-integration-file</code>
FTP/FTPS	<code>spring-integration-ftp</code>
GemFire	<code>spring-integration-gemfire</code>
HTTP	<code>spring-integration-http</code>
JDBC	<code>spring-integration-jdbc</code>
JMS	<code>spring-integration-jms</code>
JMX	<code>spring-integration-jmx</code>
JPA	<code>spring-integration-jpa</code>
Kafka	<code>spring-integration-kafka</code>
MongoDB	<code>spring-integration-mongodb</code>
MQTT	<code>spring-integration-mqtt</code>
R2DBC	<code>spring-integration-r2dbc</code>
Redis	<code>spring-integration-redis</code>
RMI	<code>spring-integration-rmi</code>

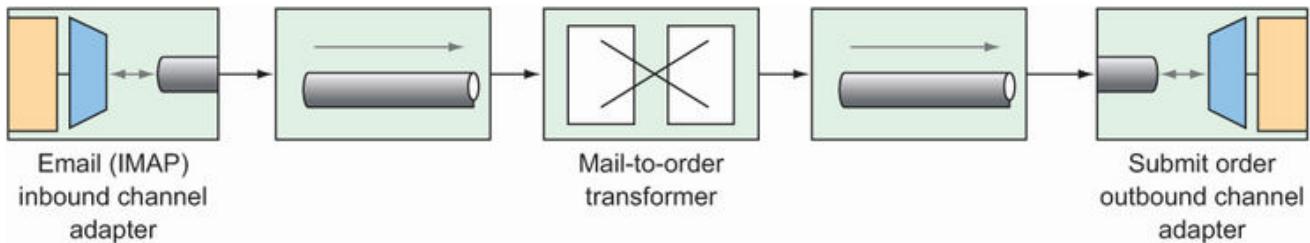
RSocket	spring-integration-rsocket
SFTP	spring-integration-sftp
STOMP	spring-integration-stomp
Stream	spring-integration-stream
Syslog	spring-integration-syslog
TCP/UDP	spring-integration-ip
WebFlux	spring-integration-webflux
Web Services	spring-integration-ws
WebSocket	spring-integration-websocket
XMPP	spring-integration-xmpp
ZeroMQ	spring-integration-zeromq
ZooKeeper	spring-integration-zookeeper

```
1 package tacos.email;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4 import org.springframework.stereotype.Component;
5 import lombok.Data;
6
7 @Data
8 @ConfigurationProperties(prefix="tacocloud.email")
9 @Component
10 public class EmailProperties {
11
12     private String username;
13     private String password;
14     private String host;
15     private String mailbox;
16     private long pollRate = 30000;
17     public String getImapUrl() {
18         return String.format("imaps:// %s:%s@%s/%s",
19             this.username, this.password, this.host, this.mailbox);
20     }
21
22 }
```

```
1 tacocloud:
2   email:
3     host: imap.tacocloud.com
4     mailbox: INBOX
5     username: taco-in-flow
6     password: 1L0v3T4c0s
7     poll-rate: 10000
```

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-configuration-processor</artifactId>
4   <optional>true</optional>
5 </dependency>
```

Figure 10.10 An integration flow to accept taco orders by email



Listing 10.5 Defining an integration flow to accept emails and submit them as orders

```
1 package tacos.email;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.integration.dsl.IntegrationFlow;
6 import org.springframework.integration.dsl.IntegrationFlows;
7 import org.springframework.integration.dsl.Pollers;
8 import org.springframework.integration.mail.dsl.Mail;
9
10 @Configuration
11 public class TacoOrderEmailIntegrationConfig {
12
13     @Bean
14     public IntegrationFlow tacoOrderEmailFlow(
15         EmailProperties emailProps,
16         EmailToOrderTransformer emailToOrderTransformer,
17         OrderSubmitMessageHandler orderSubmitHandler) {
18
19         return IntegrationFlows
20             .from(Mail imapInboundAdapter(emailProps.getImapUrl()),
21                   e -> e.poller()
22                     Pollers.fixedDelay(emailProps.getPollRate()))
23             .transform(emailToOrderTransformer)
24             .handle(orderSubmitHandler)
25             .get();
26     }
27
28 }
```

```
1 <dependency>
2   <groupId>org.springframework.integration</groupId>
3   <artifactId>spring-integration-mail</artifactId>
4 </dependency>
```

Listing 10.6 Converting incoming emails to taco orders using an integration transformer

```
1 package tacos.email;
2
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6 import javax.mail.Message;
7 import javax.mail.MessagingException;
8 import javax.mail.internet.InternetAddress;
9 import org.apache.commons.text.similarity.LevenshteinDistance;
10 import org.slf4j.Logger;
11 import org.slf4j.LoggerFactory;
12 import org.springframework.integration.mail.transformer
13                                     .AbstractMailMessageTransformer;
14 import org.springframework.integration.support
15                                     .AbstractIntegrationMessageBuilder;
16 import org.springframework.integration.support.MessageBuilder;
17 import org.springframework.stereotype.Component;
18
19 @Component
20 public class EmailToOrderTransformer
21     extends AbstractMailMessageTransformer<EmailOrder> {
22     private static Logger log =
23         LoggerFactory.getLogger(EmailToOrderTransformer.class);
24
25     private static final String SUBJECT_KEYWORDS = "TACO ORDER";
26
27     @Override
28     protected AbstractIntegrationMessageBuilder<EmailOrder>
29             doTransform(Message mailMessage) throws Exception {
30         EmailOrder tacoOrder = processPayload(mailMessage);
31         return MessageBuilder.withPayload(tacoOrder);
32     }
33
34     private EmailOrder processPayload(Message mailMessage) {
35         try {
36             String subject = mailMessage.getSubject();
37             if (subject.toUpperCase().contains(SUBJECT_KEYWORDS)) {
38                 String email =
39                     ((InternetAddress) mailMessage.getFrom()[0]).getAddress();
40                 String content = mailMessage.getContent().toString();
41                 return parseEmailToOrder(email, content);
42             }
43         } catch (MessagingException e) {
44             log.error("MessagingException: {}", e);
45         } catch (IOException e) {
46             log.error("IOException: {}", e);
47         }
48         return null;
49     }
50
51     private EmailOrder parseEmailToOrder(String email, String content) {
52         EmailOrder order = new EmailOrder(email);
53         String[] lines = content.split("\r?\n");
54         for (String line : lines) {
55             if (line.trim().length() > 0 && line.contains(":")) {
56                 String[] lineSplit = line.split(":");
57                 String tacoName = lineSplit[0].trim();
58                 String ingredients = lineSplit[1].trim();
59                 String[] ingredientsSplit = ingredients.split(",");
60                 List<String> ingredientCodes = new ArrayList<>();
61                 for (String ingredientName : ingredientsSplit) {
```

```
62     String code = lookupIngredientCode(ingredientName.trim());
63     if (code != null) {
64         ingredientCodes.add(code);
65     }
66 }
67
68     Taco taco = new Taco(tacoName);
69     taco.setIngredients(ingredientCodes);
70     order.addTaco(taco);
71 }
72 }
73 return order;
74 }
75
76 private String lookupIngredientCode(String ingredientName) {
77     for (Ingredient ingredient : ALL_INGREDIENTS) {
78         String ucIngredientName = ingredientName.toUpperCase();
79         if (LevenshteinDistance.getDefaultInstance()
80             .apply(ucIngredientName, ingredient.getName()) < 3 || |
81             ucIngredientName.contains(ingredient.getName()) ||
82             ingredient.getName().contains(ucIngredientName)) {
83             return ingredient.getCode();
84         }
85     }
86     return null;
87 }
88
89 private static Ingredient[] ALL_INGREDIENTS = new Ingredient[] {
90     new Ingredient("FLTO", "FLOUR TORTILLA"),
91     new Ingredient("COTO", "CORN TORTILLA"),
92     new Ingredient("GRBF", "GROUND BEEF"),
93     new Ingredient("CARN", "CARNITAS"),
94     new Ingredient("TMTO", "TOMATOES"),
95     new Ingredient("LETC", "LETTUCE"),
96     new Ingredient("CHED", "CHEDDAR"),
97     new Ingredient("JACK", "MONTERREY JACK"),
98     new Ingredient("SLSA", "SALSA"),
99     new Ingredient("SRCR", "SOUR CREAM")
100 };
101 }
```

```
1 package tacos.email;
2 import java.util.ArrayList;
3 import java.util.List;
4 import lombok.Data;
5
6 @Data
7 public class EmailOrder {
8
9     private final String email;
10    private List<Taco> tacos = new ArrayList<>();
11
12    public void addTaco(Taco taco) {
13        this.tacos.add(taco);
14    }
15
16 }
```

Listing 10.7 Posting orders to the Taco Cloud API via a message handler

```
1 package tacos.email;
2
3 import org.springframework.integration.handler.GenericHandler;
4 import org.springframework.messaging.MessageHeaders;
5 import org.springframework.stereotype.Component;
6 import org.springframework.web.client.RestTemplate;
7
8 @Component
9 public class OrderSubmitMessageHandler
10     implements GenericHandler<EmailOrder> {
11
12     private RestTemplate rest;
13     private ApiProperties apiProps;
14
15     public OrderSubmitMessageHandler(ApiProperties apiProps, RestTemplate rest) {
16         this.apiProps = apiProps;
17         this.rest = rest;
18     }
19
20     @Override
21     public Object handle(EmailOrder order, MessageHeaders headers) {
22         rest.postForObject(apiProps.getUrl(), order, String.class);
23         return null;
24     }
25 }
```

```
1 package tacos.email;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4 import org.springframework.stereotype.Component;
5 import lombok.Data;
6
7 @Data
8 @ConfigurationProperties(prefix = "tacocloud.api")
9 @Component
10 public class ApiProperties {
11     private String url;
12 }
```

```
1 tacocloud:
2   api:
3     url: http://localhost:8080/orders/fromEmail
```

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-web</artifactId>
4 </dependency>
```

```
1 spring:
2   main:
3     web-application-type: none
```

CHAPTER 11

```
public interface Publisher<T> {  
    void subscribe(Subscriber<? super T> subscriber);  
}
```

```
public interface Subscriber<T> {  
    void onSubscribe(Subscription sub);  
    void onNext(T item);  
    void onError(Throwable ex);  
    void onComplete();  
}
```

```
public interface Subscription {  
    void request(long n);  
    void cancel();  
}
```

```
public interface Processor<T, R>  
    extends Subscriber<T>, Publisher<R> {}
```

```
String name = "Craig";  
String capitalName = name.toUpperCase();  
String greeting = "Hello, " + capitalName + "!";  
System.out.println(greeting);
```

```

Mono.just("Craig")
    .map(n -> n.toUpperCase())
    .map(cn -> "Hello, " + cn + "!")
    .subscribe(System.out::println);

```

Figure 11.1 Marble diagram illustrating the basic flow of a Flux

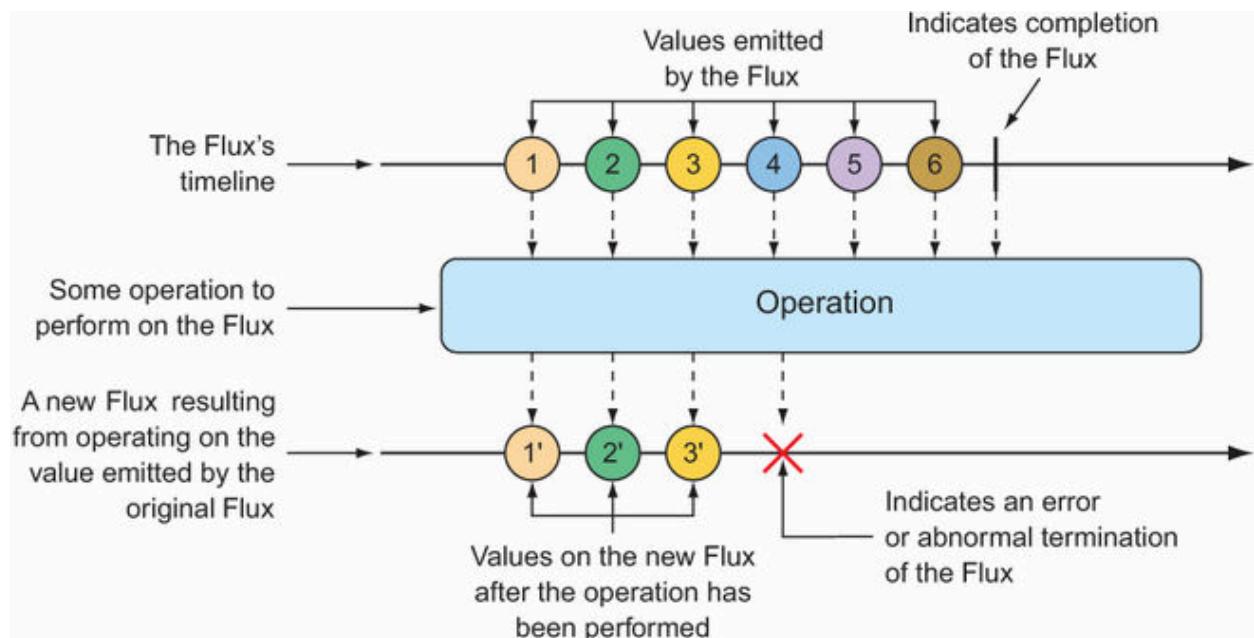
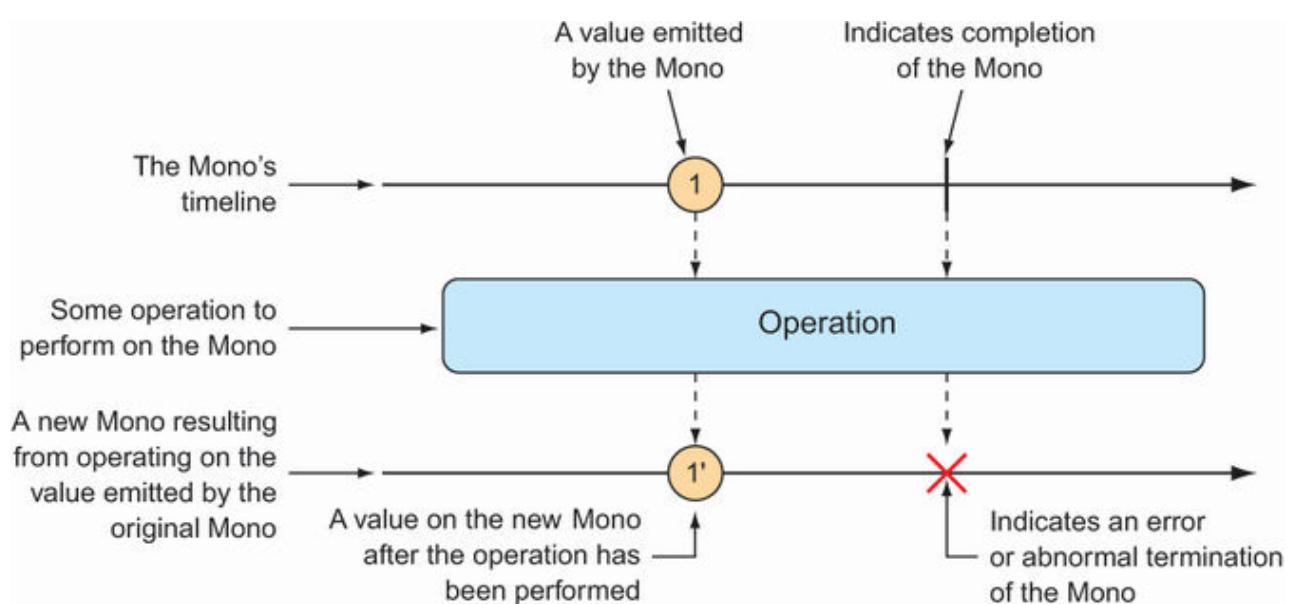


Figure 11.2 Marble diagram illustrating the basic flow of a Mono



```
<dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-core</artifactId>
</dependency>
```

```
<dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-test</artifactId>
    <scope>test</scope>
</dependency>
```

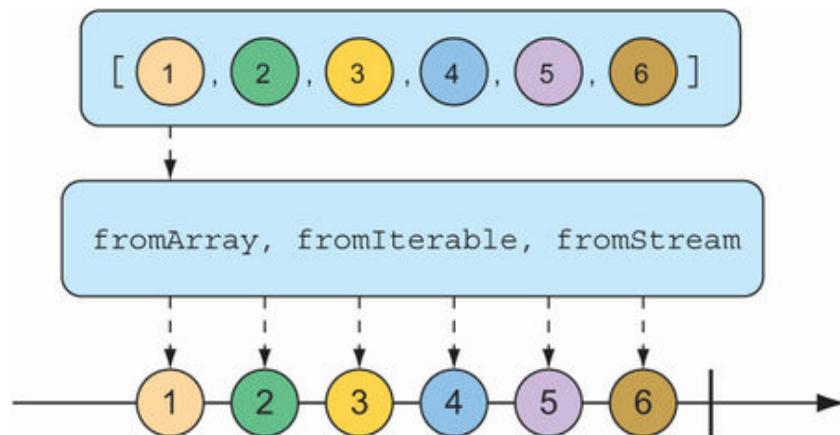
```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.projectreactor</groupId>
            <artifactId>reactor-bom</artifactId>
            <version>2020.0.4</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
@Test
public void createAFlux_just() {
    Flux<String> fruitFlux = Flux
        .just("Apple", "Orange", "Grape", "Banana", "Strawberry");
}
```

```
fruitFlux.subscribe(
    f -> System.out.println("Here's some fruit: " + f)
);
```

```
StepVerifier.create(fruitFlux)
    .expectNext("Apple")
    .expectNext("Orange")
    .expectNext("Grape")
    .expectNext("Banana")
    .expectNext("Strawberry")
    .verifyComplete();
```

Figure 11.3 A **Flux** can be created from an array, **Iterable**, or **Stream**.



```
@Test
public void createAFlux_fromArray() {
    String[] fruits = new String[] {
        "Apple", "Orange", "Grape", "Banana", "Strawberry" };

    Flux<String> fruitFlux = Flux.fromArray(fruits);

    StepVerifier.create(fruitFlux)
        .expectNext("Apple")
        .expectNext("Orange")
        .expectNext("Grape")
        .expectNext("Banana")
        .expectNext("Strawberry")
        .verifyComplete();
}
```

```

@Test
public void createAFlux_fromIterable() {
    List<String> fruitList = new ArrayList<>();
    fruitList.add("Apple");
    fruitList.add("Orange");
    fruitList.add("Grape");
    fruitList.add("Banana");
    fruitList.add("Strawberry");

    Flux<String> fruitFlux = Flux.fromIterable(fruitList);

    StepVerifier.create(fruitFlux)
        .expectNext("Apple")
        .expectNext("Orange")
        .expectNext("Grape")
        .expectNext("Banana")
        .expectNext("Strawberry")
        .verifyComplete();
}

```

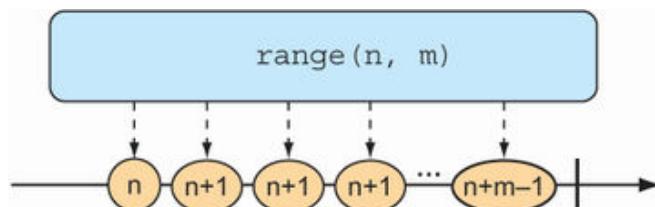
```

@Test
public void createAFlux_fromStream() {
    Stream<String> fruitStream =
        Stream.of("Apple", "Orange", "Grape", "Banana", "Strawberry");
    Flux<String> fruitFlux = Flux.fromStream(fruitStream);

    StepVerifier.create(fruitFlux)
        .expectNext("Apple")
        .expectNext("Orange")
        .expectNext("Grape")
        .expectNext("Banana")
        .expectNext("Strawberry")
        .verifyComplete();
}

```

Figure 11.4 Creating a `Flux` from a range results in a counter-style publishing of messages.



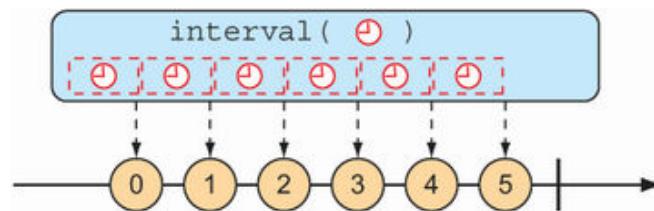
```

@Test
public void createAFlux_range() {
    Flux<Integer> intervalFlux =
        Flux.range(1, 5);

    StepVerifier.create(intervalFlux)
        .expectNext(1)
        .expectNext(2)
        .expectNext(3)
        .expectNext(4)
        .expectNext(5)
        .verifyComplete();
}

```

Figure 11.5 A `Flux` created from an interval has a periodic entry published to it.



