API TESTING WORKSHOP

http://sli.do | C331

Diana Zaharia

@dian4sz

Danut Turta

Robert Romaniuc

Andrei Serdenciuc

WHO ARE YOU PEOPLE?

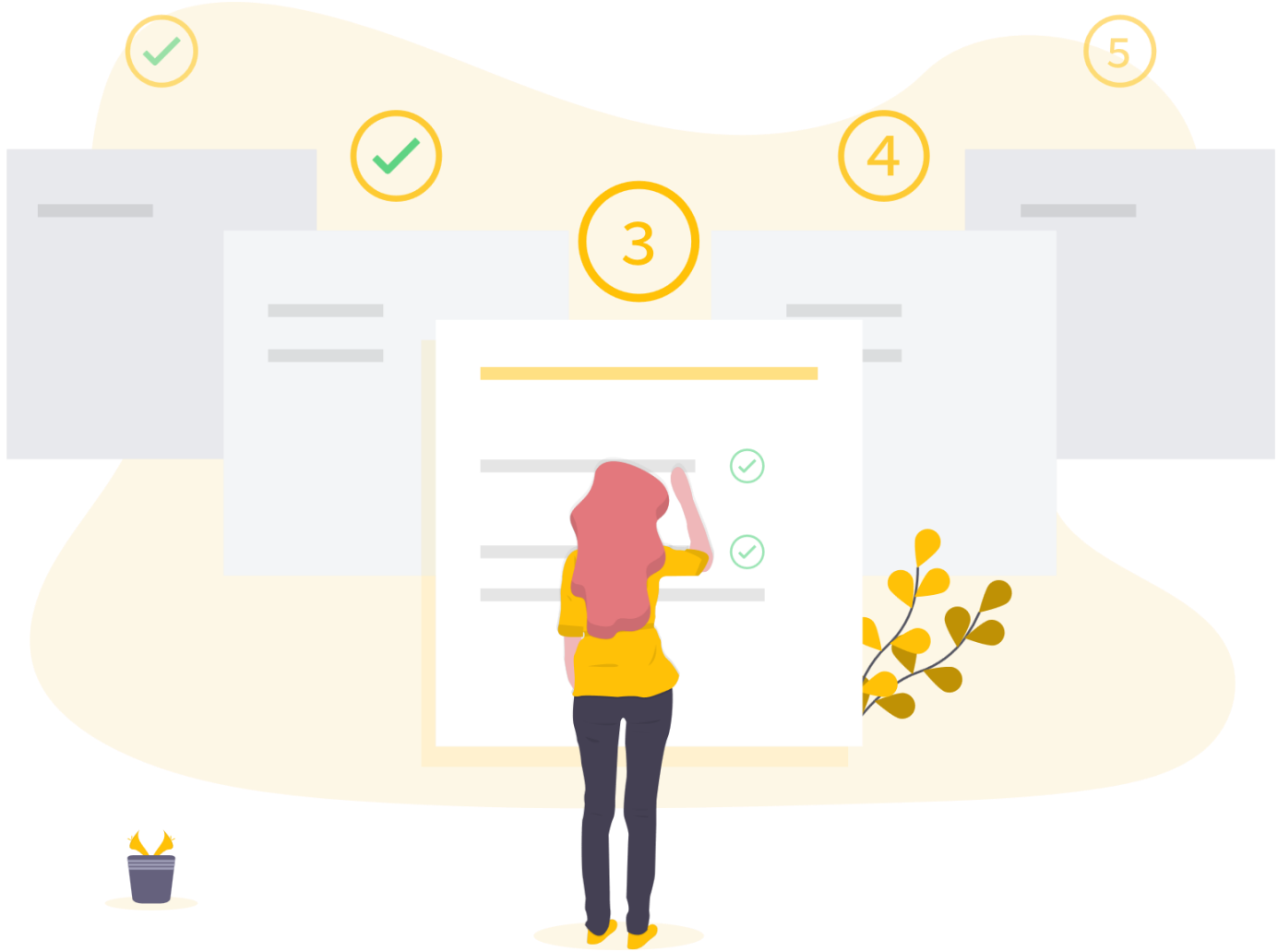
STUFF WE'LL COVER

RestAssured

- Making requests
- Asserting responses
- Reusing specifications

Junit 5

- Annotations
- Parameterized tests



REST ASSURED

Is a **Java Specific Language API** for simplifying testing of RESTful web services.

