```
public void the_user_gets_a_response_indicating_the_attempt_is_
right () throws Throwable {
    // Escriba aquí el código que convierta la frase anterior en
    acciones concretas
    lanzar nueva PendingException ();
}

@Entonces ("^ el usuario obtiene (\\ d +) puntos por el intento $")
public void the_user_gets_points_for_the_attempt (int arg1) throws
Throwable {
    // Escriba aquí el código que convierta la frase anterior en
    acciones concretas
    lanzar nueva PendingException ();
}

@Entonces ("^ el usuario obtiene la insignia FIRST_WON $")
public void the_user_gets_the_FIRST_WON_badge () throws
Throwable {
    // Escriba aquí el código que convierta la frase anterior en
    acciones concretas
    lanzar nueva PendingException ();
}

@Given ("^ el usuario john_snow envía (\\ d +) intentos correctos $")
public void the_user_john_snow_sends_right_attempts (int arg1) throws
Throwable {
    // Escriba aquí el código que convierta la frase anterior en
    acciones concretas
    lanzar nueva PendingException ();
}
. . .
```

Ese código necesita algunas modificaciones, pero es un buen comienzo. Lo copiamos en una nueva clase llamadaMultiplicaciónCaracterísticaPasos dentro del mismo microservices.book paquete. Esa clase contendrá la lógica de todos los pasos incluidos en esta función. Tenga en cuenta que el orden de los pasos en su clase Java no es importante, pero es posible que desee mantener algún orden de apariencia para facilitar la lectura al compararlo con la función.

Primero, corrijamos las diferentes definiciones de pasos definidas en el Dado, cuando, y Luego anotaciones. Queremos usar algunas palabras adicionales como argumentos, no solo números, que son los únicos que se interpretan como argumentos en el código autogenerado:

- *Alias de usuario:* Nos gustaría pasar el alias del usuario como un argumento (texto), para poder reutilizar los mismos pasos para diferentes intentos de varios usuarios.
- *Exactitud:* Nos gustaría especificar si el intento es *derecho* o *equivocado* (texto).
- *Placa de identificación:* Queremos pasar el nombre de la insignia (texto) en lugar de codificarlo en diferentes pasos.

Para lograr eso, solo necesitamos reemplazar estas palabras en nuestros patrones de oraciones con expresiones regulares. Podemos usar ([^ \s] +) como una simple expresión regular para unir la palabra entre espacios. Luego, cuando se reemplaza con la palabra codificada, funcionará. Apliquemos el cambio a la versión original, como se muestra en Listado6-17.

Listado 6-17. MultiplicationFeatureSteps.java: Adición de argumentos (tests-e2e v9)

```
// Original
@When ("^ el usuario john_snow envía (\\ d +) intentos correctos $")
público anular the_user_john_snow_sends_right_attempts (int arg1)
lanza Throwable {
```

```
    // Escriba aquí el código que convierta la frase anterior en
    acciones concretas
    tirar nuevo PendingException ();
}

// Modificado con parámetros extra
@Given ("^ el usuario ([^ \\ s] +) envía (\\ d +) ([^ \\ s] +) intentos")
público void the_user_sends_attempts (cadena final userAlias,
  intentos int finales, cadena final rightOrWrong) lanza
  Throwable {
    // Escriba aquí el código que convierta la frase anterior en
    acciones concretas
    tirar nuevo PendingException ();
}
```

Tenga en cuenta que también debemos agregar los argumentos adicionales al método, que debe seguir el mismo orden de aparición que en la oración.

Así parametrizamos nuestros pasos. Ahora podemos escribir frases como “el usuario john_snow envía 12 intentos correctos” o “el usuario jack_ smith envía 3 intentos incorrectos” y ambos serán manejados por el mismo método de paso.

Todavía perdemos la lógica real de los pasos. Para evitar crear una clase enorme con mucha lógica, usaremos nuestro sentido común orientado a objetos y lo dividiremos:

- La MultiplicaciónCaracterísticaPasos La clase contendrá la lógica principal de nuestras pruebas, orquestando las acciones y afirmando los resultados. Deberíamos intentar mantener cada paso por debajo de las 10 líneas de código.
- La Aplicación de multiplicación es una nueva clase que crearemos para modelar los servicios expuestos en nuestra aplicación: enviar un intento, obtener estadísticas, etc.

- La ApplicationHttpUtils es una clase que agregaremos para brindar soporte básico para realizar llamadas HTTP para que podamos acceder a la API REST de nuestra aplicación.

Listado 6-18 muestra la versión final de MultiplicationFeatureSteps.