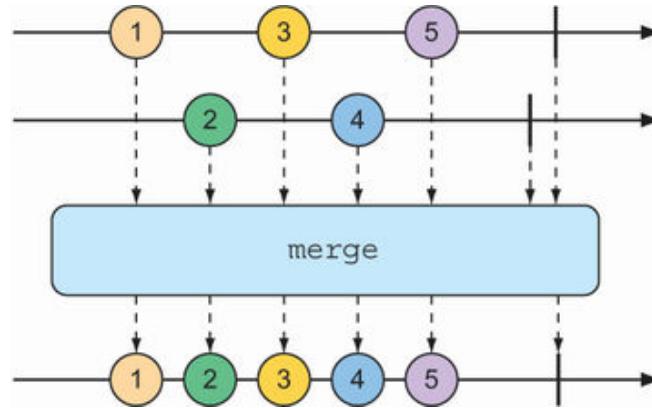
```

@Test
public void createAFlux_interval() {
    Flux<Long> intervalFlux =
        Flux.interval(Duration.ofSeconds(1))
            .take(5);

    StepVerifier.create(intervalFlux)
        .expectNext(0L)
        .expectNext(1L)
        .expectNext(2L)
        .expectNext(3L)
        .expectNext(4L)
        .verifyComplete();
}

```

Figure 11.6 Merging two `Flux` streams interleaves their messages into a new `Flux`.



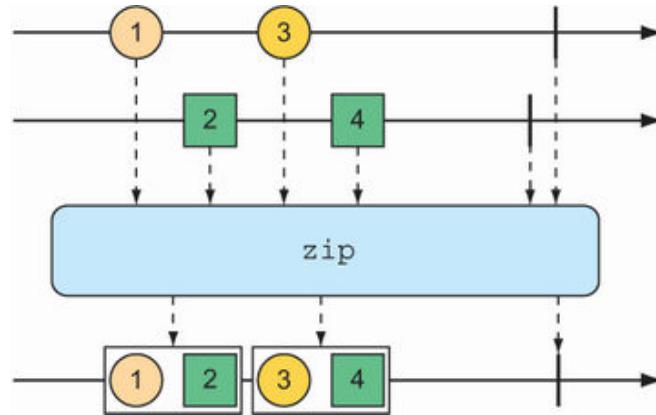
```
@Test
public void mergeFluxes() {

    Flux<String> characterFlux = Flux
        .just("Garfield", "Kojak", "Barbossa")
        .delayElements(Duration.ofMillis(500));
    Flux<String> foodFlux = Flux
        .just("Lasagna", "Lollipops", "Apples")
        .delaySubscription(Duration.ofMillis(250))
        .delayElements(Duration.ofMillis(500));

    Flux<String> mergedFlux = characterFlux.mergeWith(foodFlux);

    StepVerifier.create(mergedFlux)
        .expectNext("Garfield")
        .expectNext("Lasagna")
        .expectNext("Kojak")
        .expectNext("Lollipops")
        .expectNext("Barbossa")
        .expectNext("Apples")
        .verifyComplete();
}
```

Figure 11.7 Zipping two `Flux` streams results in a `Flux` containing tuples of one element from each `Flux`.

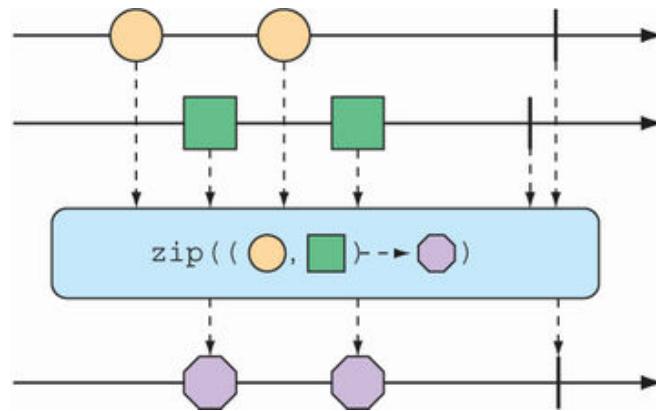


```
@Test
public void zipFluxes() {
    Flux<String> characterFlux = Flux
        .just("Garfield", "Kojak", "Barbossa");
    Flux<String> foodFlux = Flux
        .just("Lasagna", "Lollipops", "Apples");

    Flux<Tuple2<String, String>> zippedFlux =
        Flux.zip(characterFlux, foodFlux);

    StepVerifier.create(zippedFlux)
        .expectNextMatches(p ->
            p.getT1().equals("Garfield") &&
            p.getT2().equals("Lasagna"))
        .expectNextMatches(p ->
            p.getT1().equals("Kojak") &&
            p.getT2().equals("Lollipops"))
        .expectNextMatches(p ->
            p.getT1().equals("Barbossa") &&
            p.getT2().equals("Apples"))
        .verifyComplete();
}
```

Figure 11.8 An alternative form of the `zip()` operation results in a `Flux` of messages created from one element of each incoming `Flux`.

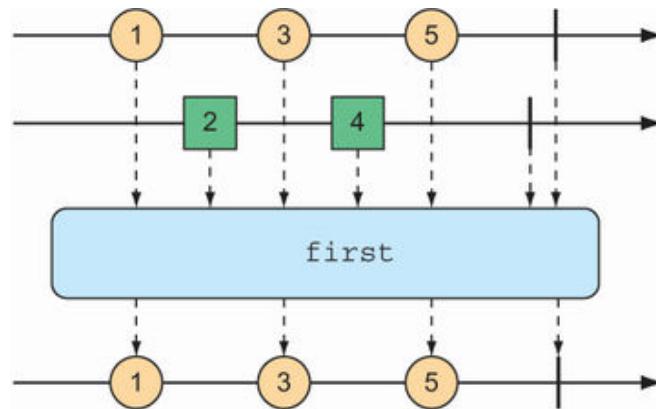


```
@Test
public void zipFluxesToObject() {
    Flux<String> characterFlux = Flux
        .just("Garfield", "Kojak", "Barbossa");
    Flux<String> foodFlux = Flux
        .just("Lasagna", "Lollipops", "Apples");

    Flux<String> zippedFlux =
        Flux.zip(characterFlux, foodFlux, (c, f) -> c + " eats " + f);

    StepVerifier.create(zippedFlux)
        .expectNext("Garfield eats Lasagna")
        .expectNext("Kojak eats Lollipops")
        .expectNext("Barbossa eats Apples")
        .verifyComplete();
}
```

Figure 11.9 The `first()` operation chooses the first `Flux` to emit a message and thereafter produces messages only from that `Flux`.



```

@Test
public void firstWithSignalFlux() {

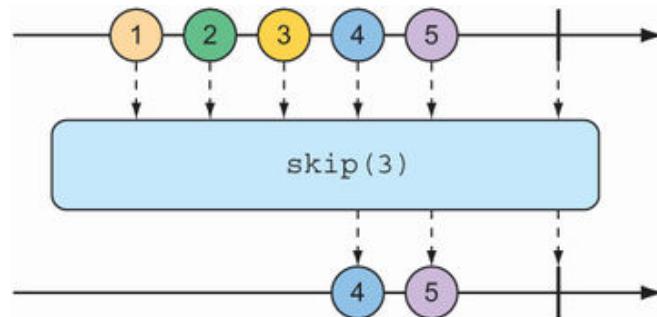
    Flux<String> slowFlux = Flux.just("tortoise", "snail", "sloth")
        .delaySubscription(Duration.ofMillis(100));
    Flux<String> fastFlux = Flux.just("hare", "cheetah", "squirrel");

    Flux<String> firstFlux = Flux.firstWithSignal(slowFlux, fastFlux);

    StepVerifier.create(firstFlux)
        .expectNext("hare")
        .expectNext("cheetah")
        .expectNext("squirrel")
        .verifyComplete();
}

```

Figure 11.10 The `skip()` operation skips a specified number of messages before passing the remaining messages on to the resulting `Flux`.



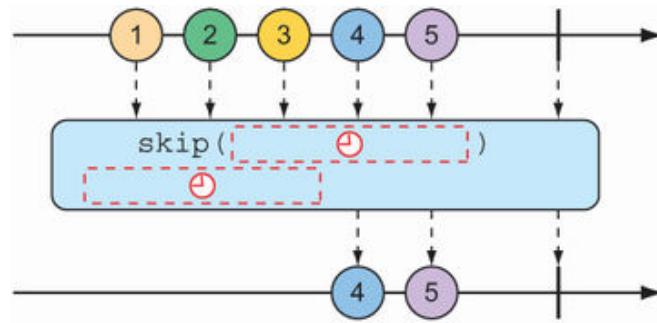
```

@Test
public void skipAFew() {
    Flux<String> countFlux = Flux.just(
        "one", "two", "skip a few", "ninety nine", "one hundred")
        .skip(3);

    StepVerifier.create(countFlux)
        .expectNext("ninety nine", "one hundred")
        .verifyComplete();
}

```

Figure 11.11 An alternative form of the `skip()` operation waits until some duration has passed before passing messages on to the resulting `Flux`.



```
@Test
public void skipAFewSeconds() {
    Flux<String> countFlux = Flux.just(
        "one", "two", "skip a few", "ninety nine", "one hundred")
        .delayElements(Duration.ofSeconds(1))
        .skip(Duration.ofSeconds(4));

    StepVerifier.create(countFlux)
        .expectNext("ninety nine", "one hundred")
        .verifyComplete();
}
```

```
@Test
public void take() {
    Flux<String> nationalParkFlux = Flux.just(
        "Yellowstone", "Yosemite", "Grand Canyon", "Zion", "Acadia")
        .take(3);

    StepVerifier.create(nationalParkFlux)
        .expectNext("Yellowstone", "Yosemite", "Grand Canyon")
        .verifyComplete();
}
```

Figure 11.12 The `take()` operation passes only the first so many messages from the incoming `Flux` and then cancels the subscription.

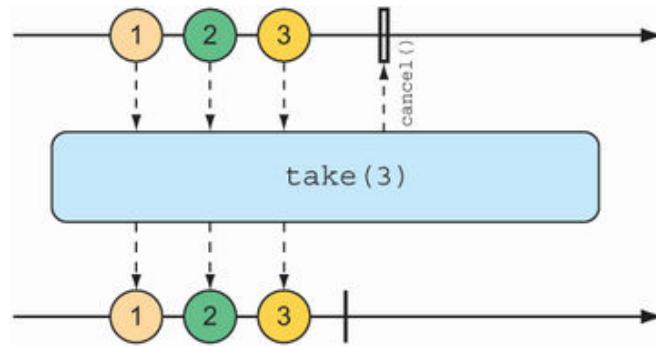
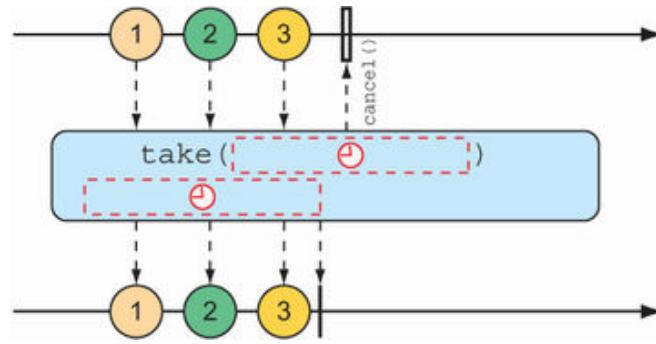


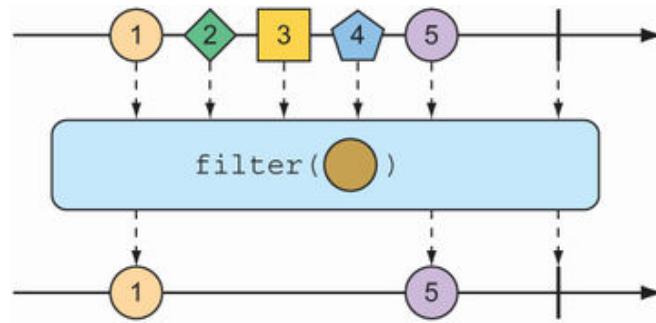
Figure 11.13 An alternative form of the `take()` operation passes messages on to the resulting `Flux` until some duration has passed.



```
@Test
public void takeForAwhile() {
    Flux<String> nationalParkFlux = Flux.just(
        "Yellowstone", "Yosemite", "Grand Canyon", "Zion", "Grand Teton")
        .delayElements(Duration.ofSeconds(1))
        .take(Duration.ofMillis(3500));

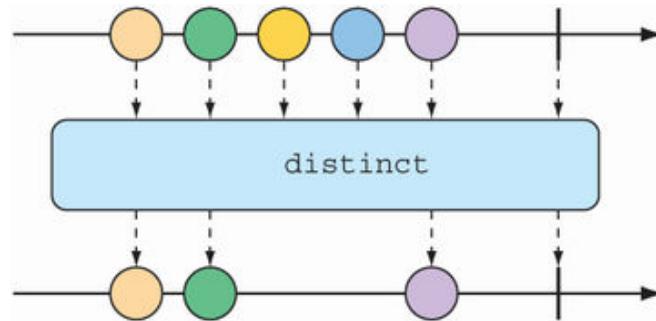
    StepVerifier.create(nationalParkFlux)
        .expectNext("Yellowstone", "Yosemite", "Grand Canyon")
        .verifyComplete();
}
```

Figure 11.14 An incoming `Flux` can be filtered so that the resulting `Flux` receives only messages that match a given predicate.



```
@Test+
public void filter() {
    Flux<String> nationalParkFlux = Flux.just(
        "Yellowstone", "Yosemite", "Grand Canyon", "Zion", "Grand Teton")
        .filter(np -> !np.contains(" "));
    StepVerifier.create(nationalParkFlux)
        .expectNext("Yellowstone", "Yosemite", "Zion")
        .verifyComplete();
}
```

Figure 11.15 The `distinct()` operation filters out any duplicate messages.



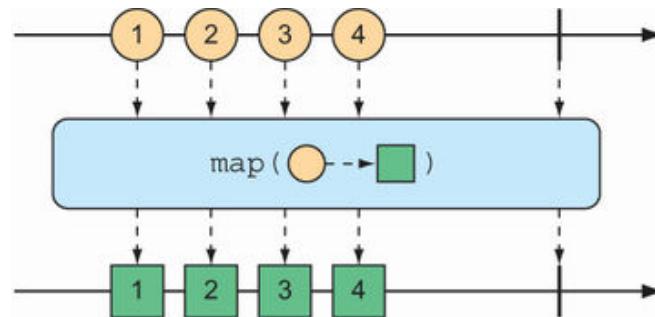
```

@Test
public void distinct() {
    Flux<String> animalFlux = Flux.just(
        "dog", "cat", "bird", "dog", "bird", "anteater")
        .distinct();

    StepVerifier.create(animalFlux)
        .expectNext("dog", "cat", "bird", "anteater")
        .verifyComplete();
}

```

Figure 11.16 The `map()` operation performs a transformation of incoming messages into new messages on the resulting stream.



```

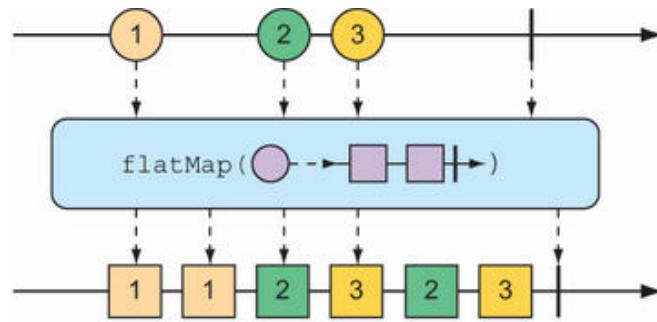
@Test
public void map() {
    Flux<Player> playerFlux = Flux
        .just("Michael Jordan", "Scottie Pippen", "Steve Kerr")
        .map(n -> {
            String[] split = n.split("\\s");
            return new Player(split[0], split[1]);
        });

    StepVerifier.create(playerFlux)
        .expectNext(new Player("Michael", "Jordan"))
        .expectNext(new Player("Scottie", "Pippen"))
        .expectNext(new Player("Steve", "Kerr"))
        .verifyComplete();
}

@Data
private static class Player {
    private final String firstName;
    private final String lastName;
}

```

Figure 11.17 The `flatMap()` operation uses an intermediate `Flux` to perform a transformation, consequently allowing for asynchronous transformations.



```
@Test
public void flatMap() {
    Flux<Player> playerFlux = Flux
        .just("Michael Jordan", "Scottie Pippen", "Steve Kerr")
        .flatMap(n -> Mono.just(n)
            .map(p -> {
                String[] split = p.split("\\s");
                return new Player(split[0], split[1]);
            })
            .subscribeOn(Schedulers.parallel())
        );
}

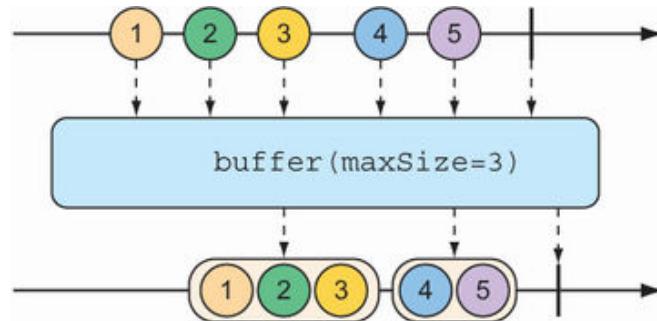
List<Player> playerList = Arrays.asList(
    new Player("Michael", "Jordan"),
    new Player("Scottie", "Pippen"),
    new Player("Steve", "Kerr"));

StepVerifier.create(playerFlux)
    .expectNextMatches(p -> playerList.contains(p))
    .expectNextMatches(p -> playerList.contains(p))
    .expectNextMatches(p -> playerList.contains(p))
    .verifyComplete();
}
```

Table 11.1 Concurrency models for **Schedulers** ([view table figure](#))

Schedulers method	Description
<code>.immediate()</code>	Executes the subscription in the current thread.
<code>.single()</code>	Executes the subscription in a single, reusable thread. Reuses the same thread for all callers.
<code>.newSingle()</code>	Executes the subscription in a per-call dedicated thread.
<code>.elastic()</code>	Executes the subscription in a worker pulled from an unbounded, elastic pool. New worker threads are created as needed, and idle workers are disposed of (by default, after 60 seconds).
<code>.parallel()</code>	Executes the subscription in a worker pulled from a fixed-size pool, sized to the number of CPU cores.

Figure 11.18 The `buffer()` operation results in a **Flux** of lists of a given maximum size that are collected from the incoming **Flux**.



```
@Test
public void buffer() {
    Flux<String> fruitFlux = Flux.just(
        "apple", "orange", "banana", "kiwi", "strawberry");

    Flux<List<String>> bufferedFlux = fruitFlux.buffer(3);

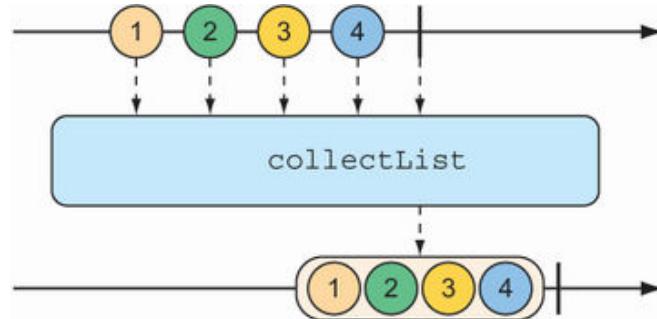
    StepVerifier
        .create(bufferedFlux)
        .expectNext(NSArray.asList("apple", "orange", "banana"))
        .expectNext(NSArray.asList("kiwi", "strawberry"))
        .verifyComplete();
}
```

```
@Test
public void bufferAndflatMap() throws Exception {
    Flux.just(
        "apple", "orange", "banana", "kiwi", "strawberry")
        .buffer(3)
        .flatMap(x ->
            Flux.fromIterable(x)
                .map(y -> y.toUpperCase())
                .subscribeOn(Schedulers.parallel())
                .log()
        ).subscribe();
}
```

```
[main] INFO reactor.Flux.SubscribeOn.1 -
    onSubscribe(FluxSubscribeOn.SubscribeOnSubscriber)
[main] INFO reactor.Flux.SubscribeOn.1 - request(32)
[main] INFO reactor.Flux.SubscribeOn.2 -
    onSubscribe(FluxSubscribeOn.SubscribeOnSubscriber)
[main] INFO reactor.Flux.SubscribeOn.2 - request(32)
[parallel-1] INFO reactor.Flux.SubscribeOn.1 - onNext(APPLE)
[parallel-2] INFO reactor.Flux.SubscribeOn.2 - onNext(KIWI)
[parallel-1] INFO reactor.Flux.SubscribeOn.1 - onNext(ORANGE)
[parallel-2] INFO reactor.Flux.SubscribeOn.2 - onNext(STRAWBERRY)
[parallel-1] INFO reactor.Flux.SubscribeOn.1 - onNext(BANANA)
[parallel-1] INFO reactor.Flux.SubscribeOn.1 - onComplete()
[parallel-2] INFO reactor.Flux.SubscribeOn.2 - onComplete()
```

```
Flux<List<String>> bufferedFlux = fruitFlux.buffer();
```

Figure 11.19 The `collectList()` operation results in a `Mono` containing a list of all messages emitted by the incoming `Flux`.