It can be used to invoke REST web services and match response content to test them.
It can be used to test **XML** as well as **JSON** based web services.

REST Assured can be integrated with **JUnit** and **TestNG** frameworks for writing test
cases for our application.

Supports **POST**, **GET**, **PUT**, **DELETE**, **OPTIONS**, **PATCH**, and **HEAD** requests and can
be used to validate and verify the response of these requests

GETTING IT TO WORK

```
@Test
public void testMyStuff() {
        given().
                param("x", "y").
        when().
                get("http://somesite.com/resources/users/{x}").
        then().
                statusCode(200).
                body(“id", equalTo(5));}
```

GIMME SOME SUGAR

```
given()
        .param("x", "y")
        .and()
        .header("z", "w")
.when()
        .get("/something")
.then()
        .assertThat()
        .statusCode(200)
        .and()
        .body("y", equalTo("z"))
```

WHAT'S ALL THIS?

Assertions

Status Code

```
get("/x").then().assertThat().statusCode(200)
```

Cookies

```
get("/x").then().assertThat().cookie("cookieName", "cookieValue")
```

Headers

```
get("/x").then().assertThat().header("Content-Length", equalTo("4567"))
```

Content-Type

```
get("/x").then().assertThat().contentType(ContentType.JSON)
```

Full body/content matching

```
get("/x").then().assertThat().body(equalTo("something"))
```

LET'S MAKE SOME SENSE OF IT

Hamcrest Matchers org.hamcrest.Matchers.*

- **The response body contains the string "winning-numbers"**

```
get("/lotto").then().assertThat()
    .body(containsString("winning-numbers"));
```

- **The response body contains the string "winning-numbers" and "winners"**

```
get("/lotto").then().assertThat()
    .body(containsString("winning-numbers"), containsString("winners"));
```

- **The lottoId is equal to 5**

```
get("/lotto").then().assertThat()
    .body("lotto.lottoId", equalTo(5));
```

- **One of the the winning numbers is 45**

```
get("/lotto").then().assertThat()
     .body("lotto.winning-numbers", hasItem(45));
```

- **Multiple validations**

```
get("/lotto").then().assertThat()
    .body("lotto.lottoId", equalTo(5), "lotto.winning-numbers",hasItem(45));
```

JSON BODY


```
{
"lotto":{
   "lottoId":5,
   "winning-numbers":[2,45,34,23,7,5,3],
   "winners":[{
      "winnerId":23,
      "numbers":[2,45,34,23,3,5]
      },{
      "winnerId":54,
      "numbers":[52,3,12,11,18,22]
      }]
   }
}
```

ON THE RIGHT PATH

Using Root Path

The long path

```
get(..).then().assertThat()
   .body("data.user1.userId", is(..),
   "data.user1.href", is(..));
```

The shortcut

```
RestAssured.rootPath = "data.user1";
get(..).then().assertThat()
   .body("userId", is(..))
   .body("href", is(..));
```

Using Path Arguments

Example

```
get(..).then().assertThat()
   .root("data.user1.%s")
   .body(withArgs("userId"),equalTo(..))
   .body(withArgs("href"), equalTo(..));
```

