TAE week - 2018

Service testing

WTF Does This API Do?

Diego Montoya Lopera
Diego Berdugo Castañeda
Mauricio Rodriguez

Globant
we are ready

AGENDA
What to expect today

Background
Web services (ReST)
API Testing

Black box API Testing
REST Assured framework

White box API Testing
JOOQ - Framework

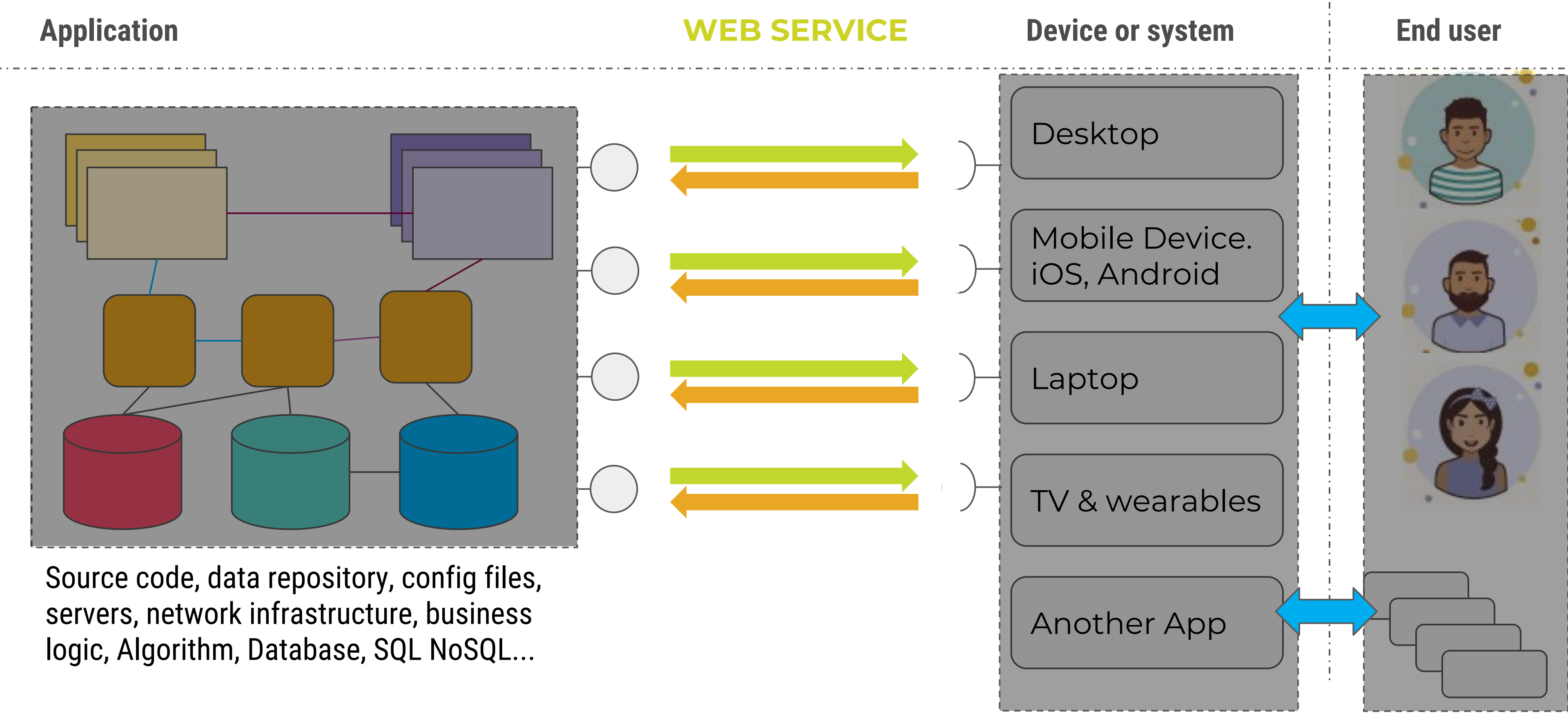
What is a web service?

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

A web service is not targeted at humans but rather at other programs.

Is not tied to any one operating system or programming language

Web services



Source code, data repository, config files, servers, network infrastructure, business logic, Algorithm, Database, SQL NoSQL...

ReST web services

Resources Based

ReST: "REpresentational State Transfer"

Usages

Systems integration. Data sharing. Don't reinvent the wheel.

XML and JSON

Used formats for data transfer.

HTTP Based

Client server. Uses HTTP methods. Uses HTTP message codes.