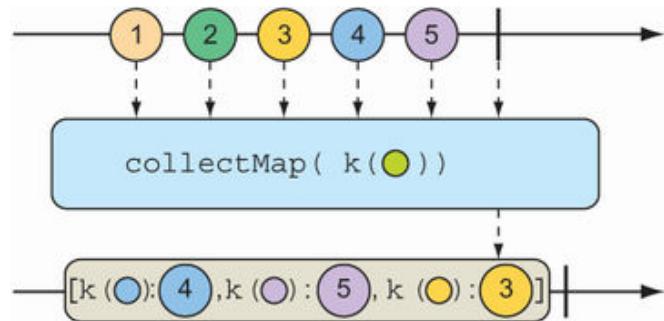


```
@Test
public void collectList() {
    Flux<String> fruitFlux = Flux.just(
        "apple", "orange", "banana", "kiwi", "strawberry");

    Mono<List<String>> fruitListMono = fruitFlux.collectList();

    StepVerifier
        .create(fruitListMono)
        .expectNext(NSArray.asList(
            "apple", "orange", "banana", "kiwi", "strawberry"))
        .verifyComplete();
}
```

Figure 11.20 The `collectMap()` operation results in a `Mono` containing a map of messages emitted by the incoming `Flux`, where the key is derived from some characteristic of the incoming messages.



```
@Test
public void collectMap() {
    Flux<String> animalFlux = Flux.just(
        "aardvark", "elephant", "koala", "eagle", "kangaroo");

    Mono<Map<Character, String>> animalMapMono =
        animalFlux.collectMap(a -> a.charAt(0));

    StepVerifier
        .create(animalMapMono)
        .expectNextMatches(map -> {
            return
                map.size() == 3 &&
                map.get('a').equals("aardvark") &&
                map.get('e').equals("eagle") &&
                map.get('k').equals("kangaroo");
        })
        .verifyComplete();
}
```

Figure 11.21 A `Flux` can be tested to ensure that all messages meet some condition with the `all()` operation.

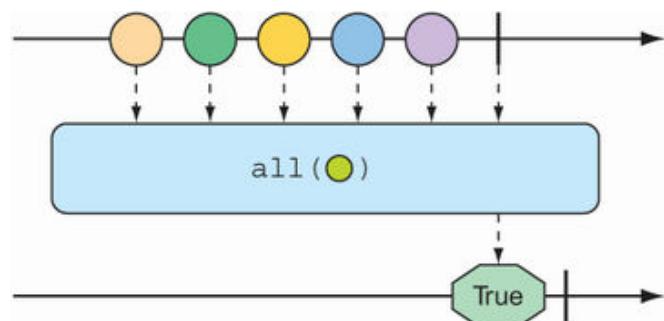
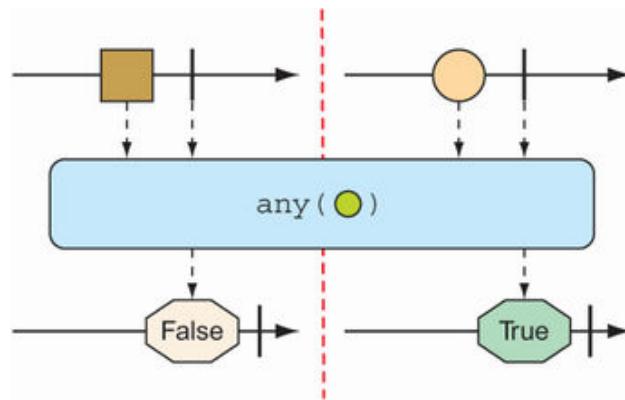


Figure 11.22 A `Flux` can be tested to ensure that at least one message meets some condition with the `any()` operation.



```
@Test
public void all() {
    Flux<String> animalFlux = Flux.just(
        "aardvark", "elephant", "koala", "eagle", "kangaroo");

    Mono<Boolean> hasAMono = animalFlux.all(a -> a.contains("a"));
    StepVerifier.create(hasAMono)
        .expectNext(true)
        .verifyComplete();

    Mono<Boolean> hasKMono = animalFlux.all(a -> a.contains("k"));
    StepVerifier.create(hasKMono)
        .expectNext(false)
        .verifyComplete();
}
```

```
@Test
public void any() {
    Flux<String> animalFlux = Flux.just(
        "aardvark", "elephant", "koala", "eagle", "kangaroo");

    Mono<Boolean> hasTMono = animalFlux.any(a -> a.contains("t"));

    StepVerifier.create(hasTMono)
        .expectNext(true)
        .verifyComplete();

    Mono<Boolean> hasZMono = animalFlux.any(a -> a.contains("z"));
    StepVerifier.create(hasZMono)
        .expectNext(false)
        .verifyComplete();
}
```

CHAPTER 12

Figure 12.1 Asynchronous web frameworks apply event looping to handle more requests with fewer threads.

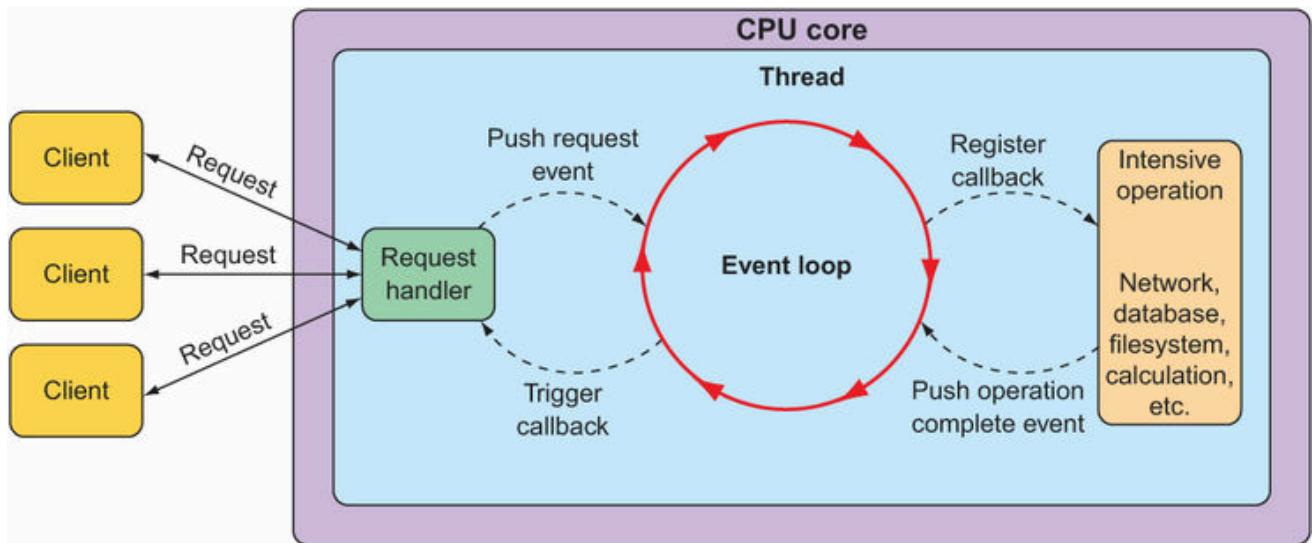
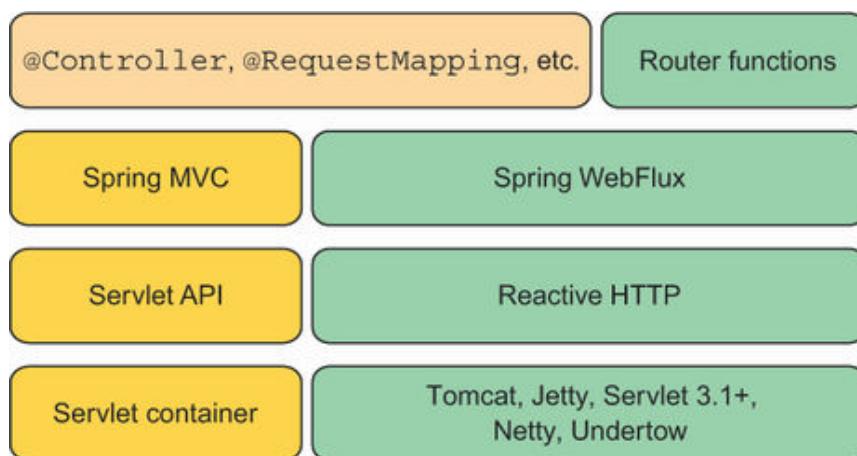


Figure 12.2 Spring supports reactive web applications with a new web framework called WebFlux, which is a sibling to Spring MVC and shares many of its core components.



```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-webflux</artifactId>
4 </dependency>
```

```
@RestController
@RequestMapping(path="/api/tacos",
    produces="application/json")
@CrossOrigin(origins="*")
public class TacoController {

    ...

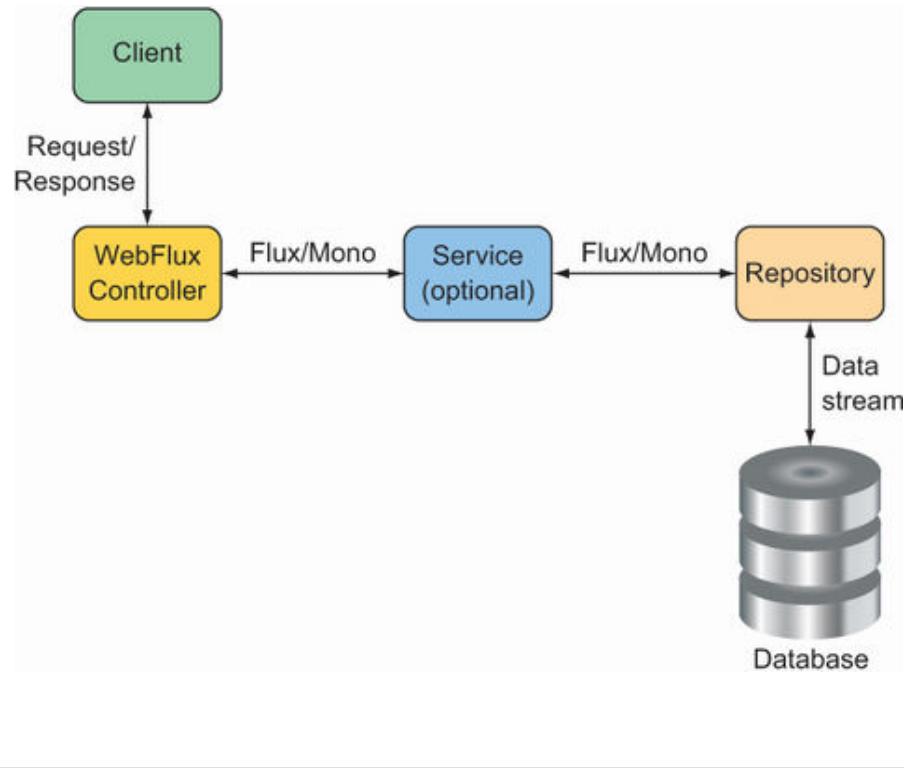
    @GetMapping(params="recent")
    public Iterable<Taco> recentTacos() {
        PageRequest page = PageRequest.of(
            0, 12, Sort.by("createdAt").descending());
        return tacoRepo.findAll(page).getContent();
    }

    ...
}
```

```
@GetMapping(params="recent")
public Flux<Taco> recentTacos() {
    return Flux.fromIterable(tacoRepo.findAll()).take(12);
}
```

```
@GetMapping(params="recent")
public Flux<Taco> recentTacos() {
    return tacoRepo.findAll().take(12);
}
```

Figure 12.3 To maximize the benefit of a reactive web framework, it should be part of a full end-to-end reactive stack.



```
1 package tacos.data;
2
3 import org.springframework.data.repository.reactive.ReactiveCrudRepository;
4 import tacos.Taco;
5
6 public interface TacoRepository
7     extends ReactiveCrudRepository<Taco, Long> {
8 }
```

```
1 @GetMapping("/{id}")
2 public Taco tacoById(@PathVariable("id") Long id) {
3     Optional<Taco> optTaco = tacoRepo.findById(id);
4     if (optTaco.isPresent()) {
5         return optTaco.get();
6     }
7     return null;
8 }
```

```
1 @GetMapping("/{id}")
2 public Mono<Taco> tacoById(@PathVariable("id") Long id) {
3     return tacoRepo.findById(id);
4 }
```

```
1 @GetMapping(params = "recent")
2 public Observable<Taco> recentTacos() {
3     return tacoService.getRecentTacos();
4 }
```

```
1 @GetMapping("/{id}")
2 public Single<Taco> tacoById(@PathVariable("id") Long id) {
3     return tacoService.lookupTaco(id);
4 }
```

```
1 @PostMapping(consumes="application/json")
2 @ResponseStatus(HttpStatus.CREATED)
3 public Taco postTaco(@RequestBody Taco taco) {
4     return tacoRepo.save(taco);
5 }
```

```
1 @PostMapping(consumes = "application/json")
2 @ResponseStatus(HttpStatus.CREATED)
3 public Mono<Taco> postTaco(@RequestBody Mono<Taco> tacoMono) {
4     return tacoRepo.saveAll(tacoMono).next();
5 }
```

```
1 @PostMapping(consumes = "application/json")
2 @ResponseStatus(HttpStatus.CREATED)
3 public Mono<Taco> postTaco(@RequestBody Mono<Taco> tacoMono) {
4     return tacoMono.flatMap(tacoRepo::save);
5 }
```

```
1 package hello;
2
3 import static org.springframework.web
4         .reactive.function.server.RequestPredicates.GET;
5 import static org.springframework.web
6         .reactive.function.server.RouterFunctions.route;
7 import static org.springframework.web
8         .reactive.function.server.ServerResponse.ok;
9 import static reactor.core.publisher.Mono.just;
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.context.annotation.Configuration;
12 import org.springframework.web.reactive.function.server.RouterFunction;
13
14 @Configuration
15 public class RouterFunctionConfig {
16
17     @Bean
18     public RouterFunction<?> helloRouterFunction() {
19         return route(GET("/hello"),
20                     request -> ok().body(just("Hello World!"), String.class))
21                 ;
22     }
23
24 }
```

```
1 @Bean
2 public RouterFunction<?> helloRouterFunction() {
3     return route(GET("/hello"),
4                 request -> ok().body(just("Hello World!"), String.class))
5                 .andRoute(GET("/bye"),
6                 request -> ok().body(just("See ya!"), String.class))
7             ;
8 }
```

```
1 package tacos.web.api;
2
3 import static org.springframework.web.reactive.function.server
4 .RequestPredicates.GET;
5 import static org.springframework.web.reactive.function.server
6 .RequestPredicates.POST;
7 import static org.springframework.web.reactive.function.server
8 .RequestPredicates.queryParam;
9 import static org.springframework.web.reactive.function.server
10 .RouterFunctions.route;
11
12 import java.net.URI;
13
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.context.annotation.Bean;
16 import org.springframework.context.annotation.Configuration;
17 import org.springframework.web.reactive.function.server.RouterFunction;
18 import org.springframework.web.reactive.function.server.ServerRequest;
19 import org.springframework.web.reactive.function.server.ServerResponse;
20
21 import reactor.core.publisher.Mono;
22 import tacos.Taco;
23 import tacos.data.TacoRepository;
24 @Configuration
25 public class RouterFunctionConfig {
26
27     @Autowired
28     private TacoRepository tacoRepo;
29
30     @Bean
31     public RouterFunction<?> routerFunction() {
32         return route(GET("/api/tacos").
33                     and(queryParam("recent", t->t != null )),
34                     this::recents)
35                     .andRoute(POST("/api/tacos"), this::postTaco);
36     }
37
38     public Mono<ServerResponse> recents(ServerRequest request) {
39         return ServerResponse.ok()
40             .body(tacoRepo.findAll().take(12), Taco.class);
41     }
42
43     public Mono<ServerResponse> postTaco(ServerRequest request) {
44         return request.bodyToMono(Taco.class)
45             .flatMap(taco -> tacoRepo.save(taco))
46             .flatMap(savedTaco -> {
47                 return ServerResponse
48                     .created(URI.create(
49                         "http://localhost:8080/api/tacos/" +
50                         savedTaco.getId()))
51                     .body(savedTaco, Taco.class);
52             });
53     }
54 }
```

Listing 12.1 Using `WebTestClient` to test `TacoController`

```
1 package tacos.web.api;
2 import static org.mockito.ArgumentMatchers.any;
3 import static org.mockito.Mockito.when;
4 import java.util.ArrayList;
5 import java.util.List;
6 import org.junit.jupiter.api.Test;
7 import org.mockito.Mockito;
8 import org.springframework.http.MediaType;
9 import org.springframework.test.web.reactive.server.WebTestClient;
10 import reactor.core.publisher(Flux);
11 import reactor.core.publisher.Mono;
12 import tacos.Ingredient;
13 import tacos.Ingredient.Type;
14 import tacos.Taco;
15 import tacos.data.TacoRepository;
16
17 public class TacoControllerTest {
18
19     @Test
20     public void shouldReturnRecentTacos() {
21         Taco[] tacos = {
22             testTaco(1L), testTaco(2L),
23             testTaco(3L), testTaco(4L),
24             testTaco(5L), testTaco(6L),
25             testTaco(7L), testTaco(8L),
26             testTaco(9L), testTaco(10L),
27             testTaco(11L), testTaco(12L),
28             testTaco(13L), testTaco(14L),
29             testTaco(15L), testTaco(16L)};
30         Flux<Taco> tacoFlux = Flux.just(tacos);
31
32         TacoRepository tacoRepo = Mockito.mock(TacoRepository.class);
33         when(tacoRepo.findAll()).thenReturn(tacoFlux);
34
35         WebTestClient testClient = WebTestClient.bindToController(
36             new TacoController(tacoRepo))
37             .build();
38
39         testClient.get().uri("/api/tacos?recent")
40             .exchange()
41             .expectStatus().isOk()
42             .expectBody()
43                 .jsonPath("$.id").isArray()
44                 .jsonPath("$.id").isNotEmpty()
45                 .jsonPath("$.id").isEqualTo(tacos[0].getId().toString())
46                 .jsonPath("$.name").isEqualTo("Taco 1")
47                 .jsonPath("$.id").isEqualTo(tacos[1].getId().toString())
48                 .jsonPath("$.name").isEqualTo("Taco 2")
49                 .jsonPath("$.id").isEqualTo(tacos[11].getId().toString())
50                 .jsonPath("$.name").isEqualTo("Taco 12")
51                 .jsonPath("$.id").doesNotExist();
52     }
53
54     ...
55
56 }
```