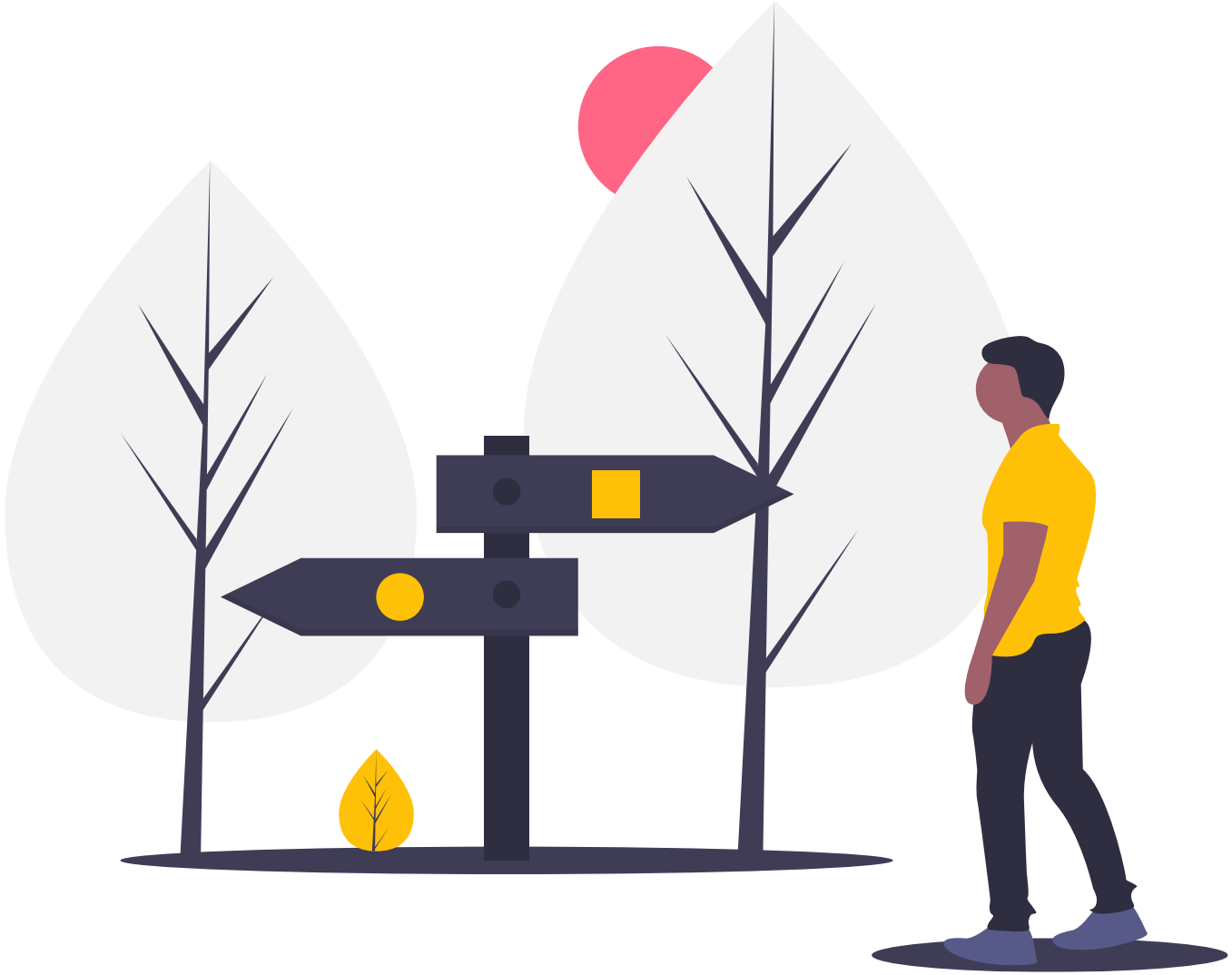
Wait, there's more

```
String subPath = "winners.winnerid";
int index = 1;
get("/x").then()
   .body("loto.%s[%d]",
      withArgs(subPath, index),
      equalTo("some value")).
```

JSON BODY

```
{
   "data" : {
      "user1" : {
         "userId" : "my-id1",
         "href":"http://localhost:8080/my-id1"
         },
      "user2" : {
         "userId" : "my-id2",
         "href":"http://localhost:8080/my-id2"
          }
      }
}
```

LET’S SEE IF THIS WORKS



THE FOUNDATION

RestAssured.**baseURI** = "https://somesite.com"; (Default : localhost)
RestAssured.**port** = 443; (Default : 8080)
RestAssured.**basePath** = "/resources"

get("/users");

is equivalent to

get("http://somesite.com:443/resource/users");

WHY IS IT DOING THAT?