Listado 6-18. MultiplicationFeatureSteps.java (tests-e2e v9)

```
clase pública MultiplicationFeatureSteps {

    privado Aplicación MultiplicationApplication;
    privado AttemptResponse lastAttemptResponse;
    privado Estadísticas lastStatsResponse;

    público MultiplicationFeatureSteps () {
        esto.app = nuevo MultiplicationApplication ();
    }

    @Antes
    público limpieza de vacío () {
        app.deleteData ();
    }

    @Given ("^ el usuario ([^ \\ s] +) envía (\\ d +) ([^ \\ s] +) intentos")
    público void the_user_sends_attempts (cadena final userAlias,
                                          intentos finales de int,
                                          cadena final
                                          bien o mal)
                                                  lanza
                                                  Throwable {
        int triesSent = IntStream.range (0, intentos)
                . mapToObj (i -> app.sendAttempt (userAlias, 10, 10,
                                                  "derecho".
                                                  es igual a (correcto o incorrecto)?
                                                          100: 258))
```

```
                // almacena el último intento para su uso posterior
                . peek (respuesta -> lastAttemptResponse =
                respuesta)
                . mapToInt (respuesta -> respuesta.isCorrect
                ()? 1: 0)
                . suma();
        afirmar que (intentosEnviados) .esEqualTo ("correcto".
        es igual a (correcto o incorrecto)? intentos: 0)
                . withFailMessage ("Error al enviar intentos a la
                aplicación");
}
@Entonces ("^ el usuario obtiene una respuesta que indica que el intento es ([^ \
\ s] +) $")
público anular the_user_gets_a_response_indicating_the_
intent_is (
            final String rightOrWrong) lanza Throwable
        {assertThat (lastAttemptResponse.isCorrect ())
                . isEqualTo ("correcto" .equals (rightOrWrong))
                . withFailMessage ("Esperando una respuesta con un"
                        + rightOrWrong + "intento");
}

@Entonces ("^ el usuario obtiene (\\ d +) puntos por el intento $")
público anular the_user_gets_points_for_the_attempt (
            puntos int finales) lanza Throwable {long
        intentId = lastAttemptResponse.getId ();
        Thread.currentThread (). Sleep (2000);
        int score = app.getScoreForAttempt (intentId).
        getScore ();
        assertThat (puntuación) .isEqualTo (puntos);
}
```

```
@Luego ("^ el usuario obtiene la insignia ([^ \\ s] +) $")
público anular the_user_gets_the_type_badge (
        final String badgeType) lanza Throwable {
    long userId = lastAttemptResponse.getUser (). getId ();
    Thread.currentThread (). Sleep (200);
    lastStatsResponse = app.getStatsForUser (userId);
    Lista <String> userBadges = lastStatsResponse.
    getBadges ();
    assertThat (userBadges) .contains (badgeType);
}

@Entonces ("^ el usuario no recibe ninguna insignia $")
público anular the_user_does_not_get_any_badge () lanza
Throwable {
    long userId = lastAttemptResponse.getUser (). getId ();
    Estadísticas estadísticas = app.getStatsForUser (userId); List
    <String> userBadges = stats.getBadges ();
    Si (stats.getScore () == 0) {
        afirmarque (stats.getBadges ()). isNullOrEmpty (); }
    demás {
        asertThat (userBadges) .isEqualTo (lastStatsResponse.
        getBadges ());
    }
}

@Given ("^ el usuario tiene (\\ d +) puntos $")
público anular the_user_has_points (puntos int finales) lanza
Throwable {
```

```
        long userId = lastAttemptResponse.getUser (). getId (); int
        statPoints = app.getStatsForUser (userId). getScore ();

        assertThat (puntos) .isEqualTo (statPoints);
    }

    público AttemptResponse getLastAttemptResponse () {
        regreso lastAttemptResponse;
    }

    público Estadísticas getLastStatsResponse () {
        regreso lastStatsResponse;
    }

    público MultiplicationApplication getApp () {
        regreso aplicación;
    }
}
```

Tenga en cuenta que esta clase no se compilará hasta que implementemos la Multiplicación Aplicación, Estadísticas, y AttemptResponse clases. Esa parte es bastante sencilla, por lo que cubriremos eso más adelante en el capítulo para evitar desviaciones de nuestra implementación de los pasos de la función de Cucumber.

El primer concepto importante que hay que entender es que esta clase se instanciará según el escenario que ejecutemos, de modo que podamos mantener el estado en los diferentes pasos del script. Por lo tanto, podemos usar campos de clase para compartir información entre pasos. En este caso, necesitamos mantener una referencia a nuestro modelo de aplicación.aplicación y a las últimas respuestas recibidas para una solicitud de intento (lastAttemptResponse) y una solicitud de estadísticas (lastStatsResponse).

Dada la similitud con JUnit, no es una sorpresa que el método anotado con @Antes se ejecutará antes de cada escenario. En este caso, invocamos el método de aplicación llamadolimpiar(), que llamará (como veremos más adelante en el capítulo) nuestros nuevos puntos finales de API creados específicamente para pruebas: /gamificación / admin / delete-db y / multiplicación / admin / delete-db. Queremos ejecutar cada escenario con una base de datos limpia.

Teniendo ese conocimiento, la implementación de nuestros pasos es solo una tarea de Java como de costumbre. Solo necesitamos usar nuestra clase de aplicación Aplicación de multiplicación para llamar a la API REST (compatible con ApplicationHttpUtils). Como puede ver aquí, usamos AssertJ para verificar los resultados en nuestro MultiplicaciónCaracterísticaPasos clase. Tenga en cuenta que Cucumber no nos proporciona un marco de prueba completo: es solo una herramienta para vincular las definiciones de características en Gherkin al código Java utilizando un enfoque BDD. Para poder construir nuestras pruebas necesitamos combinarlo con JUnit y muy probablemente con otros frameworks / bibliotecas como AssertJ, TestNG, Mockito, etc., dependiendo de nuestras necesidades. Figura6-1 muestra una vista general de las diferentes partes de nuestro proyecto de principio a fin y cómo se conecta al sistema existente.