1
2
3
4
5

```
1 private Taco testTaco(Long number) {
2     Taco taco = new Taco();
3     taco.setId(number != null ? number.toString() : "TESTID");
4     taco.setName("Taco " + number);
5     List<Ingredient> ingredients = new ArrayList<>();
6     ingredients.add(
7         new Ingredient("INGA", "Ingredient A", Type.WRAP));
8     ingredients.add(
9         new Ingredient("INGB", "Ingredient B", Type.PROTEIN));
10    taco.setIngredients(ingredients);
11    return taco;
12 }
```

```
1 ClassPathResource recentsResource =
2     new ClassPathResource("/tacos/recent-tacos.json");
3 String recentsJson = StreamUtils.copyToString(
4     recentsResource.getInputStream(), Charset.defaultCharset());
5
6 testClient.get().uri("/api/tacos?recent")
7     .accept(MediaType.APPLICATION_JSON)
8     .exchange()
9     .expectStatus().isOk()
10    .expectBody()
11        .json(recentsJson);
```

```
1 testClient.get().uri("/api/tacos?recent")
2     .accept(MediaType.APPLICATION_JSON)
3     .exchange()
4     .expectStatus().isOk()
5     .expectBodyList(Taco.class)
6         .contains(Arrays.copyOf(tacos, 12));
```

Table 12.1 `WebTestClient` tests any kind of request against Spring WebFlux controllers. ([view table figure](#))

HTTP method	<code>WebTestClient</code> method
GET	<code>.get()</code>
POST	<code>.post()</code>
PUT	<code>.put()</code>
PATCH	<code>.patch()</code>
DELETE	<code>.delete()</code>
HEAD	<code>.head()</code>

```

1  @SuppressWarnings("unchecked")
2  @Test
3  public void shouldSaveATaco() {
4      TacoRepository tacoRepo = Mockito.mock(
5          TacoRepository.class);
6
7      WebTestClient testClient = WebTestClient.bindToController(
8          new TacoController(tacoRepo)).build();
9
10     Mono<Taco> unsavedTacoMono = Mono.just(testTaco(1L));
11     Taco savedTaco = testTaco(1L);
12     Flux<Taco> savedTacoMono = Flux.just(savedTaco);
13
14     when(tacoRepo.saveAll(any(Mono.class))).thenReturn(savedTacoMono); 3
15
16
17     testClient.post()
18         .uri("/api/tacos")
19         .contentType(MediaType.APPLICATION_JSON)
20         .body(unsavedTacoMono, Taco.class)
21         .exchange()
22         .expectStatus().isCreated()
23         .expectBody(Taco.class)
24         .isEqualTo(savedTaco);
25 } 4

```

```
1 package tacos;
2
3 import java.io.IOException;
4 import org.junit.jupiter.api.Test;
5 import org.junit.jupiter.api.extension.ExtendWith;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
9 import org.springframework.http.MediaType;
10 import org.springframework.test.context.junit.jupiter.SpringExtension;
11 import org.springframework.test.web.reactive.server.WebTestClient;
12
13 @ExtendWith(SpringExtension.class)
14@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
15 public class TacoControllerWebTest {
16     @Autowired
17     private WebTestClient testClient;
18
19 }
```

```
1 @Test
2 public void shouldReturnRecentTacos() throws IOException {
3     testClient.get().uri("/api/tacos?recent")
4         .accept(MediaType.APPLICATION_JSON).exchange()
5         .expectStatus().isOk()
6         .expectBody()
7             .jsonPath("$.isArray()")
8             .jsonPath("$.length()").isEqualTo(3)
9             .jsonPath("$.?[(@.name == 'Carnivore')]").exists()
10            .jsonPath("$.?[(@.name == 'Bovine Bounty')]").exists()
11            .jsonPath("$.?[(@.name == 'Veg-Out')]").exists();
12 }
```

```
1 Mono<Ingredient> ingredient = WebClient.create()
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .retrieve()
5     .bodyToMono(Ingredient.class);
6
7 ingredient.subscribe(i -> { ... });
```

```
1 Flux<Ingredient> ingredients = WebClient.create()  
2     .get()  
3     .uri("http://localhost:8080/ingredients")  
4     .retrieve()  
5     .bodyToFlux(Ingredient.class);  
6  
7 ingredients.subscribe(i -> { ... });
```

```
1 @Bean  
2 public WebClient webClient() {  
3     return WebClient.create("http://localhost:8080");  
4 }
```

```
1 @Autowired  
2 WebClient webClient;  
3  
4 public Mono<Ingredient> getIngredientById(String ingredientId) {  
5     Mono<Ingredient> ingredient = webClient  
6         .get()  
7         .uri("/ingredients/{id}", ingredientId)  
8         .retrieve()  
9         .bodyToMono(Ingredient.class);  
10  
11     ingredient.subscribe(i -> { ... });  
12 }
```

```
1 Flux<Ingredient> ingredients = webclient  
2     .get()  
3     .uri("/ingredients")  
4     .retrieve()  
5     .bodyToFlux(Ingredient.class);  
6  
7 ingredients  
8     .timeout(Duration.ofSeconds(1))  
9     .subscribe(  
10        i -> { ... },  
11        e -> {  
12            // handle timeout error  
13        });

```

```
1 Mono<Ingredient> ingredientMono = Mono.just(  
2     new Ingredient("INGC", "Ingredient C", Ingredient.Type.VEGGIES));  
3  
4 Mono<Ingredient> result = webClient  
5     .post()  
6     .uri("/ingredients")  
7     .body(ingredientMono, Ingredient.class)  
8     .retrieve()  
9     .bodyToMono(Ingredient.class);  
10  
11 result.subscribe(i -> { ... });
```

```
1 Ingredient ingredient = ...;  
2  
3 Mono<Ingredient> result = webClient  
4     .post()  
5     .uri("/ingredients")  
6     .bodyValue(ingredient)  
7     .retrieve()  
8     .bodyToMono(Ingredient.class);  
9  
10 result.subscribe(i -> { ... });
```

```
1 Mono<Void> result = webClient  
2     .put()  
3     .uri("/ingredients/{id}", ingredient.getId())  
4     .bodyValue(ingredient)  
5     .retrieve()  
6     .bodyToMono(Void.class);  
7  
8 result.subscribe();
```

```
1 Mono<Void> result = webClient  
2     .delete()  
3     .uri("/ingredients/{id}", ingredientId)  
4     .retrieve()  
5     .bodyToMono(Void.class);  
6  
7 result.subscribe();
```

```
1 Mono<Ingredient> ingredientMono = webClient
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .retrieve()
5     .bodyToMono(Ingredient.class);
```

```
1 ingredientMono.subscribe(
2     ingredient -> {
3         // handle the ingredient data
4         ...
5     },
6     error-> {
7         // deal with the error
8         ...
9 });
```

```
1 Mono<Ingredient> ingredientMono = webClient
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .retrieve()
5     .onStatus(HttpStatus::is4xxClientError,
6             response -> Mono.just(new UnknownIngredientException()))
7     .bodyToMono(Ingredient.class);
```

```
1 Mono<Ingredient> ingredientMono = webClient
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .retrieve()
5     .onStatus(status -> status == HttpStatus.NOT_FOUND,
6             response -> Mono.just(new UnknownIngredientException()))
7     .bodyToMono(Ingredient.class);
```

```
1 Mono<Ingredient> ingredientMono = webClient
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .exchangeToMono(cr -> cr.bodyToMono(Ingredient.class));
```

```
1 Mono<Ingredient> ingredientMono = webClient
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .retrieve()
5     .bodyToMono(Ingredient.class);
```

```
1 Mono<Ingredient> ingredientMono = webClient
2     .get()
3     .uri("http://localhost:8080/ingredients/{id}", ingredientId)
4     .exchangeToMono(cr -> {
5         if (cr.headers().header("X_UNAVAILABLE").contains("true")) {
6             return Mono.empty();
7         }
8         return Mono.just(cr);
9     })
10    .flatMap(cr -> cr.bodyToMono(Ingredient.class));
```

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-security</artifactId>
4 </dependency>
```

```
1 @Configuration
2 @EnableWebSecurity
3 public class SecurityConfig extends WebSecurityConfigurerAdapter {
4
5     @Override
6     protected void configure(HttpSecurity http) throws Exception {
7         http
8             .authorizeRequests()
9                 .antMatchers("/api/tacos", "/orders").hasAuthority("USER")
10                .antMatchers("/**").permitAll();
11    }
12
13 }
```

Listing 12.2 Configuring Spring Security for a Spring WebFlux application

```
1 @Configuration
2 @EnableWebFluxSecurity
3 public class SecurityConfig {
4
5     @Bean
6     public SecurityWebFilterChain securityWebFilterChain(
7                                     ServerHttpSecurity http) {
8
9         return http
10            .authorizeExchange()
11                .pathMatchers("/api/tacos", "/orders").hasAuthority("USER")
12                .anyExchange().permitAll()
13            .and()
14            .build();
15
16 }
```

```
1 @Autowired
2 UserRepository userRepo;
3
4 @Override
5 protected void
6     configure(AuthenticationManagerBuilder auth)
7     throws Exception {
8     auth
9         .userDetailsService(new UserDetailsService() {
10             @Override
11             public UserDetails loadUserByUsername(String username)
12                     throws UsernameNotFoundException {
13                 User user = userRepo.findByUsername(username)
14                 if (user == null) {
15                     throw new UsernameNotFoundException(
16                         username + " not found")
17                 }
18                 return user.toUserDetails();
19             }
20         });
21 }
```

```
1 @Bean
2 public ReactiveUserDetailsService userDetailsService(
3                                     UserRepository userRepo) {
4     return new ReactiveUserDetailsService() {
5         @Override
6         public Mono<UserDetails> findByUsername(String username) {
7             return userRepo.findByUsername(username)
8                 .map(user -> {
9                     return user.toUserDetails();
10                });
11        }
12    };
13 }
```

CHAPTER 13

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-r2dbc</artifactId>
</dependency>
```

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.r2dbc</groupId>
    <artifactId>r2dbc-h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Listing 13.1 The `Ingredient` entity class for R2DBC persistence

```
package tacos;

import org.springframework.data.annotation.Id;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@NoArgsConstructor
@RequiredArgsConstructor
@EqualsAndHashCode(exclude = "id")
public class Ingredient {

    @Id
    private Long id;

    private @NonNull String slug;

    private @NonNull String name;
    private @NonNull Type type;

    public enum Type {
        WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
    }

}
```

```
create table Ingredient (
    id identity,
    slug varchar(4) not null,
    name varchar(25) not null,
    type varchar(10) not null
);
```

Listing 13.2 The **Taco** entity class for R2DBC persistence

```
package tacos;

import java.util.HashSet;
import java.util.Set;
import org.springframework.data.annotation.Id;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@NoArgsConstructor
@RequiredArgsConstructor
public class Taco {

    @Id
    private Long id;

    private @NonNull String name;

    private Set<Long> ingredientIds = new HashSet<>();

    public void addIngredient(Ingredient ingredient) {
        ingredientIds.add(ingredient.getId());
    }

}
```

```
create table Taco (
    id identity,
    name varchar(50) not null,
    ingredient_ids array
);
```

Listing 13.3 The `TacoOrder` entity class for R2DBC persistence

```
package tacos;

import java.util.LinkedHashSet;
import java.util.Set;
import org.springframework.data.annotation.Id;
import lombok.Data;

@Data
public class TacoOrder {

    @Id
    private Long id;

    private String deliveryName;
    private String deliveryStreet;
    private String deliveryCity;
    private String deliveryState;
    private String deliveryZip;
    private String ccNumber;
    private String ccExpiration;
    private String ccCVV;

    private Set<Long> tacoIds = new LinkedHashSet<>();

    private List<Taco> tacos = new ArrayList<>();
    public void addTaco(Taco taco) {
        this.tacos.add(taco);
    }

}
```

```
create table Taco_Order (
    id identity,
    delivery_name varchar(50) not null,
    delivery_street varchar(50) not null,
    delivery_city varchar(50) not null,
    delivery_state varchar(2) not null,
    delivery_zip varchar(10) not null,
    cc_number varchar(16) not null,
    cc_expiration varchar(5) not null,
    cc_cvv varchar(3) not null,
    taco_ids array
);
```

```
package tacos.data;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;

import tacos.TacoOrder;

public interface OrderRepository
    extends ReactiveCrudRepository<TacoOrder, Long> {
}
```

Listing 13.4 Calling a reactive repository method

```
@Autowired
OrderRepository orderRepo;

...

orderRepository.findAll()
    .doOnNext(order -> {
        System.out.println(
            "Deliver to: " + order.getDeliveryName());
    })
    .subscribe();
```

Listing 13.5 Persisting Taco objects with a reactive repository

```
package tacos.data;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;
import tacos.Taco;

public interface TacoRepository
    extends ReactiveCrudRepository<Taco, Long> {
}
```

Listing 13.6 Persisting `Ingredient` objects with a reactive repository

```
package tacos.data;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;
import reactor.core.publisher.Mono;
import tacos.Ingredient;

public interface IngredientRepository
    extends ReactiveCrudRepository<Ingredient, Long> {

    Mono<Ingredient> findBySlug(String slug);

}
```

Listing 13.7 Testing a Spring Data R2DBC repository

```
package tacos.data;

import static org.assertj.core.api.Assertions.assertThat;

import java.util.ArrayList;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.data.r2dbc.DataR2dbcTest;

import reactor.core.publisher.Flux;
import reactor.test.StepVerifier;
import tacos.Ingredient;
import tacos.Ingredient.Type;

@DataR2dbcTest
public class IngredientRepositoryTest {

    @Autowired
    IngredientRepository ingredientRepo;

    @BeforeEach
    public void setup() {
        Flux<Ingredient> deleteAndInsert = ingredientRepo.deleteAll()
            .thenMany(ingredientRepo.saveAll(
                Flux.just(
                    new Ingredient("FLTO", "Flour Tortilla", Type.WRAP),
                    new Ingredient("GRBF", "Ground Beef", Type.PROTEIN),
                    new Ingredient("CHED", "Cheddar Cheese", Type.CHEESE)
                )));
    }

    StepVerifier.create(deleteAndInsert)
        .expectNextCount(3)
        .verifyComplete();
}

@Test
public void shouldSaveAndFetchIngredients() {

    StepVerifier.create(ingredientRepo.findAll())
        .recordWith(ArrayList::new)
        .thenConsumeWhile(x -> true)
        .consumeRecordedWith(ingredients -> {
            assertThat(ingredients).hasSize(3);
            assertThat(ingredients).contains(
                new Ingredient("FLTO", "Flour Tortilla", Type.WRAP));
            assertThat(ingredients).contains(
                new Ingredient("GRBF", "Ground Beef", Type.PROTEIN));
            assertThat(ingredients).contains(
                new Ingredient("CHED", "Cheddar Cheese", Type.CHEESE));
        })
        .verifyComplete();

    StepVerifier.create(ingredientRepo.findBySlug("FLTO"))
        .assertNext(ingredient -> {
            assertThat(ingredient.equals(new Ingredient("FLTO", "Flour Tortilla", Type.WRAP)));
        });
}
```

Listing 13.8 Adding a `Taco` collection to `TacoOrder`

```
@Data
public class TacoOrder {

    ...
    @Transient
    private transient List<Taco> tacos = new ArrayList<>();

    public void addTaco(Taco taco) {
        this.tacos.add(taco);
        if (taco.getId() != null) {
            this.tacoIds.add(taco.getId());
        }
    }
}
```

Listing 13.9 Saving `TacoOrders` and `Tacos` as an aggregate

```
package tacos.web.api;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Service;

import lombok.RequiredArgsConstructor;
import reactor.core.publisher.Mono;
import tacos.Taco;
import tacos.TacoOrder;
import tacos.data.OrderRepository;
import tacos.data.TacoRepository;

@Service
@RequiredArgsConstructor
public class TacoOrderAggregateService {

    private final TacoRepository tacoRepo;
    private final OrderRepository orderRepo;

    public Mono<TacoOrder> save(TacoOrder tacoOrder) {
        return Mono.just(tacoOrder)
            .flatMap(order -> {
                List<Taco> tacos = order.getTacos();
                order.setTacos(new ArrayList<>());
                return tacoRepo.saveAll(tacos)
                    .map(taco -> {
                        order.addTaco(taco);
                        return order;
                    }).last();
            })
            .flatMap(orderRepo::save);
    }

}
```

Listing 13.10 Reading `TacoOrders` and `Tacos` as an aggregate

```
public Mono<TacoOrder> findById(Long id) {
    return orderRepo
        .findById(id)
        .flatMap(order -> {
            return tacoRepo.findAllById(order.getTacoIds())
                .map(taco -> {
                    order.addTaco(taco);
                    return order;
                }).last();
        });
}
```

Listing 13.11 Testing the TacoOrderAggregateService

```
package tacos.web.api;

import static org.assertj.core.api.Assertions.assertThat;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.data.r2dbc.DataR2dbcTest;
import org.springframework.test.annotation.DirtiesContext;

import reactor.test.StepVerifier;
import tacos.Taco;
import tacos.TacoOrder;
import tacos.data.OrderRepository;
import tacos.data.TacoRepository;

@DataR2dbcTest
@DirtiesContext
public class TacoOrderAggregateServiceTests {

    @Autowired
    TacoRepository tacoRepo;

    @Autowired
    OrderRepository orderRepo;

    TacoOrderAggregateService service;

    @BeforeEach
    public void setup() {
        this.service = new TacoOrderAggregateService(tacoRepo, orderRepo);
    }

    @Test
    public void shouldSaveAndFetchOrders() {
        TacoOrder newOrder = new TacoOrder();
        newOrder.setDeliveryName("Test Customer");
        newOrder.setDeliveryStreet("1234 North Street");
        newOrder.setDeliveryCity("Notrees");
        newOrder.setDeliveryState("TX");
        newOrder.setDeliveryZip("79759");
        newOrder.setCcNumber("4111111111111111");
        newOrder.setCcExpiration("12/24");
        newOrder.setCcCVV("123");

        newOrder.addTaco(new Taco("Test Taco One"));
        newOrder.addTaco(new Taco("Test Taco Two"));

        StepVerifier.create(service.save(newOrder))
            .assertNext(this::assertOrder)
            .verifyComplete();

        StepVerifier.create(service.findById(1L))
            .assertNext(this::assertOrder)
            .verifyComplete();
    }

    private void assertOrder(TacoOrder savedOrder) {
        assertThat(savedOrder.getId()).isEqualTo(1L);
        assertThat(savedOrder.getDeliveryName()).isEqualTo("Test Customer");
        assertThat(savedOrder.getDeliveryName()).isEqualTo("Test Customer");
        assertThat(savedOrder.getDeliveryStreet()).isEqualTo("1234 North Street");
        assertThat(savedOrder.getDeliveryCity()).isEqualTo("Notrees");
    }
}
```

```
        assertThat(savedOrder.getDeliveryState()).isEqualTo("TX");
        assertThat(savedOrder.getDeliveryZip()).isEqualTo("79759");
        assertThat(savedOrder.getCcNumber()).isEqualTo("4111111111111111");
        assertThat(savedOrder.getCcExpiration()).isEqualTo("12/24");
        assertThat(savedOrder.getCcCVV()).isEqualTo("123");
        assertThat(savedOrder.getTacoIds()).hasSize(2);
        assertThat(savedOrder.getTacos().get(0).getId()).isEqualTo(1L);
        assertThat(savedOrder.getTacos().get(0).getName())
            .isEqualTo("Test Taco One");
        assertThat(savedOrder.getTacos().get(1).getId()).isEqualTo(2L);
        assertThat(savedOrder.getTacos().get(1).getName())
            .isEqualTo("Test Taco Two");
    }

}
```

```
package tacos;

import org.springframework.boot.SpringBootConfiguration;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@SpringBootConfiguration
@EnableAutoConfiguration
public class TestConfig {

}
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
</dependency>
```

Listing 13.12 An `Ingredient` class annotated for Mongo persistence

```
package tacos;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AccessLevel;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
@Document
public class Ingredient {

    @Id
    private String id;
    private String name;
    private Type type;

    public enum Type {
        WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
    }

}
```

Listing 13.13 A `Taco` class annotated for Mongo persistence

```
package tacos;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.rest.core.annotation.RestResource;

import lombok.Data;

@Data
@RestResource(rel = "tacos", path = "tacos")
@Document
public class Taco {

    @Id
    private String id;

    @NotNull
    @Size(min = 5, message = "Name must be at least 5 characters long")
    private String name;

    private Date createdAt = new Date();

    @Size(min=1, message="You must choose at least 1 ingredient")
    private List<Ingredient> ingredients = new ArrayList<>();

    public void addIngredient(Ingredient ingredient) {
        this.ingredients.add(ingredient);
    }

}
```