Request logs

```
given().log().all(). ..
              .params(). ..
              .body(). ..
              .headers(). ..
              .cookies(). ..
              .method(). ..
              .path(). ..
```

Response logs

```
get("/x").then().log().all(). ..
                      .body(). ..
                      .statusLine(). ..
                      .headers(). ..
                      .cookies(). ..
                      .ifError(). ..
                      .ifStatusCodeIsEqualTo(400).
```

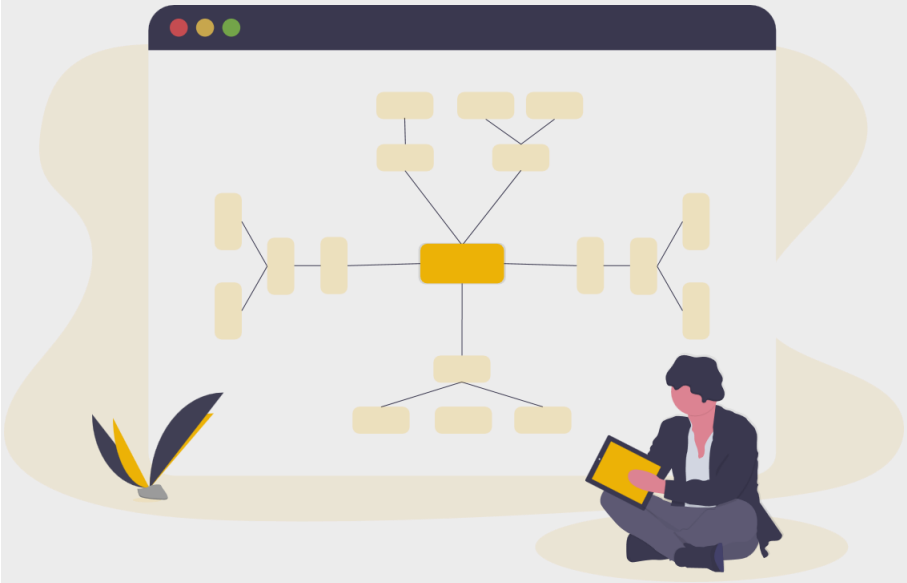
MORE WORK?

?PARAMETERS

Query Parameters

Example:
http://somesite.com/?text=test

"text" is a query parameter
(with value "test").



We can use queryParam(parameter-name, value) to specify a query parameter:

```
given().queryParam("q", "john")
.when().get("/search/users")
.then().statusCode(200);
```

For adding multiple query parameters, we can either chain several queryParam() methods, or add the parameters to a queryParams() method:

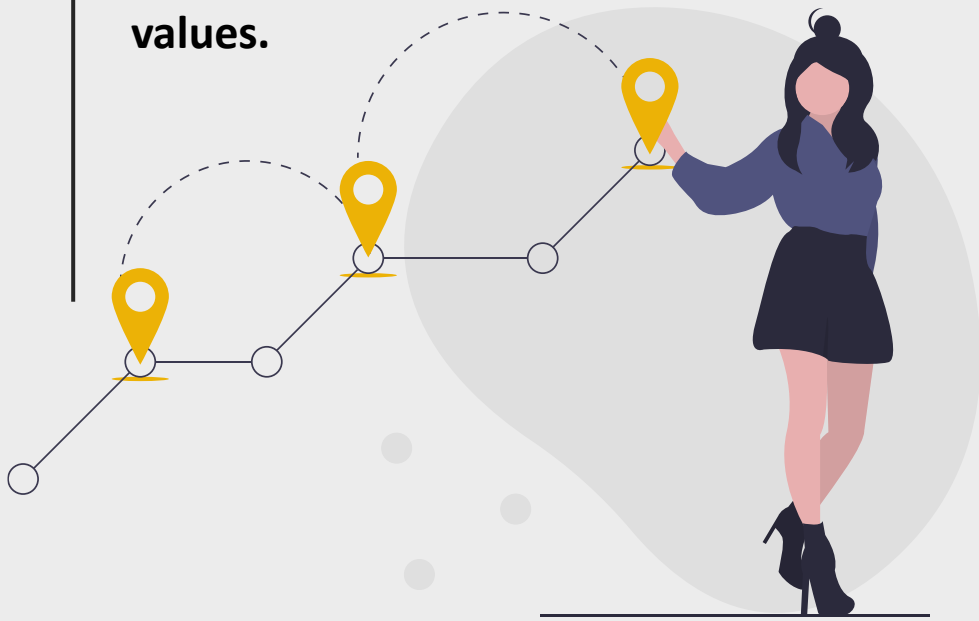
```
int perPage = 20;
given().queryParam("q", "john").queryParam("per_page",perPage)
.when().get("/search/users").then().statusCode(200);

given().queryParams("q", "john","per_page",perPage)
.when().get("/search/users") .then().statusCode(200);
```