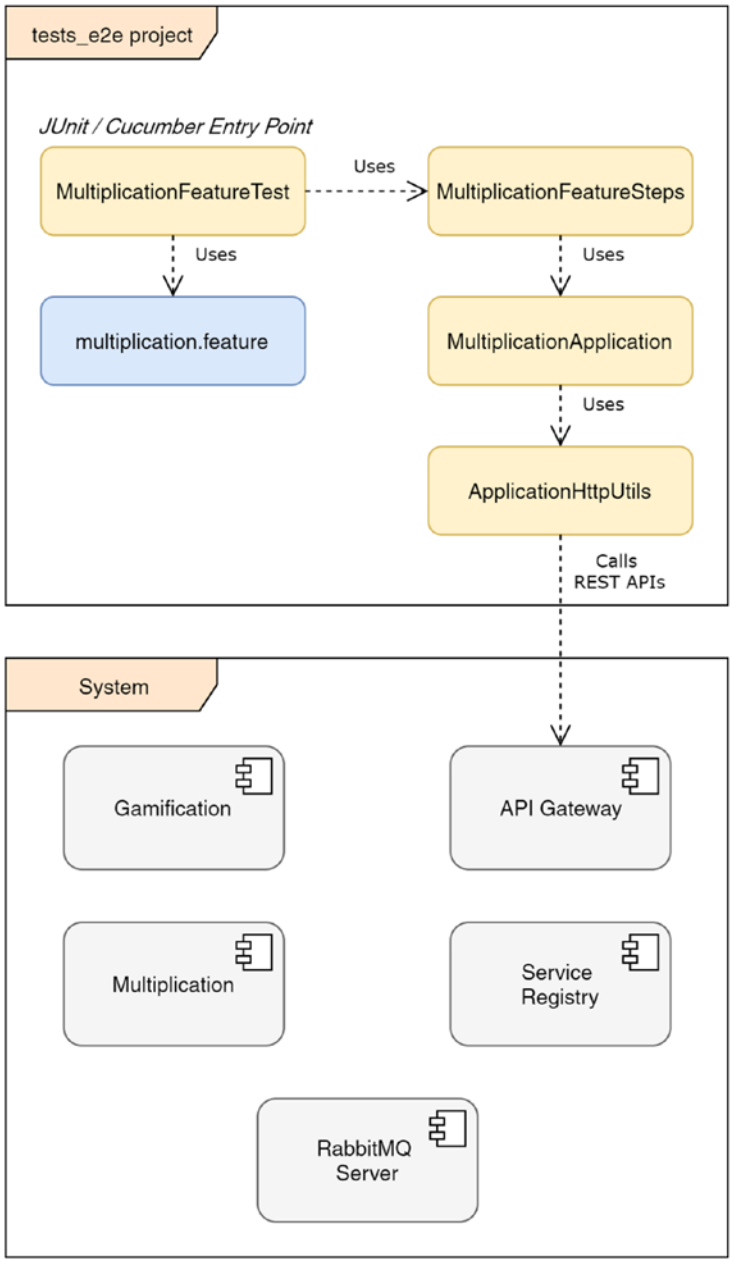

Figura 6-1. *Vista general del proyecto de un extremo a otro y cómo se conecta al sistema existente*

Por último, pero no menos importante, vale la pena mencionar que Pepino no distingue entre las anotaciones @Dado, @Cuando, y @Luego para vincular los pasos al código. Puede ver algunos ejemplos en el archivo de características: el paso ^ el usuario ([^ \ s] +) envía (\ d +) ([^ \ s] +) intentos está siendo utilizado en un Dado y un Cuándo declaración indistintamente, pero el paso se define con una @ Dado anotación en el código. Esto es útil para implementar casos como estos, en los que el mismo paso se puede reutilizar para configurar el escenario y afirmar un resultado esperado.

PREPARACIÓN PARA LA PRODUCCIÓN: PROTECTORES DE SUEÑO EN SISTEMAS CI

tal vez hayas notado que estamos usando algunos dormir() llama a esperar antes de continuar con algunos pasos. Los necesitamos allí porque nuestro sistema está *eventualmente consistente*: El segundo microservicio tardará algún tiempo en consumir el evento, y éste se tomará su tiempo para completar la operación. ladormir() los métodos están ahí solo para ilustrar esto; asegúrese de no utilizar esas esperas en un sistema Ci real. pueden convertirse fácilmente en la fuente de muchos errores falsos (por ejemplo, si tiene una máquina Ci congestionada que no responde a tiempo).

tome esto como un ejercicio e implemente un mecanismo de reintento en el sistema que realice la llamada de descanso varias veces hasta que obtenga una respuesta válida o expire un tiempo de espera.

Las clases de apoyo

Como se presentó anteriormente, Cucumber no es un marco de prueba completo de extremo a extremo. Cumple muy bien sus promesas: vincular definiciones de características en un lenguaje amigable para los humanos para probar suites en código, con pasos definidos. En nuestro sistema, necesitamos crear una lógica adicional para cumplir con nuestra estrategia de extremo a extremo. Tenga en cuenta que esta lógica no es específica de Pepino ni Spring-Boot, por lo que puede tomar

toda esta subsección como un desafío e implementar el resto del código usted mismo, utilizando los métodos que vio en el MultiplicaciónCaracterísticaPasos class como referencia.

La Aplicación de multiplicación La clase, que dejamos pendiente de ser implementada, modelará el comportamiento del sistema (ver Listado 6-19). Los métodos representan acciones que puede realizar un consumidor REST. Tenga en cuenta que se basa enApplicationHttpUtils, que aún no hemos cubierto. Sin embargo, como puedes imaginar, realiza el verdadero*Fontanería de conexión HTTP.*