Listing 13.14 A **TacoOrder** class annotated for Mongo persistence

```
package tacos;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;

@Data
@Document
public class TacoOrder implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private String id;
    private Date placedAt = new Date();

    private User user;

    private String deliveryName;

    private String deliveryStreet;

    private String deliveryCity;

    private String deliveryState;

    private String deliveryZip;

    private String ccNumber;

    private String ccExpiration;

    private String ccCVV;

    private List<Taco> tacos = new ArrayList<>();

    public void addTaco(Taco taco) {
        this.tacos.add(taco);
    }

}
```

```
package tacos.data;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;
import org.springframework.web.bind.annotation.CrossOrigin;

import tacos.Ingredient;
@CrossOrigin(origins="http://localhost:8080")
public interface IngredientRepository
    extends ReactiveCrudRepository<Ingredient, String> {

}
```

```
package tacos.data;

import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.reactive.ReactiveCrudRepository;

import reactor.core.publisher.Flux;
import tacos.TacoOrder;
import tacos.User;

public interface OrderRepository
    extends ReactiveCrudRepository<TacoOrder, String> {

    Flux<TacoOrder> findByUserOrderByPlacedAtDesc(
        User user, Pageable pageable);

}
```

Listing 13.15 Testing a reactive Mongo repository

```
package tacos.data;
import static org.assertj.core.api.Assertions.assertThat;
import java.util.ArrayList;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.mongo.DataMongoTest;
import reactor.core.publisher.Flux;
import reactor.test.StepVerifier;
import tacos.Ingredient;
import tacos.Ingredient.Type;

@DataMongoTest
public class IngredientRepositoryTest {

    @Autowired
    IngredientRepository ingredientRepo;

    @BeforeEach
    public void setup() {
        Flux<Ingredient> deleteAndInsert = ingredientRepo.deleteAll()
            .thenMany(ingredientRepo.saveAll(
                Flux.just(
                    new Ingredient("FLTO", "Flour Tortilla", Type.WRAP),
                    new Ingredient("GRBF", "Ground Beef", Type.PROTEIN),
                    new Ingredient("CHED", "Cheddar Cheese", Type.CHEESE)
                )));
        StepVerifier.create(deleteAndInsert)
            .expectNextCount(3)
            .verifyComplete();
    }

    @Test
    public void shouldSaveAndFetchIngredients() {

        StepVerifier.create(ingredientRepo.findAll())
            .recordWith(ArrayList::new)
            .thenConsumeWhile(x -> true)
            .consumeRecordedWith(ingredients -> {
                assertThat(ingredients).hasSize(3);
                assertThat(ingredients).contains(
                    new Ingredient("FLTO", "Flour Tortilla", Type.WRAP));
                assertThat(ingredients).contains(
                    new Ingredient("GRBF", "Ground Beef", Type.PROTEIN));
                assertThat(ingredients).contains(
                    new Ingredient("CHED", "Cheddar Cheese", Type.CHEESE));
            })
            .verifyComplete();

        StepVerifier.create(ingredientRepo.findById("FLTO"))
            .assertNext(ingredient -> {
                assertThat(ingredient.equals(new Ingredient("FLTO", "Flour Tortilla", Type.WRAP)));
            });
    }
}
```

Listing 13.16 Testing the Mongo OrderRepository

```
package tacos.data;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest;

import reactor.test.StepVerifier;
import tacos.Ingredient;
import tacos.Taco;
import tacos.TacoOrder;
import tacos.Ingredient.Type;

@DataMongoTest
public class OrderRepositoryTest {

    @Autowired
    OrderRepository orderRepo;

    @BeforeEach
    public void setup() {
        orderRepo.deleteAll().subscribe();
    }

    @Test
    public void shouldSaveAndFetchOrders() {
        TacoOrder order = createOrder();

        StepVerifier
            .create(orderRepo.save(order))
            .expectNext(order)
            .verifyComplete();

        StepVerifier
            .create(orderRepo.findById(order.getId()))
            .expectNext(order)
            .verifyComplete();

        StepVerifier
            .create(orderRepo.findAll())
            .expectNext(order)
            .verifyComplete();
    }

    private TacoOrder createOrder() {
        TacoOrder order = new TacoOrder();
        ...
        return order;
    }
}
```

```
$ docker run -p27017:27017 mongo
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-cassandra-reactive</artifactId>
</dependency>
```

```
spring:
  data:
    rest:
      base-path: /data-api
  cassandra:
    keyspace-name: tacocloud
    schema-action: recreate
    local-datacenter: datacenter1
```

```
$ docker network create cassandra-net
$ docker run --name my-cassandra --network cassandra-net \
  -p 9042:9042 -d cassandra:latest
```

```
$ docker run -it --network cassandra-net --rm cassandra cqlsh my-cassandra
cqlsh> create keyspace tacocloud
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
```

Listing 13.17 Annotating **Ingredient** for Cassandra persistence

```
package tacos;

import org.springframework.data.cassandra.core.mapping.PrimaryKey;
import org.springframework.data.cassandra.core.mapping.Table;

import lombok.AccessLevel;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor(access=AccessLevel.PRIVATE, force=true)
@Table("ingredients")
public class Ingredient {

    @PrimaryKey
    private String id;
    private String name;
    private Type type;

    public enum Type {
        WRAP, PROTEIN, VEGGIES, CHEESE, SAUCE
    }

}
```

Listing 13.18 Annotating `Taco` for Cassandra persistence

```
package tacos;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.UUID;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.springframework.data.cassandra.core.cql.Ordering;
import org.springframework.data.cassandra.core.cql.PrimaryKeyType;
import org.springframework.data.cassandra.core.mapping.Column;
import org.springframework.data.cassandra.core.mapping.PrimaryKeyColumn;
import org.springframework.data.cassandra.core.mapping.Table;
import org.springframework.data.rest.core.annotation.RestResource;

import com.datastax.oss.driver.api.core.uuid.Uuids;

import lombok.Data;

@Data
@RestResource(rel = "tacos", path = "tacos")
@Table("tacos")
public class Taco {

    @PrimaryKeyColumn(type=PrimaryKeyType.PARTITIONED)
    private UUID id = Uuids.timeBased();

    @NotNull
    @Size(min = 5, message = "Name must be at least 5 characters long")
    private String name;

    @PrimaryKeyColumn(type=PrimaryKeyType.CLUSTERED,
                      ordering=Ordering.DESCENDING)
    private Date createdAt = new Date();

    @Size(min=1, message="You must choose at least 1 ingredient")
    @Column("ingredients")
    private List<IngredientUDT> ingredients = new ArrayList<>();

    public void addIngredient(Ingredient ingredient) {
        this.ingredients.add(new IngredientUDT(ingredient.getName(), ingredient.getType())
    }

}
```

Listing 13.19 An `Ingredient` user-defined type for Cassandra persistence

```
package tacos;

import org.springframework.data.cassandra.core.mapping.UserDefinedType;

import lombok.AccessLevel;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@NoArgsConstructor(access = AccessLevel.PRIVATE, force = true)
@UserDefinedType("ingredient")
public class IngredientUDT {
    private String name;
    private Ingredient.Type type;
}
```

Listing 13.20 Annotating **TacoOrder** for Cassandra persistence

```
package tacos;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.UUID;

import org.springframework.data.cassandra.core.mapping.Column;
import org.springframework.data.cassandra.core.mapping.PrimaryKey;
import org.springframework.data.cassandra.core.mapping.Table;

import com.datastax.oss.driver.api.core.uuid.Uuids;

import lombok.Data;

@Data
@Table("tacoorders")
public class TacoOrder implements Serializable {
    private static final long serialVersionUID = 1L;

    @PrimaryKey
    private UUID id = Uuids.timeBased();
    private Date placedAt = new Date();

    @Column("user")
    private UserUDT user;

    private String deliveryName;

    private String deliveryStreet;

    private String deliveryCity;

    private String deliveryState;

    private String deliveryZip;

    private String ccNumber;

    private String ccExpiration;

    private String ccCVV;
    @Column("tacos")
    private List<TacoUDT> tacos = new ArrayList<>();

    public void addTaco(Taco taco) {
        this.addTaco(new TacoUDT(taco.getName(), taco.getIngredients()));
    }

    public void addTaco(TacoUDT tacoUDT) {
        this.tacos.add(tacoUDT);
    }

}
```

Listing 13.21 An `Taco` user-defined type for Cassandra persistence

```
package tacos;

import java.util.List;

import org.springframework.data.cassandra.core.mapping.UserDefinedType;

import lombok.Data;

@Data
@UserDefinedType("taco")
public class TacoUDT {

    private final String name;
    private final List<IngredientUDT> ingredients;

}
```

```
package tacos.data;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;

import tacos.Ingredient;

public interface IngredientRepository
    extends ReactiveCrudRepository<Ingredient, String> {

}
```

```
package tacos.data;

import java.util.UUID;

import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.reactive.ReactiveCrudRepository;

import reactor.core.publisher.Flux;
import tacos.TacoOrder;
import tacos.User;

public interface OrderRepository
    extends ReactiveCrudRepository<TacoOrder, UUID> {

    Flux<TacoOrder> findByUserOrderByPlacedAtDesc(
        User user, Pageable pageable);

}
```

Listing 13.22 Testing the Cassandra `IngredientRepository`

```
package tacos.data;

import static org.assertj.core.api.Assertions.assertThat;
import java.util.ArrayList;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.data.cassandra
.DataCassandraTest;

import reactor.core.publisher.Flux;
import reactor.test.StepVerifier;
import tacos.Ingredient;
import tacos.Ingredient.Type;

@DataCassandraTest
public class IngredientRepositoryTest {

    @Autowired
    IngredientRepository ingredientRepo;

    @BeforeEach
    public void setup() {
        Flux<Ingredient> deleteAndInsert = ingredientRepo.deleteAll()
            .thenMany(ingredientRepo.saveAll(
                Flux.just(
                    new Ingredient("FLTO", "Flour Tortilla", Type.WRAP),
                    new Ingredient("GRBF", "Ground Beef", Type.PROTEIN),
                    new Ingredient("CHED", "Cheddar Cheese", Type.CHEESE)
                )));
    }

    StepVerifier.create(deleteAndInsert)
        .expectNextCount(3)
        .verifyComplete();
}

@Test
public void shouldSaveAndFetchIngredients() {

    StepVerifier.create(ingredientRepo.findAll())
        .recordWith(ArrayList::new)
        .thenConsumeWhile(x -> true)
        .consumeRecordedWith(ingredients -> {
            assertThat(ingredients).hasSize(3);
            assertThat(ingredients).contains(
                new Ingredient("FLTO", "Flour Tortilla", Type.WRAP));
            assertThat(ingredients).contains(
                new Ingredient("GRBF", "Ground Beef", Type.PROTEIN));
            assertThat(ingredients).contains(
                new Ingredient("CHED", "Cheddar Cheese", Type.CHEESE));
        })
        .verifyComplete();

    StepVerifier.create(ingredientRepo.findById("FLTO"))
        .assertNext(ingredient -> {
            assertThat(ingredient.equals(new Ingredient("FLTO", "Flour Tortilla", Type.WRAP)));
        });
}
```

```
@DataCassandraTest  
public class OrderRepositoryTest {  
    ...  
}
```

CHAPTER 14

Figure 14.1 RSocket's request-response communication model

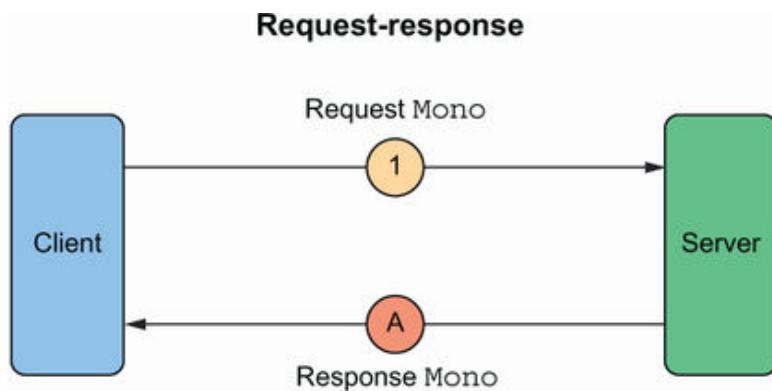


Figure 14.2 RSocket's request-stream communication model

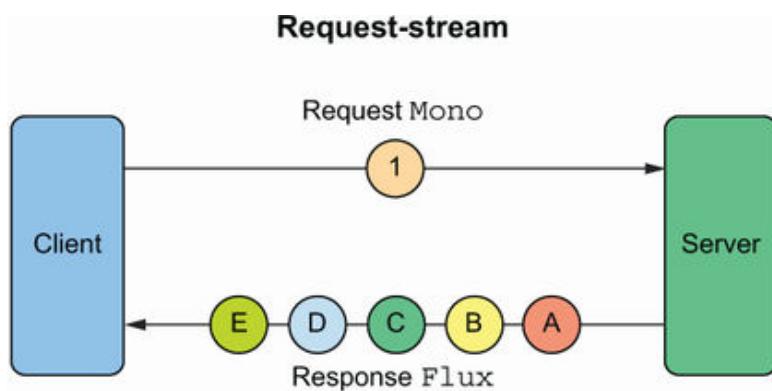


Figure 14.3 RSocket's fire-and-forget communication model

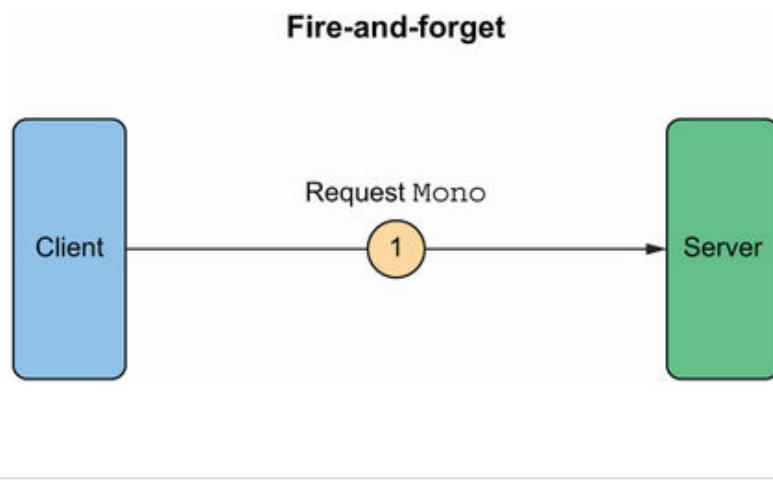
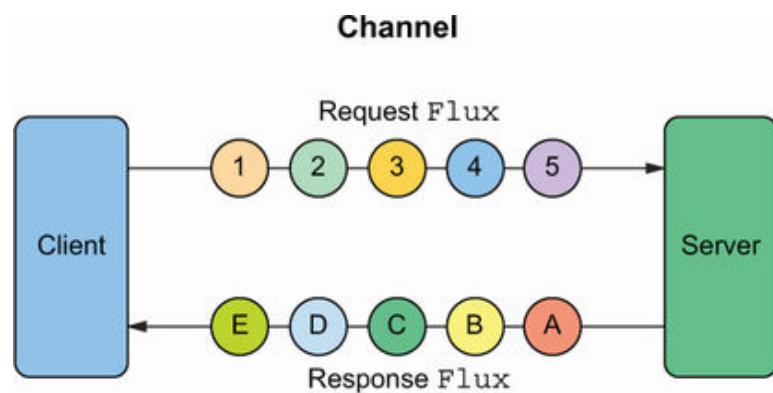


Figure 14.4 RSocket's channel communication model



Listing 14.1 Spring Boot's RSocket starter dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-rsocket</artifactId>
</dependency>
```

Listing 14.2 A simple RSocket request-response server

```
package rsocket;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;
import lombok.extern.slf4j.Slf4j;
import reactor.core.publisher.Mono;

@Controller
@Slf4j
public class GreetingController {

    @MessageMapping("greeting")
    public Mono<String> handleGreeting(Mono<String> greetingMono) {
        return greetingMono
            .doOnNext(greeting ->
                log.info("Received a greeting: {}", greeting))
            .map(greeting -> "Hello back to you!");
    }

}
```

```
@MessageMapping("greeting/{name}")
public Mono<String> handleGreeting(
    @DestinationVariable("name") String name,
    Mono<String> greetingMono) {

    return greetingMono
        .doOnNext(greeting ->
            log.info("Received a greeting from {} : {}", name, greeting))
        .map(greeting -> "Hello to you, too, " + name);
}
```

```
spring:
  rsocket:
    server:
      port: 7000
```

```
package rsocket;
import org.springframework.boot.ApplicationRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.rsocket.RSocketRequester;

@Configuration
@Slf4j
public class RSocketClientConfiguration {

    @Bean
    public ApplicationRunner sender(RSocketRequester.Builder requesterBuilder) {
        return args -> {
            RSocketRequester tcp = requesterBuilder.tcp("localhost", 7000);

            // ... send messages with RSocketRequester ...

        };
    }
}
```

Listing 14.3 Sending a request from a client

```
RSocketRequester tcp = requesterBuilder.tcp("localhost", 7000);

// ... send messages with RSocketRequester ...
tcp
    .route("greeting")
    .data("Hello RSocket!")
    .retrieveMono(String.class)
    .subscribe(response -> log.info("Got a response: {}", response));
```

```
String who = "Craig";
tcp
    .route("greeting/{name}", who)
    .data("Hello RSocket!")
    .retrieveMono(String.class)
    .subscribe(response -> log.info("Got a response: {}", response));
```

Listing 14.4 A model class representing a stock quote

```
package rssocket;
import java.math.BigDecimal;
import java.time.Instant;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class StockQuote {

    private String symbol;
    private BigDecimal price;
    private Instant timestamp;

}
```

Listing 14.5 An RSocket controller to stream stock quotes

```
package rssocket;
import java.math.BigDecimal;
import java.time.Duration;
import java.time.Instant;

import org.springframework.messaging.handler.annotation.DestinationVariable;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;

import reactor.core.publisher.Flux;

@Controller
public class StockQuoteController {

    @MessageMapping("stock/{symbol}")
    public Flux<StockQuote> getStockPrice(
        @DestinationVariable("symbol") String symbol) {
        return Flux
            .interval(Duration.ofSeconds(1))
            .map(i -> {
                BigDecimal price = BigDecimal.valueOf(Math.random() * 10);
                return new StockQuote(symbol, price, Instant.now());
            });
    }
}
```

```
String stockSymbol = "XYZ";

RSocketRequester tcp = requesterBuilder.tcp("localhost", 7000);
tcp
    .route("stock/{symbol}", stockSymbol)
    .retrieveFlux(StockQuote.class)
    .doOnNext(stockQuote ->
        log.info(
            "Price of {} : {} (at {})",
            stockQuote.getSymbol(),
            stockQuote.getPrice(),
            stockQuote.getTimestamp()
        )
    )
    .subscribe();
```

Listing 14.6 A model class representing an alert

```
package rsocket;

import java.time.Instant;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class Alert {

    private Level level;
    private String orderedBy;
    private Instant orderedAt;

    public enum Level {
        YELLOW, ORANGE, RED, BLACK
    }
}
```

Listing 14.7 An RSocket controller to handle alert updates

```
package rsocket;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;
import lombok.extern.slf4j.Slf4j;
import reactor.core.publisher.Mono;

@Controller
@Slf4j
public class AlertController {

    @MessageMapping("alert")
    public Mono<Void> setAlert(Mono<Alert> alertMono) {
        return alertMono
            .doOnNext(alert ->
                log.info("{} alert ordered by {} at {}",
                    alert.getLevel(),
                    alert.getOrderedBy(),
                    alert.getOrderedAt())
            )
            .thenEmpty(Mono.empty());
    }

}
```

```
RSocketRequester tcp = requesterBuilder.tcp("localhost", 7000);
tcp
    .route("alert")
    .data(new Alert(
        Alert.Level.RED, "Craig", Instant.now()))
    .send()
    .subscribe();
log.info("Alert sent");
```

Listing 14.8 A model representing an inbound gratuity request

```
package rsocket;

import java.math.BigDecimal;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class GratuityIn {

    private BigDecimal billTotal;
    private int percent;

}
```

Listing 14.9 A model representing an outbound gratuity response

```
package rsocket;

import java.math.BigDecimal;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class GratuityOut {

    private BigDecimal billTotal;
    private int percent;
    private BigDecimal gratuity;

}
```

Listing 14.10 An RSocket controller that handles multiple messages on a channel

```
package rsocket;
import java.math.BigDecimal;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;

import lombok.extern.slf4j.Slf4j;
import reactor.core.publisher.Flux;