/PARAMETERS

```
given()
        .log().all()
        .pathParam("owner", "john")
        .pathParam("repo", "api")
.when()
        .get("/repos/{owner}/{repo}")
.then()
        .log().all()
        .statusCode(200);
```

Path Parameters

Path parameters makes it easier to read the request path as well as enabling the request path to easily be re-usable in many tests with different parameter values.



EVEN MORE PARAMETERS

Form Parameters

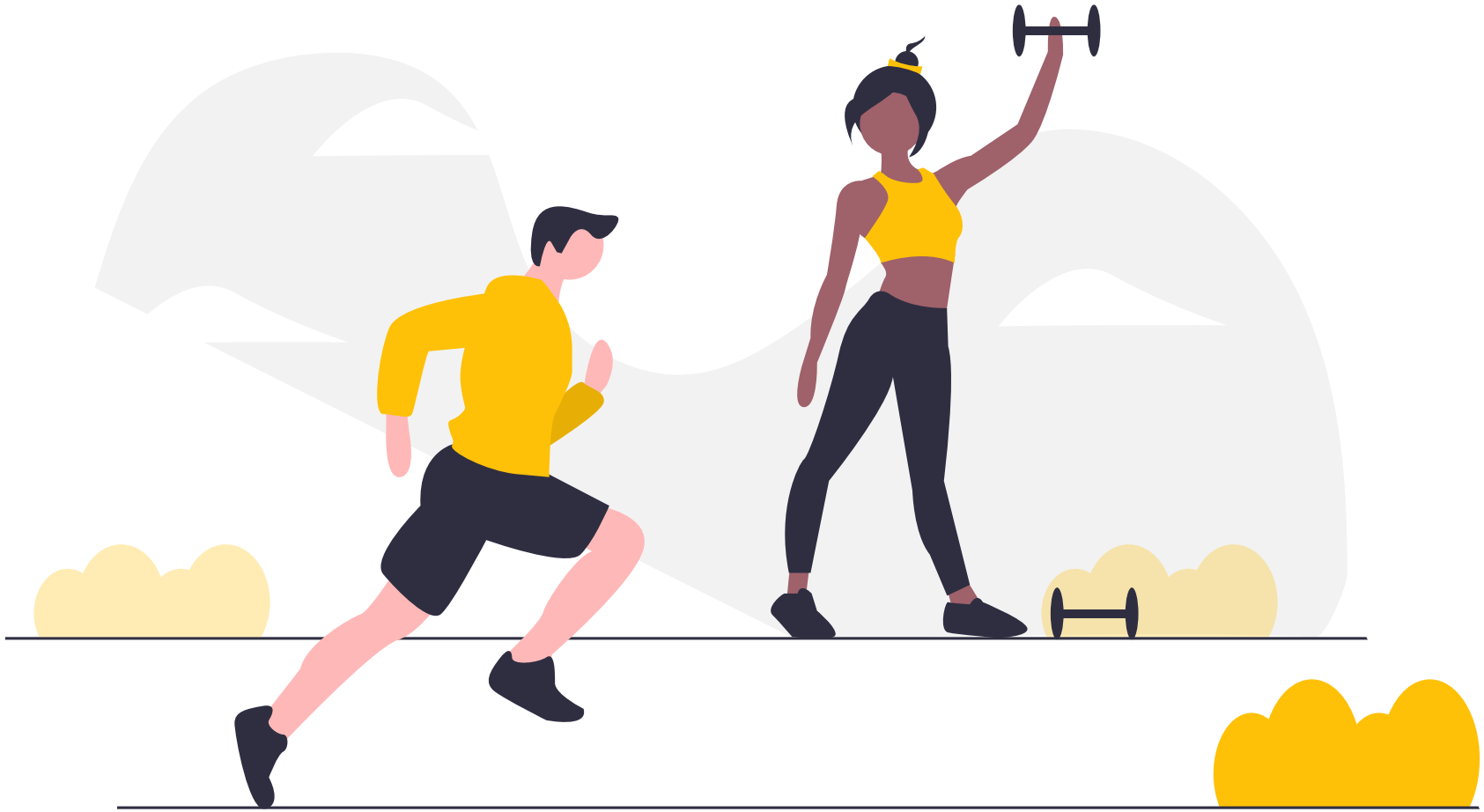
We can specify form parameters using formParam():

```
given().formParams("username", “john","password","1234")
                .post("/");
```