Listado 6-19. MultiplicationApplication.java: Adición de argumentos (tests-e2e v9)

```
clase pública MultiplicationApplication {

    privado Cadena final estática APPLICATION_BASE_URL = "http: //
    localhost: 8000 / api";
    privado Cadena final estática CONTEXT_ATTEMPTS = "/ resultados";
    privado Cadena final estática CONTEXT_SCORE = "/ puntuaciones /";
    privado Cadena final estática CONTEXT_STATS = "/ stats";
    privado Cadena final estática CONTEXT_USERS = "/ usuarios /";
    privado Cadena final estática CONTEXT_LEADERBOARD = "/
    líderes";
    privado Cadena final estática CONTEXT_DELETE_DATA_GAM = "/
    gamification / admin / delete-db";
    privado Cadena final estática CONTEXT_DELETE_DATA_MULT = "/
    multiplication / admin / delete-db";

    privado ApplicationHttpUtils httpUtils;

    público MultiplicationApplication () {
        esto.httpUtils = nuevo ApplicationHttpUtils (APPLICATIO
        N_BASE_URL);
    }
```

```
público AttemptResponse sendAttempt (String userAlias, int
factorA, int factorB, int result) {
    String intentJson = "{\" usuario \ ": {\" alias \ ": \" "+
    userAlias   +" \ "}," +
            "\" multiplicación \ ": {\" factorA \ ": \" "+ factorA
            + "\", \ "factorB \": \ "" + factorB + "\"}, "+" \
        "resultAttempt \": \ "" + resultado + "\"} "; Cadena de
        respuesta = httpUtils.post ( CONTEXT_ATTEMPTS,
        intentJson);
    ObjectMapper objectMapper = nuevo ObjectMapper ();
    objectMapper.configure (DeserializationFeature.FAIL_ON_
    UNKNOWN_PROPERTIES,falso); intentar {

        regreso objectMapper.readValue (respuesta,
        AttemptResponse.class);
    } captura (IOException e) {
        tirar nuevo RuntimeException (e);
    }
}

público ScoreResponse getScoreForAttempt (largo intentoId) {
    Respuesta de cadena = httpUtils.get (CONTEXT_SCORE
    + intentId);
    Si (response.isEmpty ()) {
        volver nuevo ScoreResponse (0); }
    demás {
        ObjectMapper objectMapper = nuevo ObjectMapper ();
        objectMapper.configure (DeserializationFeature.FAIL_
        ON_UNKNOWN_PROPERTIES,falso); intentar {

            regreso objectMapper.readValue (respuesta,
            ScoreResponse.class);
```

```
            } captura (IOException e) {
                tirar nuevo RuntimeException (e);
            }
        }
}

público Estadísticas getStatsForUser (long userId) {
        Cadena de respuesta = httpUtils.get (CONTEXT_STATS
        + "? UserId =" + userId);
        ObjectMapper objectMapper = nuevo ObjectMapper ();
        objectMapper.configure (DeserializationFeature.FAIL_ON_
        UNKNOWN_PROPERTIES,falso); intentar {

            regreso objectMapper.readValue (respuesta, clase
            Stats.);
        } captura (IOException e) {
            tirar nuevo RuntimeException (e);
        }
}

público Usuario getUser (long userId) {
        Respuesta de cadena = httpUtils.get (CONTEXT_USERS +
        userId);
        ObjectMapper objectMapper = nuevo ObjectMapper ();
        objectMapper.configure (DeserializationFeature.FAIL_ON_
        UNKNOWN_PROPERTIES,falso); intentar {

            regreso objectMapper.readValue (respuesta,
            User.class);
        } captura (IOException e) {
            tirar nuevo RuntimeException (e);
        }
}
```

```
    público List <LeaderBoardPosition> getLeaderboard () {
        Respuesta de cadena = httpUtils.get (CONTEXT_LEADERBOARD);
        ObjectMapper objectMapper =nuevo ObjectMapper ();
        objectMapper.configure (DeserializationFeature.FAIL_ON_
        UNKNOWN_PROPERTIES,falso); intentar {

            JavaType javaType = objectMapper.getTypeFactory ().
            constructCollectionType (List.class,
            LeaderBoardPosition.class);
            regreso objectMapper.readValue (respuesta, javaType); }
        captura (IOException e) {
            tirar nuevo RuntimeException (e);
        }
    }

    público void deleteData () {
        httpUtils.post (CONTEXT_DELETE_DATA_GAM, "");
        httpUtils.post (CONTEXT_DELETE_DATA_MULT, "");
    }
}
```

Básicamente, esta clase recupera las representaciones JSON de las API REST y las asigna a objetos simples utilizando Jackson. También necesitamos crear elEstadísticas, AttemptResponse, Usuario, ScoreResponse, y LeaderBoardPosition clases.

EJERCICIO

Crea el Estadísticas, AttemptResponse, Usuario, ScoreResponse, y LeaderBoardPosition clases sencillas. No olvide seguir la misma estructura que el JsOn recibido o codificar los analizadores usted mismo. para que sea sencillo, las clases enumeradas representan la misma estructura. Por ejemplo, tu

Necesito hacer referencia Usuario de AttemptResponse. si necesita ayuda, recuerde que puede encontrar el código fuente completo dentro del v9 repositorio en github (tests_e2e carpeta).

La última clase de apoyo de este pequeño marco para conectarse a nuestro sistema es ApplicationHttpUtils (ver listado 6-20). Este usa la API Apache HTTP Fluent para ejecutar las solicitudes y obtener las respuestas de API Gateway. Ver *https://tpd.io/fl-api* para obtener más información sobre Apache HTTP Fluent API.

Listado 6-20. ApplicationHttpUtils.java (tests-e2e v9)

```
clase pública ApplicationHttpUtils {

    privado final String baseUrl;

    público ApplicationHttpUtils (final String baseUrl) {
        esto.baseUrl = baseUrl;
    }

    público Publicación de cadena (contexto de cadena final, cuerpo de cadena final) {
        intentar {
            Respuesta HttpResponse = Request.Post (baseUrl +
            contexto)
                    . bodyString (cuerpo, ContentType.APPLICATION_
                    JSON)
                    . ejecutar (). returnResponse ();
            asertIs200 (respuesta);
            regreso EntityUtils.toString (response.getEntity ()); }
        captura (IOException e) {
            tirar nuevo RuntimeException (e);
        }
    }
```

```
    público String get (contexto final de String) {
        intentar {
            Respuesta de HttpResponse = Request.Get (baseUrl +
            contexto)
                    . ejecutar (). returnResponse ();
            asertIs200 (respuesta);
            regreso EntityUtils.toString (response.getEntity ()); }
        captura (IOException e) {
            tirar nuevo RuntimeException (e);
        }
    }

    privado void assertIs200 (final HttpResponse httpResponse) {
        afirmar que (httpResponse.getStatusLine ().
        getStatusCode ()). isEqualTo (200);
    }
}
```

Reutilización de pasos en todas las funciones

Esta sección presenta la segunda función para probar, y mientras la codifica, verá cómo puede usar los pasos definidos anteriormente, ahorrando así mucho tiempo y duplicación de código.

Ahora cubrimos la funcionalidad de la tabla de clasificación con algunas pruebas de un extremo a otro. Probemos dos escenarios básicos:

1. Desde un nuevo comienzo, cuando el usuario envía más intentos correctos que otro y, por lo tanto, se convierte en el primero en el ranking.

2. Dada una situación en la que un usuario está por encima de otro en el ranking, y el que está por debajo del usuario puede pasar a una posición de ranking superior obteniendo una puntuación más alta.

La característica file no es más que escribir la misma historia en Gherkin, como se muestra en Listing 6-21.