@Controller
@Slf4j
public class GratuityController {

    @MessageMapping("gratuity")
    public Flux<GratuityOut> calculate(Flux<GratuityIn> gratuityInFlux) {
        return gratuityInFlux
            .doOnNext(in -> log.info("Calculating gratuity: {}", in))
            .map(in -> {
                double percentAsDecimal = in.getPercent() / 100.0;
                BigDecimal gratuity = in.getBillTotal()
                    .multiply(BigDecimal.valueOf(percentAsDecimal));
                return new GratuityOut(
                    in.getBillTotal(), in.getPercent(), gratuity);
            });
    }
}
```

Listing 14.11 A client that sends and receives multiple messages over an open channel

```
RSocketRequester tcp = requesterBuilder.tcp("localhost", 7000);

Flux<GratuityIn> gratuityInFlux =
    Flux.fromArray(new GratuityIn[] {
        new GratuityIn(BigDecimal.valueOf(35.50), 18),
        new GratuityIn(BigDecimal.valueOf(10.00), 15),
        new GratuityIn(BigDecimal.valueOf(23.25), 20),
        new GratuityIn(BigDecimal.valueOf(52.75), 18),
        new GratuityIn(BigDecimal.valueOf(80.00), 15)
    })
    .delayElements(Duration.ofSeconds(1));

tcp
    .route("gratuity")
    .data(gratuityInFlux)
    .retrieveFlux(GratuityOut.class)
    .subscribe(out ->
        log.info(out.getPercent() + "% gratuity on "
            + out.getBillTotal() + " is "
            + out.getGratuity()));
```

Table 14.1 The supported RSocket model is determined by the handler method's parameter and return types. ([view table figure](#))

RSocket model	Handler parameter	Handler returns
Request-response	Mono	Mono
Request-stream	Mono	Flux
Fire-and-forget	Mono	Mono<Void>
Channel	Flux	Flux

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

```
spring:
  rsocket:
    server:
      transport: websocket
      mapping-path: /rsocket
```

```
RSocketRequester requester = requesterBuilder.websocket(
    URI.create("ws://localhost:8080/rsocket"));

requester
    .route("greeting")
    .data("Hello RSocket!")
    .retrieveMono(String.class)
    .subscribe(response -> log.info("Got a response: {}", response));
```


CHAPTER 15

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Table 15.1 Actuator endpoints for peeking inside and manipulating the state of a running Spring Boot application (view table figure)

HTTP method	Path	Description
GET	/auditevents	Produces a report of any audit events that have been fired
GET	/beans	Describes all the beans in the Spring application context
GET	/conditions	Produces a report of autoconfiguration conditions that either passed or failed, leading to the beans created in the application context
GET	/configprops	Describes all configuration properties along with the current values
GET , POST , DELETE	/env	Produces a report of all property sources and their properties available to the Spring application
GET	/env/{toMatch}	Describes the value of a single environment property
GET	/health	Returns the aggregate health of the application and (possibly) the health of external dependent applications
GET	/heapdump	Downloads a heap dump
GET	/httptrace	Produces a trace of the most recent 100 requests
GET	/info	Returns any developer-defined information about the application
GET	/loggers	Produces a list of packages in the application along with their configured and effective logging levels
GET , POST	/loggers/{name}	Returns the configured and effective logging level of a given logger; the effective logging level can be set with a <code>POST</code> request
GET	/mappings	Produces a report of all HTTP mappings and their

		corresponding handler methods
GET	/metrics	Returns a list of all metrics categories
GET	/metrics/{name}	Returns a multidimensional set of values for a given metric
GET	/scheduledtasks	Lists all scheduled tasks
GET	/threaddump	Returns a report of all application threads

```
management:
  endpoints:
    web:
      base-path: /management
```

```
management:
  endpoints:
    web:
      exposure:
        include: health,info,beans,conditions
```

```
management:
  endpoints:
    web:
      exposure:
        include: '*'
```

```
management:  
  endpoints:  
    web:  
      exposure:  
        include: '*'  
        exclude: threaddump,heapdump
```

```
$ curl localhost:8080/actuator  
{  
  "_links": {  
    "self": {  
      "href": "http://localhost:8080/actuator",  
      "templated": false  
    },  
    "auditevents": {  
      "href": "http://localhost:8080/actuator/auditevents",  
      "templated": false  
    },  
    "beans": {  
      "href": "http://localhost:8080/actuator/beans",  
      "templated": false  
    },  
    "health": {  
      "href": "http://localhost:8080/actuator/health",  
      "templated": false  
    },  
    ...  
  }  
}
```

```
$ curl localhost:8080/actuator/info  
{ }
```

```
info:  
  contact:  
    email: support@tacocloud.com  
    phone: 822-625-6831
```

```
{  
  "contact": {  
    "email": "support@tacocloud.com",  
    "phone": "822-625-6831"  
  }  
}
```

```
$ curl localhost:8080/actuator/health  
{"status": "UP"}
```

```
management:  
  endpoint:  
    health:  
      show-details: always
```

```
{  
  "status": "UP",  
  "details": {  
    "mongo": {  
      "status": "UP",  
      "details": {  
        "version": "3.5.5"  
      }  
    },  
    "diskSpace": {  
      "status": "UP",  
      "details": {  
        "total": 499963170816,  
        "free": 177284784128,  
        "threshold": 10485760  
      }  
    }  
  }  
}
```

```
{
  "contexts": {
    "application-1": {
      "beans": {
        ...
        "ingredientsController": {
          "aliases": [],
          "scope": "singleton",
          "type": "tacos.ingredients.IngredientsController",
          "resource": "file [/Users/habuma/Documents/Workspaces/
            TacoCloud/ingredient-service/target/classes/tacos/
            ingredients/IngredientsController.class]",
          "dependencies": [
            "ingredientRepository"
          ]
        },
        ...
      },
      "parentId": null
    }
  }
}
```

```
{
  "contexts": {
    "application-1": {
      "positiveMatches": {

        ...
        "MongoDataAutoConfiguration#mongoTemplate": [
          {
            "condition": "OnBeanCondition",
            "message": "@ConditionalOnMissingBean (types:
              org.springframework.data.mongodb.core.MongoTemplate;
              SearchStrategy: all) did not find any beans"
          }
        ],
        ...
      },
      "negativeMatches": {

        ...
        "DispatcherServletAutoConfiguration": {
          "notMatched": [
            {
              "condition": "OnClassCondition",
              "message": "@ConditionalOnClass did not find required
                class 'org.springframework.web.servlet.
                DispatcherServlet'"
            }
          ],
          "matched": []
        },
        ...
      },
      "unconditionalClasses": [
        ...
        "org.springframework.boot.autoconfigure.context.
          ConfigurationPropertiesAutoConfiguration",
        ...
        ]
      }
    }
  }
}
```

Listing 15.1 The results from the /env endpoint

```
$ curl localhost:8080/actuator/env
{
  "activeProfiles": [
    "development"
  ],
  "propertySources": [
    ...
    {
      "name": "systemEnvironment",
      "properties": {
        "PATH": {
          "value": "/usr/bin:/bin:/usr/sbin:/sbin",
          "origin": "System Environment Property \"PATH\""
        },
        ...
        "HOME": {
          "value": "/Users/habuma",
          "origin": "System Environment Property \"HOME\""
        }
      }
    },
    {
      "name": "applicationConfig: [classpath:/application.yml]",
      "properties": {
        "spring.application.name": {
          "value": "ingredient-service",
          "origin": "class path resource [application.yml]:3:11"
        },
        "server.port": {
          "value": 8080,
          "origin": "class path resource [application.yml]:9:9"
        },
        ...
      }
    },
    ...
  ]
}
```

```
$ curl localhost:8080/actuator/env/server.port
{
  "property": {
    "source": "systemEnvironment", "value": "8080"
  },
  "activeProfiles": [ "development" ],
  "propertySources": [
    { "name": "server.ports" },
    { "name": "mongo.ports" },
    { "name": "systemProperties" },
    { "name": "systemEnvironment",
      "property": {
        "value": "8080",
        "origin": "System Environment Property \"SERVER_PORT\""
      }
    },
    { "name": "random" },
    { "name": "applicationConfig: [classpath:/application.yml]",
      "property": {
        "value": 0,
        "origin": "class path resource [application.yml]:9:9"
      }
    },
    { "name": "springCloudClientHostInfo" },
    { "name": "refresh" },
    { "name": "defaultProperties" },
    { "name": "Management Server" }
  ]
}
```

```
$ curl localhost:8080/actuator/env \
-d'{"name":"tacocloud.discount.code","value":"TACOS1234"}' \
-H "Content-type: application/json"
{"tacocloud.discount.code":"TACOS1234"}
```

```
$ curl localhost:8080/actuator/env -X DELETE
{"tacocloud.discount.code":"TACOS1234"}
```

Listing 15.2 HTTP mappings as shown by the /mappings endpoint

```
$ curl localhost:8080/actuator/mappings | jq
{
  "contexts": {
    "application-1": {
      "mappings": {
        "dispatcherHandlers": [
          "webHandler": [
            ...
            {
              "predicate": "{ [/ingredients],methods=[GET] }",
              "handler": "public reactor.core.publisher.Flux<tacos.ingredients.Ingredients>() Lreactor/core/publisher/Flux;" ,
              "details": {
                "handlerMethod": {
                  "className": "tacos.ingredients.IngredientsController",
                  "name": "allIngredients",
                  "descriptor": "() Lreactor/core/publisher/Flux;" ,
                },
                "handlerFunction": null,
                "requestMappingConditions": {
                  "consumes": [],
                  "headers": [],
                  "methods": [
                    "GET"
                  ],
                  "params": [],
                  "patterns": [
                    "/ingredients"
                  ],
                  "produces": []
                }
              }
            },
            ...
            ]
          }
        },
        "parentId": "application-1"
      },
      "bootstrap": {
        "mappings": {
          "dispatcherHandlers": {}
        },
        "parentId": null
      }
    }
}
```

```
{  
  "levels": [ "OFF", "ERROR", "WARN", "INFO", "DEBUG", "TRACE" ],  
  "loggers": {  
    "ROOT": {  
      "configuredLevel": "INFO", "effectiveLevel": "INFO"  
    },  
    ...  
    "org.springframework.web": {  
      "configuredLevel": null, "effectiveLevel": "INFO"  
    },  
    ...  
    "tacos": {  
      "configuredLevel": null, "effectiveLevel": "INFO"  
    },  
    "tacos.ingredients": {  
      "configuredLevel": null, "effectiveLevel": "INFO"  
    },  
    "tacos.ingredients.IngredientServiceApplication": {  
      "configuredLevel": null, "effectiveLevel": "INFO"  
    }  
  }  
}
```

```
{  
  "configuredLevel": null,  
  "effectiveLevel": "INFO"  
}
```

```
$ curl localhost:8080/actuator/loggers/tacos/ingredients \  
-d'{"configuredLevel":"DEBUG"}' \  
-H"Content-type: application/json"
```

```
{  
  "configuredLevel": "DEBUG",  
  "effectiveLevel": "DEBUG"  
}
```

```
{
  "traces": [
    {
      "timestamp": "2020-06-03T23:41:24.494Z",
      "principal": null,
      "session": null,
      "request": {
        "method": "GET",
        "uri": "http://localhost:8080/ingredients",
        "headers": {
          "Host": ["localhost:8080"],
          "User-Agent": ["curl/7.54.0"],
          "Accept": ["*/*"]
        },
        "remoteAddress": null
      },
      "response": {
        "status": 200,
        "headers": {
          "Content-Type": ["application/json; charset=UTF-8"]
        }
      },
      "timeTaken": 4
    },
    ...
  ]
}
```

```
{
  "threadName": "reactor-http-nio-8",
  "threadId": 338,
  "blockedTime": -1,
  "blockedCount": 0,
  "waitedTime": -1,
  "waitedCount": 0,
  "lockName": null,
  "lockOwnerId": -1,
  "lockOwnerName": null,
  "inNative": true,
  "suspended": false,
  "threadState": "RUNNABLE",
  "stackTrace": [
    {
      "methodName": "kevent0",
      "fileName": "KQueueArrayWrapper.java",
      "lineNumber": -2,
      "className": "sun.nio.ch.KQueueArrayWrapper",
      "nativeMethod": true
    },
    {
      "methodName": "poll",
      "fileName": "KQueueArrayWrapper.java",
      "lineNumber": 198,
      "className": "sun.nio.ch.KQueueArrayWrapper",
      "nativeMethod": false
    },
    ...
  ],
  "lockedMonitors": [
    {
      "className": "io.netty.channel.nio.SelectedSelectionKeySet",
      "identityHashCode": 1039768944,
      "lockedStackDepth": 3,
      "lockedStackFrame": {
        "methodName": "lockAndDoSelect",
        "fileName": "SelectorImpl.java",
        "lineNumber": 86,
        "className": "sun.nio.ch.SelectorImpl",
        "nativeMethod": false
      }
    },
    ...
  ],
  "lockedSynchronizers": [],
  "lockInfo": null
}
```

```
$ curl localhost:8080/actuator/metrics | jq
{
  "names": [
    "jvm.memory.max",
    "process.files.max",
    "jvm.gc.memory.promoted",
    "http.server.requests",
    "system.load.average.1m",
    "jvm.memory.used",
    "jvm.gc.max.data.size",
    "jvm.memory.committed",
    "system.cpu.count",
    "logback.events",
    "jvm.buffer.memory.used",
    "jvm.threads.daemon",
    "system.cpu.usage",
    "jvm.gc.memory.allocated",
    "jvm.threads.live",
    "jvm.threads.peak",
    "process.uptime",
    "process.cpu.usage",
    "jvm.classes.loaded",
    "jvm.gc.pause",
    "jvm.classes.unloaded",
    "jvm.gc.live.data.size",
    "process.files.open",
    "jvm.buffer.count",
    "jvm.buffer.total.capacity",
    "process.start.time"
  ]
}
```

```
$ curl localhost:8080/actuator/metrics/http.server.requests
{
  "name": "http.server.requests",
  "measurements": [
    { "statistic": "COUNT", "value": 2103 },
    { "statistic": "TOTAL_TIME", "value": 18.086334315 },
    { "statistic": "MAX", "value": 0.028926313 }
  ],
  "availableTags": [
    { "tag": "exception",
      "values": [ "ResponseStatusException",
                  "IllegalArgumentException", "none" ] },
    { "tag": "method", "values": [ "GET" ] },
    { "tag": "uri",
      "values": [
        "/actuator/metrics/{requiredMetricName}",
        "/actuator/health", "/actuator/info", "/ingredients",
        "/actuator/metrics", "/*" ] },
    { "tag": "status", "values": [ "404", "500", "200" ] }
  ]
}
```

```
$ curl localhost:8080/actuator/metrics/http.server.requests? \
tag=status:404
{
  "name": "http.server.requests",
  "measurements": [
    { "statistic": "COUNT", "value": 31 },
    { "statistic": "TOTAL_TIME", "value": 0.522061212 },
    { "statistic": "MAX", "value": 0 }
  ],
  "availableTags": [
    { "tag": "exception",
      "values": [ "ResponseStatusException", "none" ] },
    { "tag": "method", "values": [ "GET" ] },
    { "tag": "uri",
      "values": [
        "/actuator/metrics/{requiredMetricName}", "/**" ] }
  ]
}
```

```
% curl "localhost:8080/actuator/metrics/http.server.requests? \
tag=status:404&tag=uri:/**"
{
  "name": "http.server.requests",
  "measurements": [
    { "statistic": "COUNT", "value": 30 },
    { "statistic": "TOTAL_TIME", "value": 0.519791548 },
    { "statistic": "MAX", "value": 0 }
  ],
  "availableTags": [
    { "tag": "exception", "values": [ "ResponseStatusException" ] },
    { "tag": "method", "values": [ "GET" ] }
  ]
}
```

Listing 15.3 A custom implementation of `InfoContributor`

```
package tacos.actuator;

import java.util.HashMap;
import java.util.Map;

import org.springframework.boot.actuate.info.Info.Builder;
import org.springframework.boot.actuate.info.InfoContributor;
import org.springframework.stereotype.Component;

import tacos.data.TacoRepository;

@Component
public class TacoCountInfoContributor implements InfoContributor {
    private TacoRepository tacoRepo;

    public TacoCountInfoContributor(TacoRepository tacoRepo) {
        this.tacoRepo = tacoRepo;
    }

    @Override
    public void contribute(Builder builder) {
        long tacoCount = tacoRepo.count().block();
        Map<String, Object> tacoMap = new HashMap<String, Object>();
        tacoMap.put("count", tacoCount);
        builder.withDetail("taco-stats", tacoMap);
    }
}
```

```
{
  "taco-stats": {
    "count": 44
  }
}
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>build-info</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```
springBoot {
  buildInfo()
}
```

```
{
  "build": {
    "artifact": "tacocloud",
    "name": "taco-cloud",
    "time": "2021-08-08T23:55:16.379Z",
    "version": "0.0.15-SNAPSHOT",
    "group": "sia"
  },
}
```

```
<build>
  <plugins>
  ...
    <plugin>
      <groupId>pl.project13.maven</groupId>
      <artifactId>git-commit-id-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

```
plugins {
    id "com.gorylenko.gradle-git-properties" version "2.3.1"
}
```

```
<build>
    <plugins>
        ...
        <plugin>
            <groupId>pl.project13.maven</groupId>
            <artifactId>git-commit-id-plugin</artifactId>
            <configuration>
                <failOnNoGitDirectory>false</failOnNoGitDirectory>
            </configuration>
        </plugin>
    </plugins>
</build>
```

```
gitProperties {
    failOnNoGitDirectory = false
}
```

```
{
    "git": {
        "branch": "main",
        "commit": {
            "id": "df45505",
            "time": "2021-08-08T21:51:12Z"
        }
    },
    ...
}
```

```
management:  
  info:  
    git:  
      mode: full
```

Listing 15.4 Full Git commit info exposed through the /info endpoint

```
"git": {
  "local": {
    "branch": {
      "ahead": "8",
      "behind": "0"
    }
  },
  "commit": {
    "id": {
      "describe-short": "df45505-dirty",
      "abbrev": "df45505",
      "full": "df455055daaf3b1347b0ad1d9dca4ebbc6067810",
      "describe": "df45505-dirty"
    },
    "message": {
      "short": "Apply chapter 18 edits",
      "full": "Apply chapter 18 edits"
    },
    "user": {
      "name": "Craig Walls",
      "email": "craig@habuma.com"
    },
    "author": {
      "time": "2021-08-08T15:51:12-0600"
    },
    "committer": {
      "time": "2021-08-08T15:51:12-0600"
    },
    "time": "2021-08-08T21:51:12Z"
  },
  "branch": "master",
  "build": {
    "time": "2021-08-09T00:13:37Z",
    "version": "0.0.15-SNAPSHOT",
    "host": "Craigs-MacBook-Pro.local",
    "user": {
      "name": "Craig Walls",
      "email": "craig@habuma.com"
    }
  },
  "tags": "",
  "total": {
    "commit": {
      "count": "196"
    }
  },
  "closest": {
    "tag": {
      "commit": {
        "count": ""
      },
      "name": ""
    }
  },
  "remote": {
    "origin": {
      "url": "git@github.com:habuma/spring-in-action-6-samples.git"
    }
  },
  "dirty": "true"
},
```

Listing 15.5 An unusual implementation of `HealthIndicator`

```
package tacos.actuator;

import java.util.Calendar;
import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class WackoHealthIndicator
    implements HealthIndicator {
    @Override
    public Health health() {
        int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
        if (hour > 12) {
            return Health
                .outOfService()
                .withDetail("reason",
                    "I'm out of service after lunchtime")
                .withDetail("hour", hour)
                .build();
        }

        if (Math.random() <= 0.1) {
            return Health
                .down()
                .withDetail("reason", "I break 10% of the time")
                .build();
        }
        return Health
            .up()
            .withDetail("reason", "All is good!")
            .build();
    }
}
```