The param() method will act like formParam() for POST requests and will act like queryParam() with GET requests.

```
given().params("username", "john","password","1234")
.when().post("/");

given().param("q", "john")
.when().get("/search/users")
```

EXERCISE TIME



LET'S BE SPECIFIC – Request specification

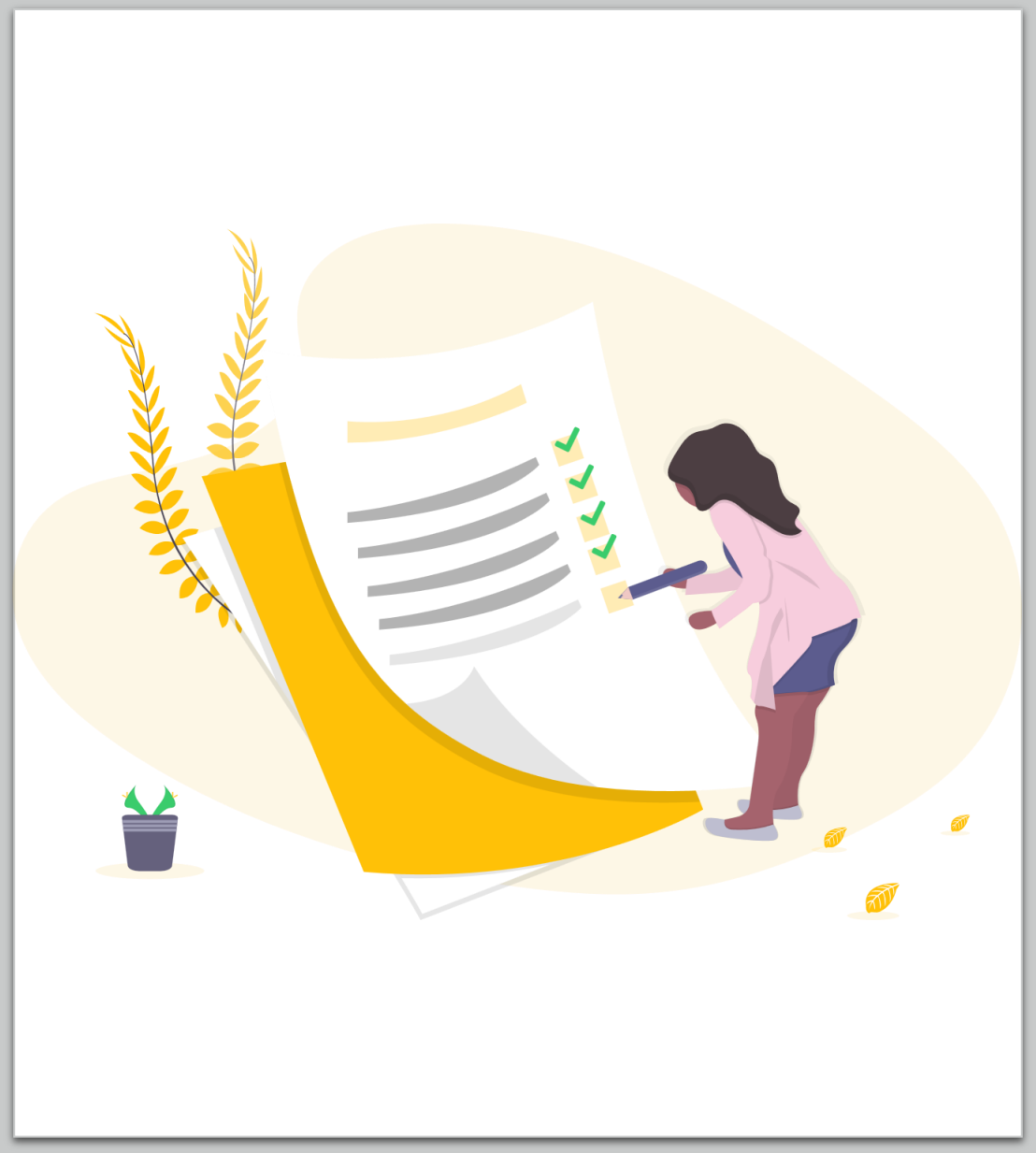
Reusing request data with RequestSpecBuilder

```
RequestSpecBuilder builder = new RequestSpecBuilder();
builder.setBaseUri("http://somesite.com")
builder.setBasePath("/resources")
builder.addParam("parameter1", "parameterValue");
builder.addHeader("header1", "headerValue");
RequestSpecification requestSpec = builder.build();


given().
        spec(requestSpec).
        param("parameter2", "paramValue").
when().
        get("/something").
then().
        statusCode(200);
```