Listado 6-21. Leaderboard.feature (tests-e2e v9)

```
Característica: los usuarios se enumeran de la puntuación más alta a la más baja, y cuando
  obtienen puntos que pueden ascender en la clasificación.

  Escenario: un usuario envía un mayor número de intentos
    correctos y se posiciona en el primer lugar del ranking. Cuando el
    usuario john envía 2 intentos correctos
      Y el usuario Peter envía 1 intentos correctos
      Entonces el usuario john es el número 1 en la tabla de clasificación y
      el usuario peter es el número 2 en la tabla de clasificación.

  Escenario: un usuario pasa a otro cuando obtiene una puntuación más alta.
      Dado el usuario, John envía 3 intentos correctos.
      Y el usuario Peter envía 2 intentos correctos.
      Y el usuario john es el número 1 en la tabla de clasificación
      Cuando el usuario peter envía 2 intentos correctos
      Entonces el usuario peter es el número 1 en la tabla de clasificación y
      el usuario john es el número 2 en la tabla de clasificación.
```

Como puede ver, estamos usando los pasos definidos en el MultiplicaciónCaracterísticaPasos class para configurar la escena (intentos de envío). La parte complicada aquí es que no queremos juntar nuestros pasos junto con otros en la misma clase, ya que queremos organizarlos correctamente y evitar tener una clase enorme con todos los pasos de todas nuestras funciones juntas. Crearemos una nueva clase llamadaLeaderboardFeatureSteps. Pero, si hacemos eso, ¿cómo podemos acceder a la funcionalidad y los datos que existen dentro MultiplicationFeatureSteps? Recuerde que el paso "enviar intento" almacena el resultado dentro de su clase (lastAttemptResponse), que es inaccesible desde la clase de pasos de la función de la tabla de clasificación. Es decir, a menos que podamos pasarlo a nuestra nueva clase.

La buena noticia es que podemos resolver esto con la inyección de dependencia estándar, que es proporcionada por el pepino-picocontenedor dependencia (tenga en cuenta que no necesitamos Spring para que funcione). Dado que Cucumber creará una instancia de un nuevo objeto de cada clase que contenga pasos (una de sus anotaciones), podemos usar la inyección del constructor para decirle a Cucumber que inyecte la instancia deMultiplicaciónCaracterísticaPasos dentro LeaderboardFeatureSteps. Para lograr esto, solo necesitamos pasar la clase de definición de paso como un argumento de constructor. La biblioteca de inyección de dependencia liviana hecha ad hoc para Cucumber (Picocontainer) hará el truco por nosotros.

Teniendo en cuenta la inyección de dependencias para nuestra nueva clase, ahora podemos codificarla e incluir el único paso nuevo necesario para verificar la funcionalidad de la tabla de clasificación. Como puede ver en el listado6-22, no necesitamos ninguna anotación para que la inyección de dependencia funcione.

Listado 6-22. LeaderboardFeatureSteps.java (tests-e2e v9)

```
clase pública LeaderboardFeatureSteps {

    privado MultiplicationFeatureSteps mSteps;

    público LeaderboardFeatureSteps (pasos finales de
    MultiplicationFeature mSteps) {
        esto.mSteps = mSteps;
    }

    @Entonces ("^ el usuario ([^ \\ s] +) es el número (\\ d +) en la tabla de
    clasificación $")
    público anular the_user_is_the_number_on_the_leaderboard
    (usuario de cadena final, posición int final) lanza Throwable {
        Thread.currentThread (). Sleep (500); Enumere
        <LeaderBoardPosition> leaderBoard = mSteps. getApp ().
        getLeaderboard ();
        afirmar que (tabla de líderes) .isNotEmpty ();
```

```
            long userId = leaderBoard.get
            (posición - 1) .getUserId ();
            String userAlias   = mSteps.getApp (). GetUser (userId).
            getAlias   ();
            assertThat (userAlias) .isEqualTo (usuario);
      }
}
```

Es una característica sencilla: solo necesitamos verificar la posición del usuario y la orquestación de los pasos realizados por el escenario Cucumber hará todo lo demás. Recuerde que, para que funcione, también necesitamos crear la clase de prueba principalTabla de clasificaciónFeatureTest con las opciones de corredor y pepino correspondientes, como se muestra en el listado 6-23.

Listado 6-23. LeaderboardFeatureTest.java (tests-e2e v9)

```
@RunWith (Pepino.class)
@CucumberOptions (plugin = {"pretty", "html: target / cucumber",
"junit: target / junit-report.xml"},
            features = "src / test / resources / leaderboard.feature")
clase pública LeaderboardFeatureTest {}
```

Ejecución de pruebas y verificación de informes

Es hora de verlo funcionar. Como de costumbre, necesitamos iniciar nuestro conjunto de servicios, pero ahora recuerde que necesitamos activar elprueba perfil para la puerta de enlace (para enrutar los puntos finales de administración), la multiplicación y los servicios de gamificación (para usar una base de datos de prueba y exponer los beans de administración). Para hacer eso, simplemente ejecútelos con el contenedor Maven como se muestra en el Listado. 6-24 (o siga las instrucciones de su IDE preferido).

Listado 6-24. Línea de comando: ejecutar un perfil específico (Gamification v9)

```
mvnw spring-boot: ejecutar -Drun.profiles = test
```

A continuación, se muestra un resumen de los pasos:

1. Ejecute el servidor RabbitMQ (si aún no se está ejecutando).
2. Ejecute el microservicio de registro de servicios (sin perfil específico).
3. Ejecute el microservicio de puerta de enlace (prueba perfil).
4. Ejecute el microservicio de multiplicación (prueba perfil).
5. Ejecute el microservicio de gamificación (prueba perfil).
6. Ejecute el servidor de embarcadero desde el ui carpeta raíz (opcional, ya que nuestras pruebas no usan la interfaz de usuario).