Listing 15.6 TacoMetrics registers metrics around taco ingredients

```
package tacos.actuator;

import java.util.List;
import org.springframework.data.rest.core.event.AbstractRepositoryEventListener;
import org.springframework.stereotype.Component;
import io.micrometer.core.instrument.MeterRegistry;
import tacos.Ingredient;
import tacos.Taco;

@Component
public class TacoMetrics extends AbstractRepositoryEventListener<Taco> {
    private MeterRegistry meterRegistry;

    public TacoMetrics(MeterRegistry meterRegistry) {
        this.meterRegistry = meterRegistry;
    }

    @Override
    protected void onAfterCreate(Taco taco) {
        List<Ingredient> ingredients = taco.getIngredients();
        for (Ingredient ingredient : ingredients) {
            meterRegistry.counter("tacocloud",
                "ingredient", ingredient.getId()).increment();
        }
    }
}
```

```
$ curl localhost:8080/actuator/metrics/tacocloud
{
  "name": "tacocloud",
  "measurements": [ { "statistic": "COUNT", "value": 84 } ],
  "availableTags": [
    {
      "tag": "ingredient",
      "values": [ "FLTO", "CHED", "LETC", "GRBF",
                  "COTO", "JACK", "TMTO", "SLSA" ]
    }
  ]
}
```

```
$ curl localhost:8080/actuator/metrics/tacocloud?tag=ingredient:FLTO

{
  "name": "tacocloud",
  "measurements": [
    { "statistic": "COUNT", "value": 39 }
  ],
  "availableTags": []
}
```

Listing 15.7 A custom endpoint for taking notes

```
package tacos.actuator;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.springframework.boot.actuate.endpoint.annotation.DeleteOperation;
import org.springframework.boot.actuate.endpoint.annotation.Endpoint;
import org.springframework.boot.actuate.endpoint.annotation.ReadOperation;
import org.springframework.boot.actuate.endpoint.annotation.WriteOperation;
import org.springframework.stereotype.Component;

@Component
@Endpoint(id="notes", enableByDefault=true)
public class NotesEndpoint {

    private List<Note> notes = new ArrayList<>();

    @ReadOperation
    public List<Note> notes() {
        return notes;
    }

    @WriteOperation
    public List<Note> addNote(String text) {
        notes.add(new Note(text));
        return notes;
    }

    @DeleteOperation
    public List<Note> deleteNote(int index) {
        if (index < notes.size()) {
            notes.remove(index);
        }
        return notes;
    }

    class Note {
        private Date time = new Date();
        private final String text;

        public Note(String text) {
            this.text = text;
        }

        public Date getTime() {
            return time;
        }

        public String getText() {
            return text;
        }
    }
}
```

```
$ curl localhost:8080/actuator/notes \
    -d'{"text":"Bring home milk"}' \
    -H"Content-type: application/json"
[{"time":"2020-06-08T13:50:45.085+0000","text":"Bring home milk"}]

$ curl localhost:8080/actuator/notes \
    -d'{"text":"Take dry cleaning"}' \
    -H"Content-type: application/json"
[{"time":"2021-07-03T12:39:13.058+0000","text":"Bring home milk"}, 
 {"time":"2021-07-03T12:39:16.012+0000","text":"Take dry cleaning"}]
```

```
$ curl localhost:8080/actuator/notes
[{"time":"2021-07-03T12:39:13.058+0000","text":"Bring home milk"}, 
 {"time":"2021-07-03T12:39:16.012+0000","text":"Take dry cleaning"}]
```

```
$ curl localhost:8080/actuator/notes?index=1 -X DELETE
[{"time":"2021-07-03T12:39:13.058+0000","text":"Bring home milk"}]
```

```
@Component
@WebEndpoint(id="notes", enableByDefault=true)
public class NotesEndpoint {
    ...
}
```

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
            .antMatchers("/actuator/**").hasRole("ADMIN")  
  
        .and()  
  
        .httpBasic();  
}
```

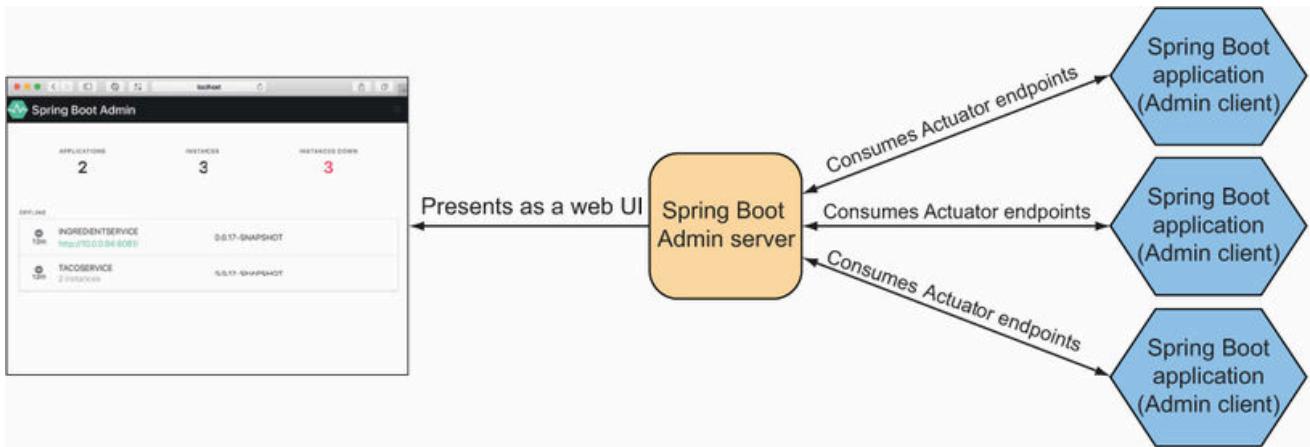
```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .requestMatcher(EndpointRequest.toAnyEndpoint())  
        .authorizeRequests()  
            .anyRequest().hasRole("ADMIN")  
        .and()  
        .httpBasic();  
}
```

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .requestMatcher(  
            EndpointRequest.toAnyEndpoint()  
                .excluding("health", "info"))  
        .authorizeRequests()  
            .anyRequest().hasRole("ADMIN")  
        .and()  
        .httpBasic();  
}
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .requestMatcher(EndpointRequest.to(
            "beans", "threaddump", "loggers"))
        .authorizeRequests()
            .anyRequest().hasRole("ADMIN")
        .and()
            .httpBasic();
}
```

CHAPTER 16

Figure 16.1 The Spring Boot Admin server consumes Actuator endpoints from one or more Spring Boot applications and presents the data in a web-based UI.



```
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-starter-server</artifactId>
</dependency>
```

```
package tacos.admin;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import de.codecentric.boot.admin.server.config.EnableAdminServer;

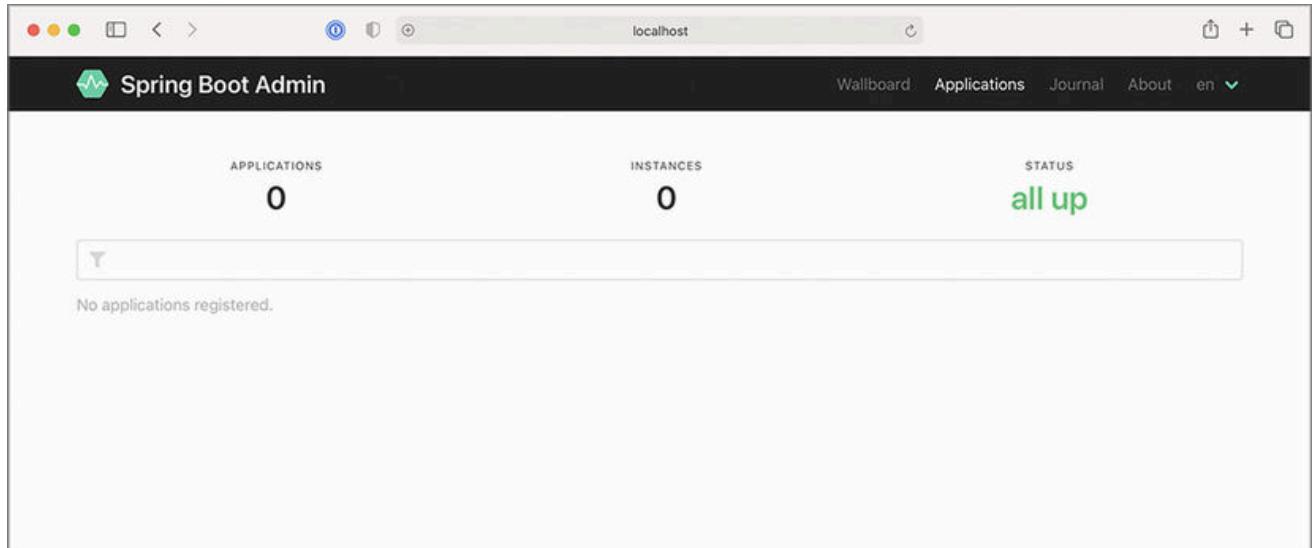
@EnableAdminServer
@SpringBootApplication
public class AdminServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(AdminServerApplication.class, args);
    }

}
```

```
server:
  port: 9090
```

Figure 16.2 A newly created server displayed in the Spring Boot Admin UI. No applications are registered yet.



```
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>
```

```
spring:
  boot:
    admin:
      client:
        url: http://localhost:9090
```

Figure 16.3 The Spring Boot Admin UI displays a single registered application.

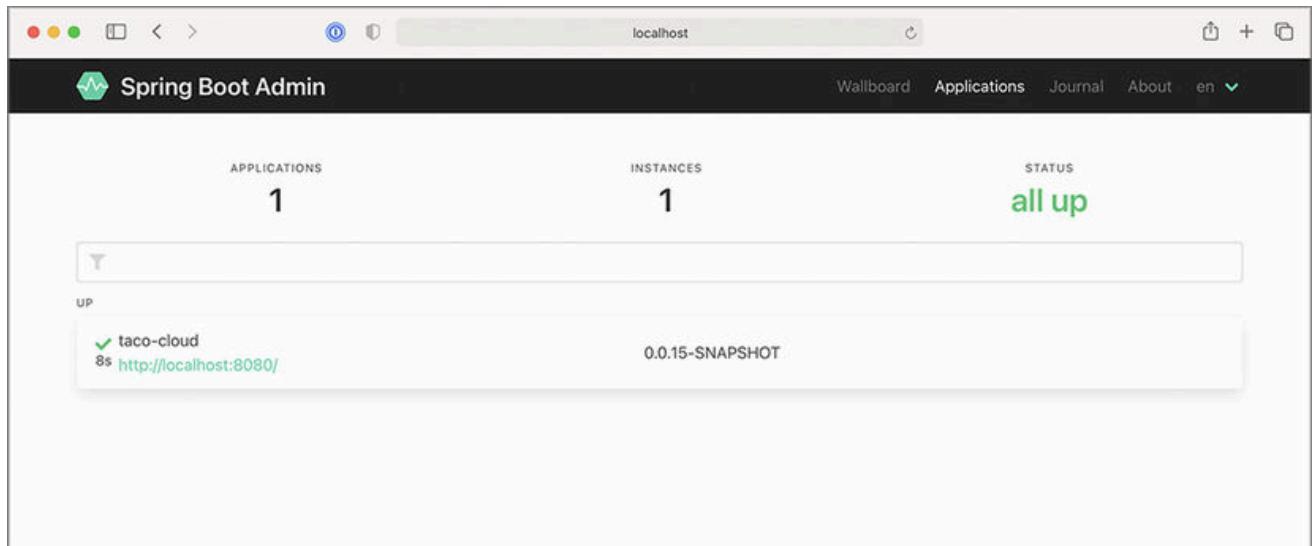


Figure 16.4 The Details screen of the Spring Boot Admin UI displays general health and information about an application.

The screenshot shows the Spring Boot Admin UI for the application 'taco-cloud' with ID 'a973ff14be49'. The left sidebar has a 'Details' tab selected. The main area is divided into two sections: 'Info' and 'Health'. The 'Info' section contains contact information (email: support@tacocloud.com, phone: 822-625-6831) and git details (branch: local, ahead: 8, behind: 0, commit: df45505). The 'Health' section shows diskSpace and mongo status as UP. Below the main content, there is a link to the application's configuration properties.

taco-cloud
Id: a973ff14be49

<http://localhost:8080/> <http://localhost:8080/actuator> <http://localhost:8080/actuator/health>

contact	email: support@tacocloud.com phone: 822-625-6831
git	local: branch: ahead: 8 behind: 0 commit: id: describe-short: df45505-dirty abbrev: df45505 full: df45505daaf3b1347b0ad1d9dca4ebbc6c describe: df45505-dirty message: short: Apply chapter 18 edits

Instance	UP
diskSpace	UP
total	494 GB
free	263 GB
threshold	10.5 MB
exists	true
mongo	
version	3.5.5

Figure 16.5 As you scroll down on the Details screen, you can view additional JVM internal information, including processor, thread, and memory statistics.

The screenshot shows the Spring Boot Admin UI for the application 'taco-cloud' with ID 'a973ff14be49'. The left sidebar has a 'Details' tab selected. The main area is divided into four sections: 'Process', 'Threads', 'Memory: Heap', and 'Memory: Non heap'. The 'Process' section shows PID 18646, Uptime 0h 3m 19s, and CPU usage (User: 0.01, System: 0.13, CpuPct: 8). The 'Threads' section shows 44 live threads, 23 daemon threads, and a peak of 44. The 'Memory: Heap' section shows used memory at 54.6 MB, size at 73.4 MB, and max at 4.29 GB. The 'Memory: Non heap' section shows metaspace at 62.8 MB, used at 93.3 MB, size at 99.5 MB, and max at 1.33 GB. All memory sections include a timeline chart from 19:09:35 to 19:09:41.

taco-cloud
a973ff14be49

PID	UPTIME	PROCESS CPU USAGE	SYSTEM CPU USAGE	CPU PCT
18646	0d 0h 3m 19s	0.01	0.13	8

COUNT	TOTAL TIME SPENT	MAX TIME SPENT
33	0.1180s	0.0120s

LIVE	DAEMON	PEAK LIVE
44	23	44

USED	SIZE	MAX
54.6 MB	73.4 MB	4.29 GB

METASPACE	USED	SIZE	MAX
62.8 MB	93.3 MB	99.5 MB	1.33 GB

Figure 16.6 On the Metrics screen, you can set up watches on any metrics published through the application's /metrics endpoint.

The screenshot shows the Spring Boot Admin interface on a Mac OS X system. The title bar says "localhost". The left sidebar has a green header "taco-cloud a973ff14be49" and a "Metrics" tab selected. The main area shows a dropdown menu "http.server.requests" and several filter dropdowns: "exception" (empty), "method" (GET), "uri" (empty), "outcome" (empty), and "status" (404). A green "Add Metric" button is at the bottom right. Below the filters is a table with columns: metric, COUNT, TOTAL_TIME, and MAX. It lists two rows: one for method:GET status:200 with COUNT 160, TOTAL_TIME 10.481907661, and MAX 0.373752458; and another for method:GET status:404 with COUNT 8, TOTAL_TIME 0.456307709, and MAX 0.186124125. Each row has a trash icon at the end.

Figure 16.7 The Environment screen displays environment properties and includes options to override and filter those values.

The screenshot shows the Spring Boot Admin interface on a Mac OS X system. The title bar says "localhost". The left sidebar has a green header "taco-cloud a973ff14be49" and an "Environment" tab selected. At the top is a search bar with "spring." typed in. The main area shows a table of environment properties under the heading "systemProperties". The first row is "spring.beaninfo.ignore" with value "true". Below it is a section titled "Config resource 'class path resource [application.yml]' via location 'optional:classpath:/' (document #0)". It contains four rows: "spring.application.name" with value "taco-cloud", "spring.data.rest.base-path" with value "/api", "spring.data.cassandra.keyspace-name" with value "tacocloud", and "spring.data.cassandra.schema-action" with value "recreate". Each row shows the source as "class path resource [application.yml] from tacocloud-0.0.15-SNAPSHOT.jar" with a timestamp.

Property	Value
spring.beaninfo.ignore	true
spring.application.name	taco-cloud
spring.data.rest.base-path	/api
spring.data.cassandra.keyspace-name	tacocloud
spring.data.cassandra.schema-action	recreate

Figure 16.8 The Loggers screen displays logging levels for packages and classes in the application and lets you override those levels.

The screenshot shows the Spring Boot Admin interface on a Mac OS X desktop. The title bar says "localhost". The main menu includes "Wallboard", "Applications", "Journal", "About", and "en". On the left, a sidebar has tabs for "Insights", "Loggers" (which is selected), "Mappings", and "Caches". The main content area is titled "Loggers" and shows a search bar with "org.springframework.boot" and a count of "171/715". Below the search bar are two checkboxes: "class only" and "configured". A table lists several Spring framework boot classes with their current log levels:

Class	OFF	ERROR	WARN	INFO	DEBUG	TRACE	Reset
org.springframework.boot				INFO			
org.springframework.boot.ApplicationServletEnvironment				INFO			
org.springframework.boot.BeanDefinitionLoader				INFO			
org.springframework.boot.BeanDefinitionLoader\$ClassExcludeFilter				INFO			
org.springframework.boot.DefaultApplicationArguments				INFO			
org.springframework.boot.DefaultApplicationArguments\$Source				INFO			
org.springframework.boot.SpringApplication				INFO			
org.springframework.boot.SpringApplicationShutdownHook				INFO			

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

```
spring:
  security:
    user:
      name: admin
      password: 53cr3t
```

```
package tacos.admin;

import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.web.reactive
.EnableWebFluxSecurity;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.web.server.SecurityWebFilterChain;

@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain filterChain(ServerHttpSecurity http) throws Exception {
        return http
            .csrf()
            .disable()
            .build();
    }

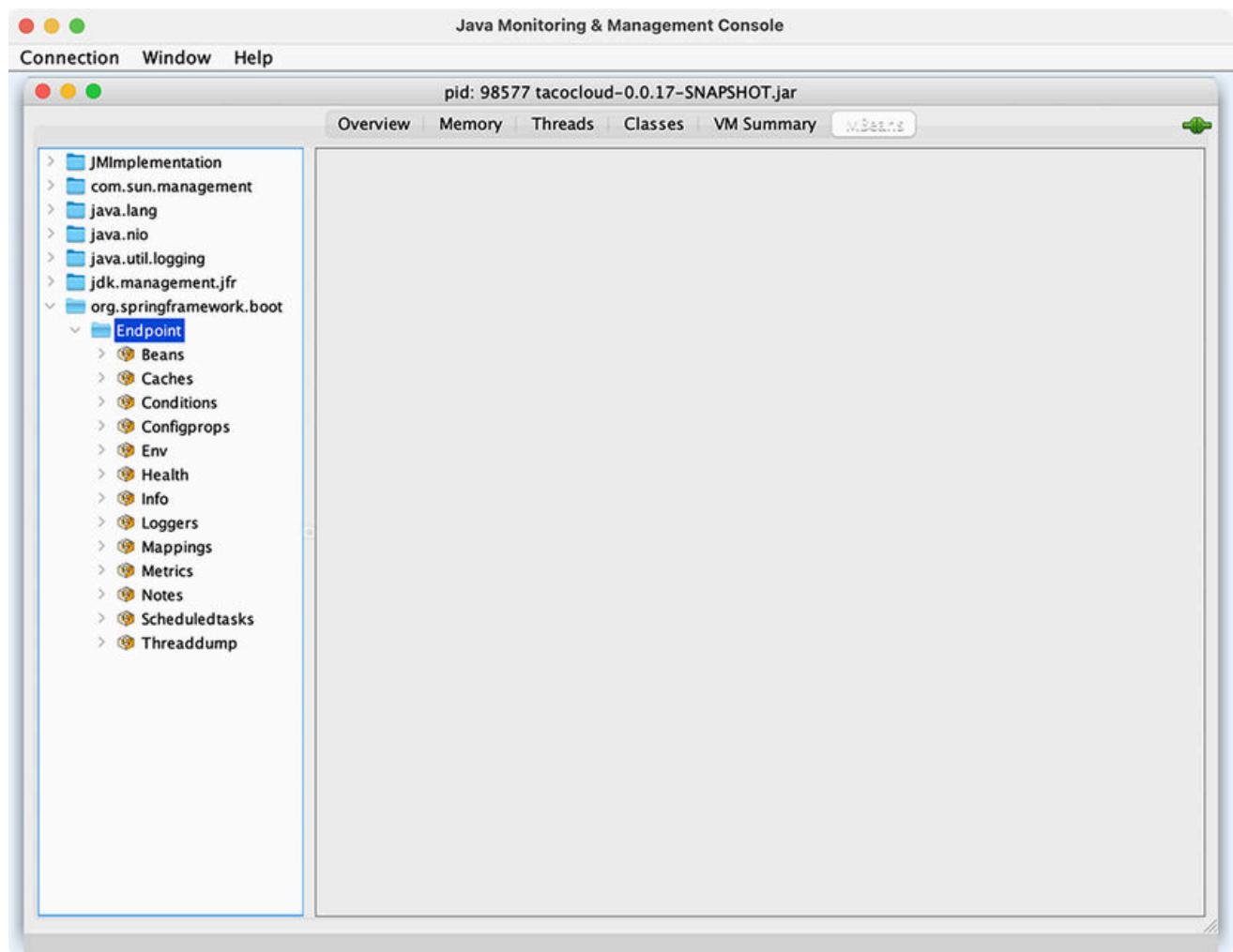
}
```

```
spring:
  boot:
    admin:
      client:
        url: http://localhost:9090
        username: admin
        password: 53cr3t
```