LET’S BE SPECIFIC – Response specification

Reusing checks with ResponseSpecBuilder

```
ResponseSpecification responseSpec =
        new ResponseSpecBuilder()
                .expectStatusCode(200)
                . expectContentType(ContentType.JSON)
                .build();


given()
.when()
        .get("http://somesite.com/resources")
.then()
        .assertThat()
        .spec(responseSpec)
        .and()
        .body("size()",is(20))
```



MORE WRITING

OBJECT MODELLING

JSON Body

```
{
   "car":{
      "maker": "Aston Martin",
      "model": "DB9",
      "year": 2004
   }
}
```

Java Model

```
public class Car {
   String maker;
   String model;
   int year;

   public Car(String maker, String model,
int year) {
      this.maker = maker;
      this.model = model;
      this.year = year;
   }

   public int getYear() { return this.year;}

   public String getModel() {return this.
model;}
}
```

OBJECT MODELLING

Response modeling

```
Car car = given()
    .when()
        .get("http://somesite.com/resources")
        .as(Car.class);

assertEquals(2004, car.getYear());
assertEquals("DB11", car.getModel());
```

Request using model

```
Car car = new Car("Aston Martin", "DB9",
2004);
given()
    .contentType("application/json")
    .body(car)
.when()
    .post("http://somesite.com/resources")
.then()
    .statusCode(201);
```

HAVING FUN YET?