Cuando la aplicación esté en funcionamiento, simplemente ejecute las pruebas para el pruebas e2e proyecto como de costumbre con Maven:

```
prueba mvnw
```

Verá una salida bastante impresa en la que Cucumber le indica el estado de los escenarios y los pasos que se están ejecutando, como se muestra en la Figura 6-2.

```
  Scenario Outline: The user sends a right attempt after <previous_attempts> right attempts and
then gets a badge <badge_name> # src/test/resources/multiplication.feature:32
    Given the user john_snow sends <previous_attempts> right attempts
    When the user john_snow sends 1 right attempts
    Then the user gets a response indicating the attempt is right
    And the user gets 10 points for the attempt
    And the user gets the <badge_name> badge

    Examples:

  Scenario Outline: The user sends a right attempt after 9 right attempts and then gets a badge
BRONZE_MULTIPLICATOR # src/test/resources/multiplication.feature:41
    Given the user john_snow sends 9 right attempts
                    # MultiplicationFeatureSteps.the_user_sends_attempts(String,int,String)
    When the user john_snow sends 1 right attempts
                    # MultiplicationFeatureSteps.the_user_sends_attempts(String,int,String)
    Then the user gets a response indicating the attempt is right
                    # MultiplicationFeatureSteps.the_user_gets_a_response_indicating_the_attemp
t_is(String)
    And the user gets 10 points for the attempt
                    # MultiplicationFeatureSteps.the_user_gets_points_for_the_attempt(int)
    And the user gets the BRONZE_MULTIPLICATOR badge
                    # MultiplicationFeatureSteps.the_user_gets_the_type_badge(String)

  Scenario Outline: The user sends a right attempt after 49 right attempts and then gets a badge
 SILVER_MULTIPLICATOR # src/test/resources/multiplication.feature:42
    Given the user john_snow sends 49 right attempts
                    # MultiplicationFeatureSteps.the_user_sends_attempts(String,int,String)
    When the user john_snow sends 1 right attempts
                    # MultiplicationFeatureSteps.the_user_sends_attempts(String,int,String)
    Then the user gets a response indicating the attempt is right
                    # MultiplicationFeatureSteps.the_user_gets_a_response_indicating_the_attem
pt_is(String)
    And the user gets 10 points for the attempt
                    # MultiplicationFeatureSteps.the_user_gets_points_for_the_attempt(int)
    And the user gets the SILVER_MULTIPLICATOR badge
                    # MultiplicationFeatureSteps.the_user_gets_the_type_badge(String)

  Scenario Outline: The user sends a right attempt after 99 right attempts and then gets a badge
 GOLD_MULTIPLICATOR # src/test/resources/multiplication.feature:43
    Given the user john_snow sends 99 right attempts
                    # MultiplicationFeatureSteps.the_user_sends_attempts(String,int,String)
    When the user john_snow sends 1 right attempts
                    # MultiplicationFeatureSteps.the_user_sends_attempts(String,int,String)
    Then the user gets a response indicating the attempt is right
                    # MultiplicationFeatureSteps.the_user_gets_a_response_indicating_the_attempt
_is(String)
    And the user gets 10 points for the attempt
                    # MultiplicationFeatureSteps.the_user_gets_points_for_the_attempt(int)
    And the user gets the GOLD_MULTIPLICATOR badge
                    # MultiplicationFeatureSteps.the_user_gets_the_type_badge(String)

6 Scenarios (6 passed)
29 Steps (29 passed)
0m14,862s

Tests run: 35, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 14.947 sec

Results :

Tests run: 47, Failures: 0, Errors: 0, Skipped: 0
```

Figura 6-2. *Salida bastante impresa de la ejecución de la prueba en Cucumber*

Además de eso, y gracias a la @PepinoOpciones anotación que usamos para nuestras clases de prueba principales, tendrá un informe Cucumber e informes XML JUnit (seguro) en el objetivo carpeta. Se pueden publicar en el sistema de CI para que pueda ver los resultados desde un lugar centralizado.

Resumen

En este capítulo, vio la importancia de tener buenos conjuntos de pruebas en un sistema distribuido. Todas las capas son importantes, pero elegir el enfoque integral puede convertirse en una decisión difícil debido al mantenimiento y la complejidad que comúnmente requiere.

Viste cómo una combinación de Cucumber y la estructuración del proyecto de prueba en capas puede proporcionar una solución simple y poderosa para cubrir una estrategia de principio a fin. Cucumber proporciona un lenguaje amigable para las empresas para diseñar los casos de prueba, Gherkin, y se integra con Java, por lo que es una opción que se combina perfectamente con nuestro caso de uso.

Además, el capítulo prestó especial atención a algunos detalles que pueden facilitar el desarrollo: parametrización de pasos y reutilización, comprensión de cómo funciona Cucumber al crear instancias de pruebas y cómo beneficiarse de la inyección de dependencia. Y, lo que es más importante, vio cómo hacer que su código sea comprobable hace que todo sea más simple simplemente agregando algo de lógica opcional a sus servicios.

Lo que obtuviste aquí también es un gran logro. Ahora puede probar el sistema de un extremo a otro, no solo verificando que cada parte funcione por sí sola, sino también que los casos comerciales generales tengan sentido. Tener un marco como este dará sus frutos aún más a medida que el proyecto madure, porque proporciona una capa adicional de estabilidad que es difícil de obtener de otra manera.

APÉNDICE A

Actualización a Spring

Arranque 2.0

Introducción

Este libro utiliza Spring Boot versión 1.5.7 para los diferentes microservicios que forman parte de la aplicación en evolución. Spring Boot 2.0, que se lanzará después de escribir este libro, presenta algunos cambios importantes en el sistema, ya que es una actualización importante.

Este apéndice le ayuda a actualizar sus aplicaciones a Spring Boot 2.0 en caso de que quieras usarlo.

CÓDIGO FUENTE DISPONIBLE CON EL LIBRO: V10

El repositorio v10 en GitHub (ver https://github.com/ microservices-practice) contiene el código del sistema, trabajando con Spring Boot 2.0. Funciona exactamente de la misma manera que la versión anterior (v9), solo que con la nueva versión. Todos los cambios realizados en la aplicación se etiquetan con un comentario que comienza conBOOT2 para que pueda encontrarlos fácilmente realizando una búsqueda de texto.

Tenga en cuenta que, dado que no existe una versión oficial al momento de escribir el libro, estamos usando una versión de hito: 2.0.0.M2.

M. Macero, *LearnMicroservices con Spring Boot,*
https://doi.org/10.1007/978-1-4842-3165-4

Actualizar las dependencias

Si desea utilizar la última versión de Spring Boot, debe cambiar su pom.xml archivos. Tenga en cuenta que también necesita actualizar la versión de lanzamiento de Spring Cloud, pero no sigue exactamente el mismo cronograma que Spring Boot, se demora algunas semanas.