CHAPTER 17

```
spring:  
  jmx:  
    enabled: true
```

Figure 17.1 Actuator endpoints are automatically exposed as JMX MBeans.



```
management:  
  endpoints:  
    jmx:  
      exposure:  
        include: health,info,bean,conditions
```

```
management:  
  endpoints:  
    jmx:  
      exposure:  
        exclude: env,metrics
```

Figure 17.2 Using JConsole to display logging levels from a Spring Boot application

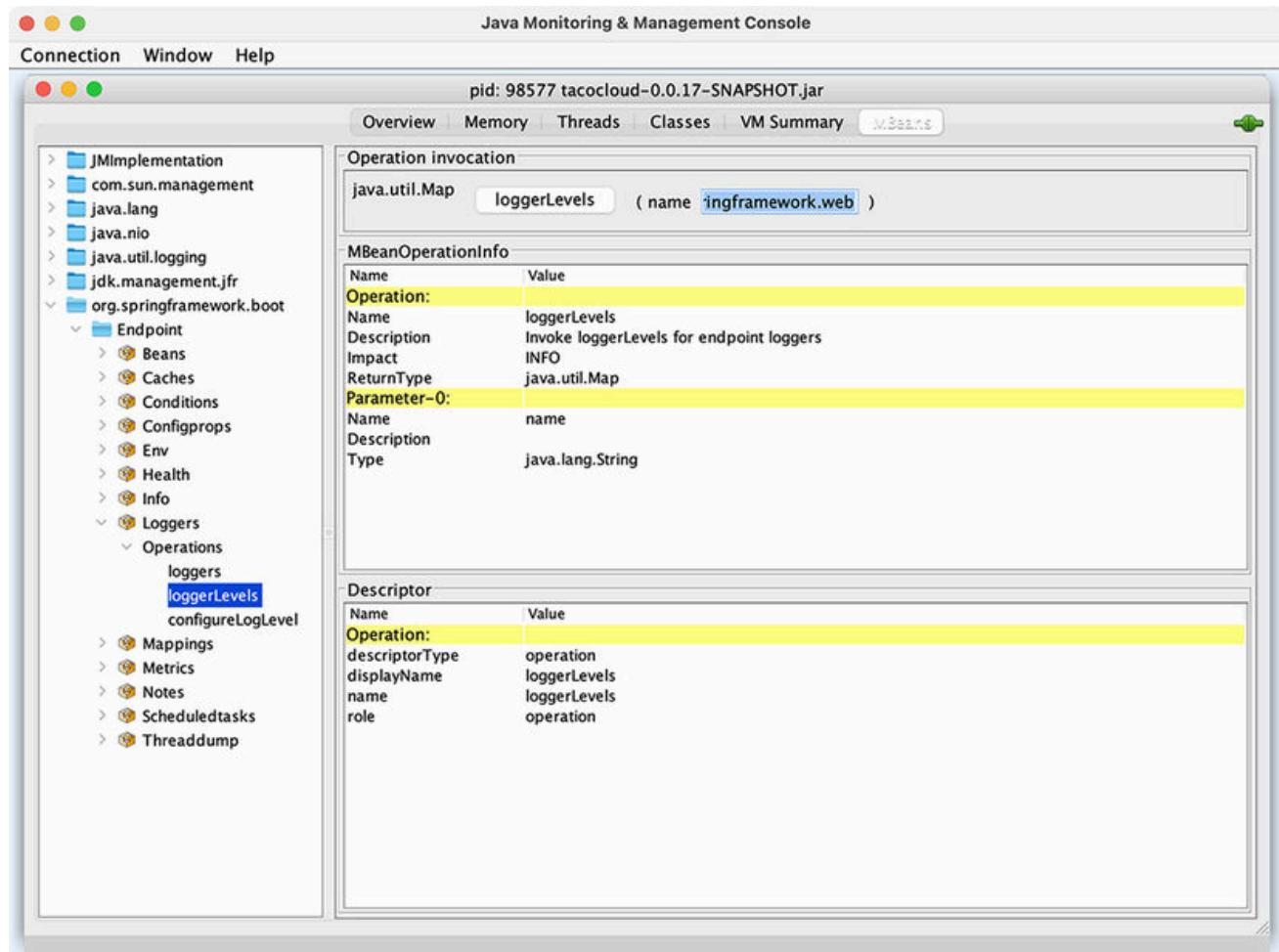
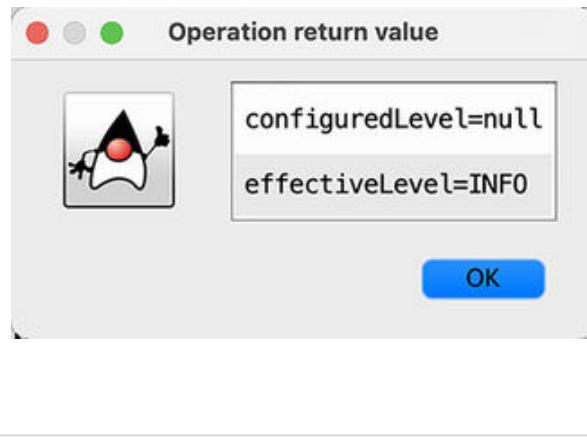


Figure 17.3 Logging levels from the /loggers endpoint MBean displayed in JConsole



Listing 17.1 An MBean that counts how many tacos have been created

```
package tacos.jmx;

import java.util.concurrent.atomic.AtomicLong;
import org.springframework.data.rest.core.event.AbstractRepositoryEventListener;
import org.springframework.jmx.export.annotation.ManagedAttribute;
import org.springframework.jmx.export.annotation.ManagedOperation;
import org.springframework.jmx.export.annotation.ManagedResource;
import org.springframework.stereotype.Service;
import tacos.Taco;
import tacos.data.TacoRepository;

@Service
@ManagedResource
public class TacoCounter
    extends AbstractRepositoryEventListener<Taco> {

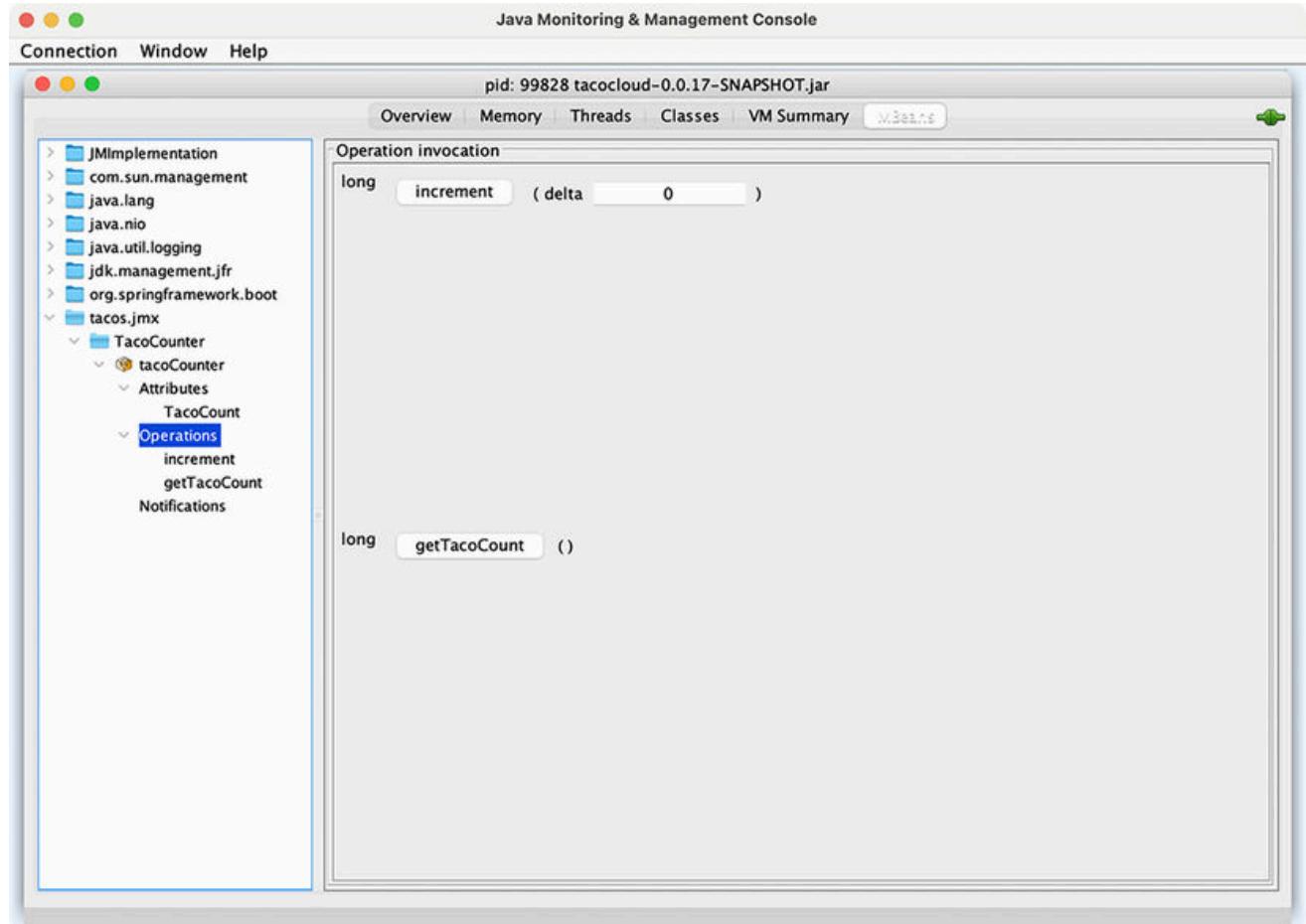
    private AtomicLong counter;
    public TacoCounter(TacoRepository tacoRepo) {
        tacoRepo
            .count()
            .subscribe(initialCount -> {
                this.counter = new AtomicLong(initialCount);
            });
    }

    @Override
    protected void onAfterCreate(Taco entity) {
        counter.incrementAndGet();
    }

    @ManagedAttribute
    public long getTacoCount() {
        return counter.get();
    }

    @ManagedOperation
    public long increment(long delta) {
        return counter.addAndGet(delta);
    }
}
```

Figure 17.4 `TacoCounter`'s operations and attributes as seen in JConsole



Listing 17.2 Sending notifications for every 100 tacos

```
package tacos.jmx;

import java.util.concurrent.atomic.AtomicLong;
import org.springframework.data.rest.core.event.AbstractRepositoryEventListener;
import org.springframework.jmx.export.annotation.ManagedAttribute;
import org.springframework.jmx.export.annotation.ManagedOperation;
import org.springframework.jmx.export.annotation.ManagedResource;
import org.springframework.stereotype.Service;

import org.springframework.jmx.export.notification.NotificationPublisher;
import org.springframework.jmx.export.notification.NotificationPublisherAware;
import javax.management.Notification;

import tacos.Taco;
import tacos.data.TacoRepository;

@Service
@ManagedResource
public class TacoCounter
    extends AbstractRepositoryEventListener<Taco>
    implements NotificationPublisherAware {

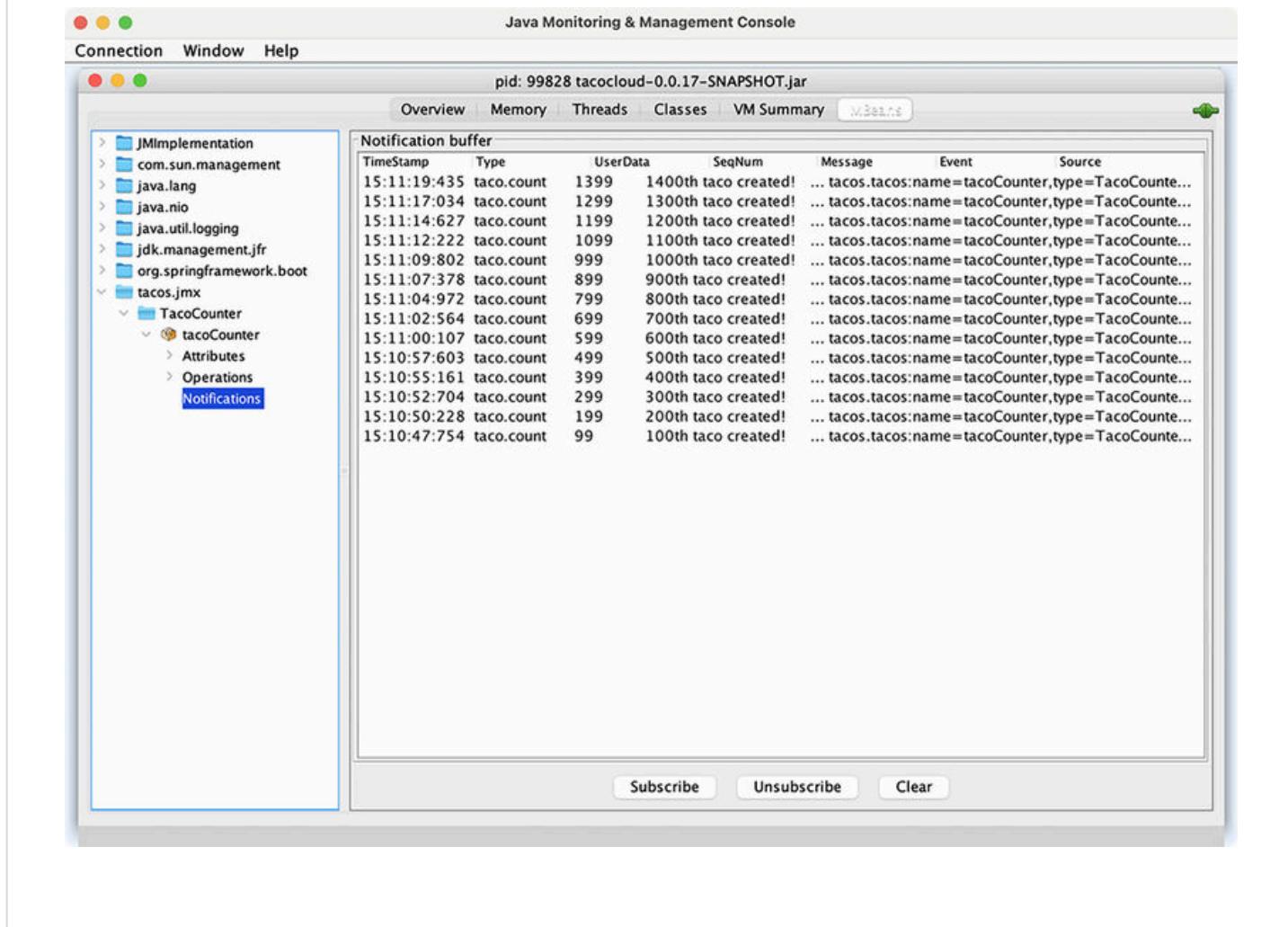
    private AtomicLong counter;
    private NotificationPublisher np;

    @Override
    public void setNotificationPublisher(NotificationPublisher np) {
        this.np = np;
    }

    ...

    @ManagedOperation
    public long increment(long delta) {
        long before = counter.get();
        long after = counter.addAndGet(delta);
        if ((after / 100) > (before / 100)) {
            Notification notification = new Notification(
                "taco.count", this,
                before, after + "th taco created!");
            np.sendNotification(notification);
        }
        return after;
    }
}
```

Figure 17.5 JConsole, subscribed to the **TacoCounter** MBean, receives a notification for every 100 tacos that are created.



CHAPTER 18

```
$ mvnw package
```

```
$ gradlew build
```

```
$ java -jar tacocloud-0.0.19-SNAPSHOT.jar
```

```
$ cf push tacocloud -p target/tacocloud-0.0.19-SNAPSHOT.jar
```

```
FROM openjdk:11.0.12-jre
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

```
$ docker build . -t habuma/tacocloud:0.0.19-SNAPSHOT
```

```
$ docker build tacocloud -t habuma/tacocloud:0.0.19-SNAPSHOT
```

```
$ docker run -p8080:8080 habuma/tacocloud:0.0.19-SNAPSHOT
```

```
$ mvnw spring-boot:build-image
```

```
$ gradlew bootBuildImage
```

```
$ docker run -p8080:8080 library/tacocloud:0.0.19-SNAPSHOT
```

```
$ mvnw spring-boot:build-image \
-Dspring-boot.build-image.imageName=tacocloud/tacocloud:0.0.19-SNAPSHOT
```

```
$ gradlew bootBuildImage --imageName=tacocloud/tacocloud:0.0.19-SNAPSHOT
```

```
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<configuration>
<image>
<name>tacocloud/${project.artifactId}:${project.version}</name>
</image>
</configuration>
</plugin>
```

```
bootBuildImage {
  imageName = "habuma/${rootProject.name}:${version}"
}
```

```
$ docker push habuma/tacocloud:0.0.19-SNAPSHOT
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: taco-cloud-deploy
  labels:
    app: taco-cloud
spec:
  replicas: 3
  selector:
    matchLabels:
      app: taco-cloud
  template:
    metadata:
      labels:
        app: taco-cloud
    spec:
      containers:
        - name: taco-cloud-container
          image: tacocloud/tacocloud:latest
```

```
$ kubectl apply -f deploy.yaml
```

```
$ kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/taco-cloud-deploy-555bd8fdb4-dln45     1/1     Running   0          20s
pod/taco-cloud-deploy-555bd8fdb4-n455b     1/1     Running   0          20s
pod/taco-cloud-deploy-555bd8fdb4-xp756     1/1     Running   0          20s
NAME                                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/taco-cloud-deploy           3/3      3           3          20s
NAME                                         DESIRED  CURRENT  READY   AGE
replicaset.apps/taco-cloud-deploy-555bd8fdb4 3         3         3        20s
```

```
$ kubectl port-forward pod/taco-cloud-deploy-555bd8fdb4-dln45 8080:8080
```

```
server:
  shutdown: graceful
```

```
spring:
  lifecycle.timeout-per-shutdown-phase: 20s
```

```
management:
  health:
    probes:
      enabled: true
```

```
{  
  "status": "UP",  
  "groups": [  
    "liveness",  
    "readiness"  
  ]  
}
```

```
{  
  "status": "UP"  
}
```

```
{  
  "status": "DOWN"  
}
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: taco-cloud-deploy
  labels:
    app: taco-cloud
spec:
  replicas: 3
  selector:
    matchLabels:
      app: taco-cloud
  template:
    metadata:
      labels:
        app: taco-cloud
    spec:
      containers:
        - name: taco-cloud-container
          image: tacocloud/tacocloud:latest
          livenessProbe:
            initialDelaySeconds: 2
            periodSeconds: 5
            httpGet:
              path: /actuator/health/liveness
              port: 8080
          readinessProbe:
            initialDelaySeconds: 2
            periodSeconds: 5
            httpGet:
              path: /actuator/health/readiness
              port: 8080
```

```
@Bean
public ApplicationRunner disableLiveness(ApplicationContext context) {
    return args -> {
        AvailabilityChangeEvent.publish(context, ReadinessState.REFUSING_TRAFFIC);
    };
}
```

```
AvailabilityChangeEvent.publish(context, ReadinessState.ACCEPTING_TRAFFIC);
```

```
AvailabilityChangeEvent.publish(context, LivenessState.BROKEN);
```

```
AvailabilityChangeEvent.publish(context, LivenessState.CORRECT);
```

Listing 18.1 Enabling Spring web applications via Java

```
package tacos;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.web.SpringBootServletInitializer;

public class TacoCloudServletInitializer
    extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(
        SpringApplicationBuilder builder) {
        return builder.sources(TacoCloudApplication.class);
    }
}
```

```
<packaging>war</packaging>
```

```
apply plugin: 'war'
```

```
$ mvnw package
```

```
$ gradlew build
```

```
$ java -jar target/taco-cloud-0.0.19-SNAPSHOT.war
```