JUnit [5] ADNOTATIONS

@Test - Denotes that a method is a test method.

```
@BeforeAll
static void setup() {
log.info("@BeforeAll - executes once before all test methods
in this class");}
```

```
@BeforeEach
void init() {
log.info("@BeforeEach - executes before each test method in
this class");}
```

```
@AfterEach
void tearDown() {
log.info("@AfterEach - executed after each test method.");}
```

```
@AfterAll
static void done() {
log.info("@AfterAll - executed after all test methods.");}
```

@DisplayName - Declares a custom display name for the test class or test method.
@DisplayName("Feature – x")

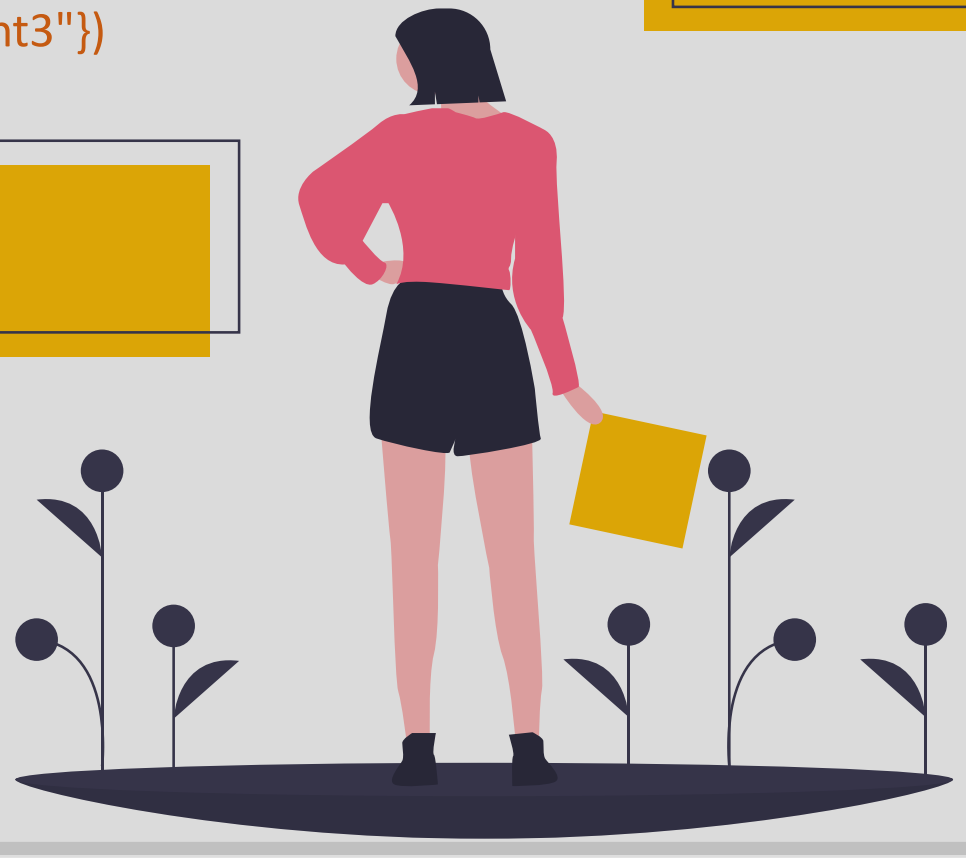
@Disabled - Used to disable a test class or test method; analogous to JUnit 4's @Ignore

@Tag - Used to declare tags for filtering tests, either at the class or method level;
@Tag("RunThis")

MORE ADNOTATIONS

MORE PARAMETERS?

```
@ParameterizedTest
@ValueSource(strings = {"endpoint1","endpoint2","endpoint3"})
public void dataDrivenTest(String endpoint) {
    given()
        .baseURI("http://somesite.com/resources")
    .when()
        .get(endpoint)
    .then()
        .statusCode(200);
}
```



PARAMETERS AGAIN

```
@ParameterizedTest
@MethodSource("continents")
public void dataDrivenTest(int id, String name) {
   given()
      .baseURI("http://somesite.com/continents")
      .pathParam("continentId", id)
   .when()
      .get("/{continentId}")
   .then()
      .body("name", equalTo(name));
}
```

```
public Stream<Arguments> continents() {
   return Stream.of(
      arguments(1, "Europe"),
      arguments(2, "Asia")
   );
}
```

Delete your tests

Feedback

https://forms.gle/JiQ9sYqB3pRM3cQU7