Por lo tanto, el plan que sigue este apéndice es incluir el Spring Boot 2.0Hito 2 (2.0.0.M2), ya que tiene una versión de trabajo equivalente para las primeras versiones de Spring Cloud: Finchley.M2. Para poder utilizar versiones de hitos en sus proyectos, también debe agregar a su pom.xml los repositorios de Spring adicionales donde residen estas versiones.

Listado A-1 muestra el archivo del microservicio de multiplicación como referencia; debe aplicar estos mismos cambios a todas las demás aplicaciones de Spring Boot (actualizar las versiones de Spring Boot y Spring Cloud y agregar elrepositorios y pluginRepositories bloques).

Listado A-1. pom.xml (multiplicación v10)

<? xml versión = "1.0" codificación = "UTF-8"?>
<proyecto xmlns = "http://maven.apache.org/POM/4.0.0" xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance" xsi: schemaLocation = "http: // maven .apache.org / POM / 4.0.0 http: // maven.apache.org/xsd/ maven-4.0.0.xsd ">

<modelVersion>4.0.0 </**modelVersion>**

<groupId>microservices.book </**groupId> <artifactId>** multiplicación-social-v10 </**artifactId> <versión>**0.10.0-INSTANTÁNEA </**versión> <embalaje>**jar </**embalaje>**

<nombre>multiplicación-social-v10 </**nombre> <descripción>**Aplicación de multiplicación social (Aprenda microservicios con Spring Boot) </**descripción>**

```
<padre>
    <groupId>org.springframework.boot </groupId>
  <artifactId>Spring-boot-starter-parent </artifactId>
  <versión>2.0.0.M2 </versión> </parent>


<propiedades>
    <project.build.sourceEncoding>UTF-8 </proyecto.build.
    sourceEncoding>
    <project.reporting.outputEncoding>UTF-8 </proyecto.
    reporting.outputEncoding>
    <versión java>1,8 </java.version> <spring-cloud.version>
    Finchley.M2 </primavera-nube. versión>

</properties>

<gestión de dependencias>
    <dependencias>
        <dependencia>
            <groupId>org.springframework.cloud </groupId>
            <artifactId>dependencias de la nube de primavera
            </artifactId>
            <versión> $ {spring-cloud.version} </versión>
            <tipo>pom </tipo>
            <alcance>importar </
        alcance> </dependencia>
    </dependencias>
</dependencyManagement>

<dependencias>
<! - Todas nuestras dependencias existentes ... ->
</dependencias>
```

```
<build>
    <plugins>
        <enchufe>
            <groupId>org.springframework.boot </groupId>
            <artifactId>Spring-boot-maven-plugin
            </artifactId>
        </plugin>
    </plugins>
</build>

<repositorios>
    <repositorio>
        <id>primavera-instantáneas </id> <nombre>Instantáneas
        de primavera </nombre> <url>https://repo.spring.io/
        snapshot</url> <instantáneas>

            <habilitado>verdadero </
        habilitado> </snapshots>
    </repository>
    <repositorio>
        <id>hitos de primavera </id> <nombre>Hitos de
        primavera </nombre> <url>https://repo.spring.io/
        milestone</url> <instantáneas>

            <habilitado>falso </habilitado>
        </snapshots>
    </repository>
</repositorios>

<pluginRepositories>
    <pluginRepository>
        <id>primavera-instantáneas </id> <nombre>
        Instantáneas de primavera </nombre>
```

```
            <url>https://repo.spring.io/snapshot</url>
            <instantáneas>
                <habilitado>verdadero </
            habilitado> </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>hitos de primavera </id> <nombre>Hitos de
            primavera </nombre> <url>https://repo.spring.io/
            milestone</url> <instantáneas>

                <habilitado>falso </habilitado>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>

</proyecto>
```

Arreglando los cambios importantes

La interfaz CrudRepository no incluye findOne ()

El código del libro usa el Encuentra uno() método incluido en el CrudRepository interfaz para recuperar una sola entidad de la base de datos usando Spring Data JPA. Por alguna razón (la documentación aún no está disponible en el momento de escribir este artículo), ese método ha sido eliminado en una versión anterior a la incluida por Spring Boot starter, que usa la versión2.0.0.M4 de spring-data-commonspuedes verlo corriendo dependencia de mvnw: árbol desde cualquiera de las carpetas raíz del proyecto).

Eso no tiene un gran impacto en el código, pero necesita cambiar estas referencias de método para hacer que todo se compile nuevamente. El código

usos Encuentra uno() en tres clases dentro del proyecto de multiplicación: MultiplicationServiceImpl, UserController, y UserControllerTest.

Siga la misma estrategia para reemplazarlo en esos tres lugares: cambiarlo al preferido findById () método y usando el devuelto Opcional lanzar un Argumento de excepción ilegal si el identificador no existe en su base de datos. ListadoA-2 muestra uno de los fragmentos de código como referencia.

Listado A-2. MultiplicationServiceImpl.java (multiplicación v10)

```
@Anular
public MultiplicationResultAttempt getResultById (final Long
resultId) {
    // BOOT2: cambiado de findOne
    return intentoRepository.findById (resultId)
            . orElseThrow (() -> new IllegalArgumentException
                    ("El resultId solicitado [" + resultId +
                            "] no existe."));
}
```

Los puntos finales del actuador se han movido

Sabemos por las notas de la versión[1] que en Spring Boot 2.0, los puntos finales del actuador se han movido debajo de /solicitud. Eso significa que debe cambiar la configuración de la estrategia de equilibrio de carga dentro de la puerta de enlace API para apuntar a /aplicación / salud en lugar de solo /salud. Si no cambia esto, Ribbon pensará que todos los servicios están inactivos y API Gateway no funcionará. Ver listadoA-3.

[1] https://tpd.io/bootm1-rn

Listado A-3. RibbonConfiguration.java (puerta de enlace v10)

```
@Frijol
ribbonPing de IP pública (configuración final de IClientConfig) {
    // BOOT2: cambiado de / health a / application / health
    return new PingUrl (false, "/ application / health");
}
```

Aplicar actualizaciones opcionales

La clase WebMvcConfigurerAdapter ha quedado obsoleta

El código actual usa la clase WebMvcConfigurerAdapter para deshabilitar CORS en los microservicios de multiplicación, gamificación y puerta de enlace. Esa clase se ha desaprobado a favor de laWebMvcConfigurer interfaz, desde que comenzaron con Java 8, las interfaces pueden incluir implementaciones predeterminadas y Spring Boot 2.0 ya no es compatible con versiones anteriores de Java.

Este cambio es opcional aquí, ya que está obsoleto, pero es mejor adaptarse al cambio lo antes posible. ListadoA-4 muestra el cambio en una de las clases; asegúrese de aplicar este cambio a las tres clases (dentro de los proyectos de microservicios antes mencionados).

Listado A-4. WebConfiguration.java (puerta de enlace v10)

```
@Configuración
@EnableWebMvc
// BOOT2 cambió a la interfaz WebMvcConfigurer en lugar de la
subclase de WebMvcConfigurerAdapter
WebConfiguration de clase pública implementa WebMvcConfigurer
{// ...}
```

Trabajando con Spring Boot 2.0

Esos son todos los cambios que debe aplicar para que el sistema funcione con Spring Boot 2.0. Tenga en cuenta que no pude probar la versión de lanzamiento, por lo que podría suceder (aunque es poco probable) que haya otros cambios importantes por venir, no cubiertos por este apéndice. Si ese es el caso, puede visitar la página de notas de la versión de Spring Boot 2.0 para verificar. (Verhttps://github.com/ spring-projects / spring-boot / wiki / Spring-Boot-2.0-Release-Notes.)

En esa misma página, también encontrará las nuevas funciones que vienen con Spring Boot 2.0. Algunas de las principales funcionalidades nuevas son el soporte para Java 9 y algunas características excelentes en Spring 5, como el marco ReactiveWeb (SpringWebFlux). ¡Échales un vistazo y sigue aprendiendo!

Epílogo

En este libro, cubrimos los principales temas relacionados con la arquitectura de microservicios. Empezamos con una mirada*adentro* una aplicación Spring Boot, viajando desde un proyecto vacío a un microservicio estructurado adecuadamente en capas. Para construirlo, seguimos un enfoque de desarrollo basado en pruebas.

El libro trató de explicar desde el principio por qué es una buena idea comenzar con un pequeño monolito. En realidad, es una idea respaldada por muchas personas con mucha experiencia en microservicios: comience con un solo proyecto, identifique los límites y decida si vale la pena dividir su funcionalidad. Lo que sucede con frecuencia es que es difícil comprender la*por qué* si nunca ha trabajado con microservicios y solo con aplicaciones monolíticas. Sin embargo, en este momento, estoy muy seguro de que comprende el dolor que podría sufrir si busca microservicios desde cero. Configurar el ecosistema sin tener una base sólida de conocimiento causará caos, como mínimo. Descubrimiento de servicios, enrutamiento, equilibrio de carga, comunicación entre servicios, manejo de errores: es importante que sepa lo que enfrentará en su camino*antes de* comienzas tu aventura con microservicios.

En esta aventura en particular, creaste una aplicación web para permitir a los usuarios practicar la multiplicación todos los días sin ninguna ayuda, para entrenar sus cerebros. Ese fue el primer microservicio, pero el verdadero desafío comenzó cuando presentamos el segundo: un servicio que reacciona a los eventos que suceden en nuestra lógica anterior y calcula la puntuación y asigna insignias para que la aplicación parezca un juego. En ese momento, el libro cubrió algunos conceptos básicos sobre*gamificación* y lo aplicó a nuestro sistema mediante un evento patrón.

M. Macero, *LearnMicroservices con Spring Boot,*
https://doi.org/10.1007/978-1-4842-3165-4

Luego profundizamos en algunos conceptos básicos de microservicios, incluido cómo pueden encontrarse entre sí con el descubrimiento de servicios y aplicarles equilibrio de carga cuando se escalan hacia arriba y hacia abajo, cómo enrutar desde el exterior a la parte correspondiente del sistema utilizando una puerta de enlace API, cómo proporcionar resiliencia utilizando el patrón de disyuntor, etc. Lo hicimos práctico, pero intenté poner el foco en los conceptos también. La razón es que hoy en día puede encontrar estos patrones integrados en muchos PaaS en la nube, incluidos Cloud Foundry, Google App Engine y Amazon AWS.

Algunas de esas plataformas administrarán el ecosistema de microservicios por usted: solo *empujar* su aplicación Spring Boot a la nube, establezca la cantidad de instancias y cómo deben enrutarse, y todo lo demás lo maneja la plataforma. *Fundición en la nube de Pivotal* incluso incluye las mismas herramientas que usamos (Spring Cloud Netflix). Sin embargo, siempre debe comprender lo que está haciendo. Llevar sus aplicaciones a la nube sin saber lo que sucede detrás de las cortinas es arriesgado. Si surgen errores, es posible que le resulte imposible saber qué parte de*la magia*
(algo que no entiendes) no está haciendo su trabajo.

Finalmente, destaqué la importancia de tener una buena base de pruebas en un sistema basado en microservicios. Viste cómo implementar escenarios de un extremo a otro para proteger los flujos de tu negocio usando Cucumber y su lenguaje amigable para los humanos, Gherkin.

Este libro no cubre *todo* necesita saber trabajar con microservicios. Eso sería imposible. Hay algunos otros temas importantes en torno a este tipo de arquitectura: contenedorización, registro centralizado, integración y despliegue continuos, etc. Le recomiendo que continúe aprendiendo estos temas siguiendo el mismo enfoque práctico. Puede seguir evolucionando la arquitectura construida en este libro, enfocándose en los problemas que desea resolver y aprendiendo conceptos de manera incremental, uno por uno.

Espero que, ahora que ha llegado al final del libro, comprenda mejor los temas y pueda utilizarlos en el trabajo o en sus proyectos personales. He disfrutado mucho escribiendo este libro y, lo más importante, he aprendido a lo largo del camino. Gracias.

Índice

M. Macero, *LearnMicroservices con Spring Boot,*
https://doi.org/10.1007/978-1-4842-3165-4

mi

F

GRAMO

HOLA

J, K

L

M, N, O

P, Q

R

S

T

U, V

W, X, Y

Z