JSON FORMAT

```
{
   "title": "Deadpool",
   "releaseDate": "2016-02-08",
   "director": "Tim Miller",
   "starring": ["Ryan Reynols”, “Morena Baccarin”,”Skrein"]
}
```

HTTP

HTTP Methods

GET:
Obtain information

POST:
Create a resource

PUT:
Update existing resource

DELETE:
Delete a resource

PATCH:
Partially update a resource

HTTP Codes

1xx: Informational

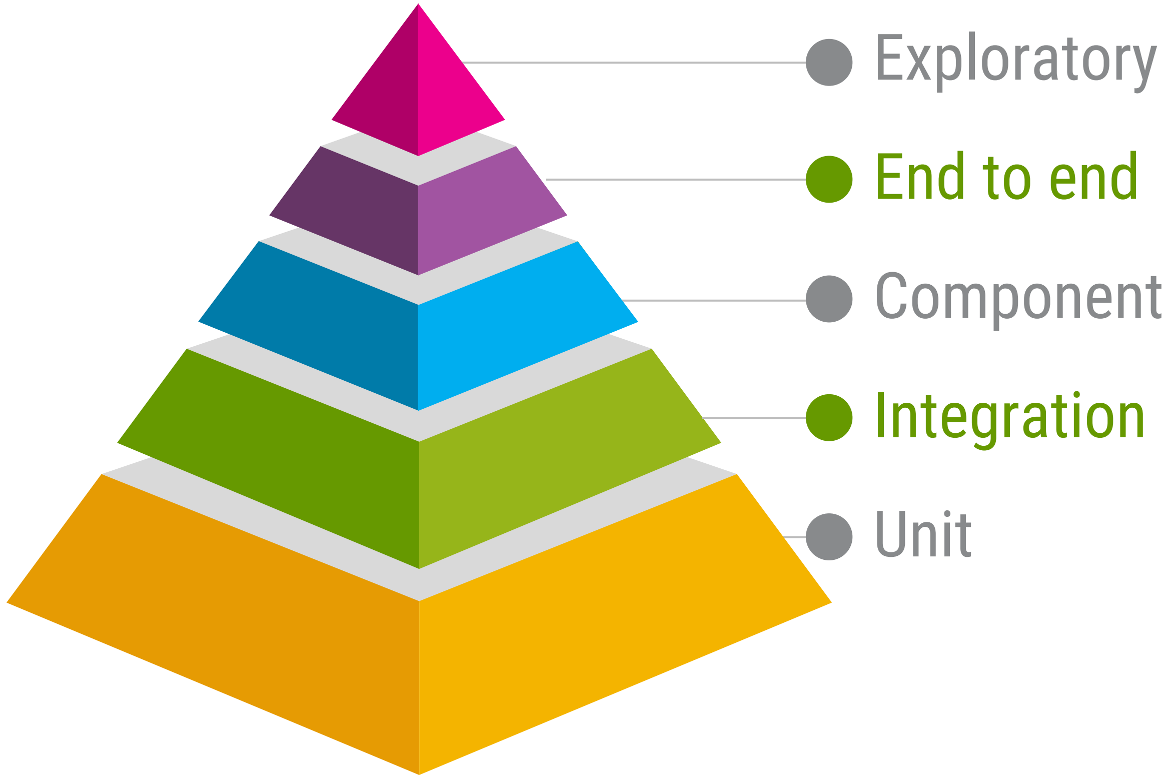
2xx: Success

3xx: Redirection

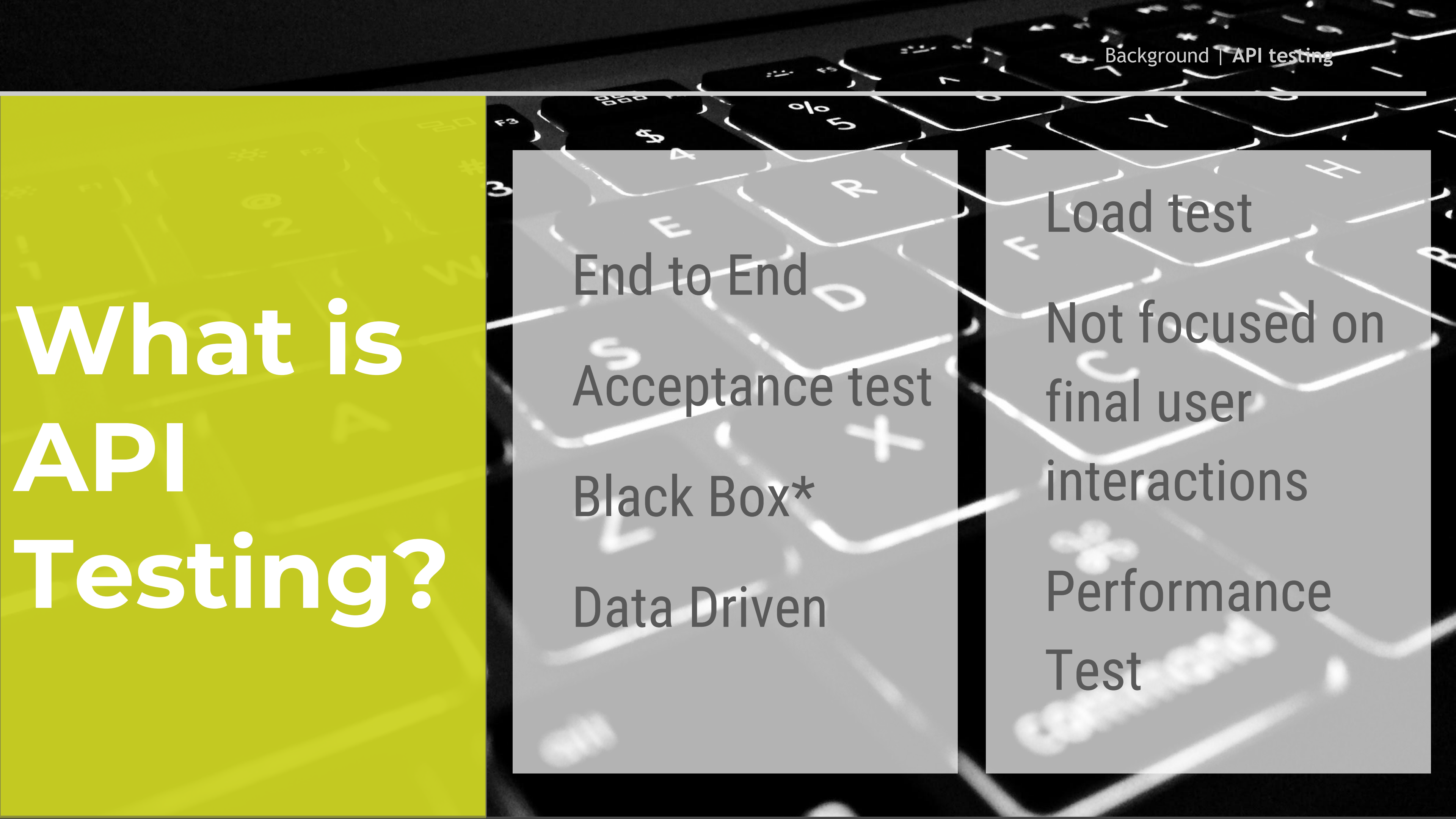
4xx: Client Error
(404: Not found)

5xx: Server Error

API Testing



https://martinfowler.com/articles/microservice-testing/#conclusion-test-pyramid



What is API Testing?

End to End

Acceptance test

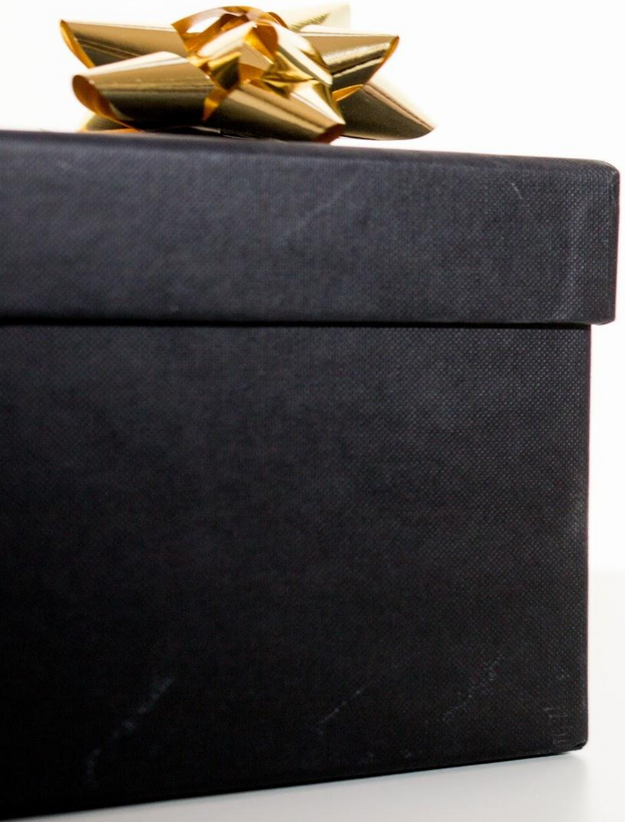
Black Box*

Data Driven

Load test

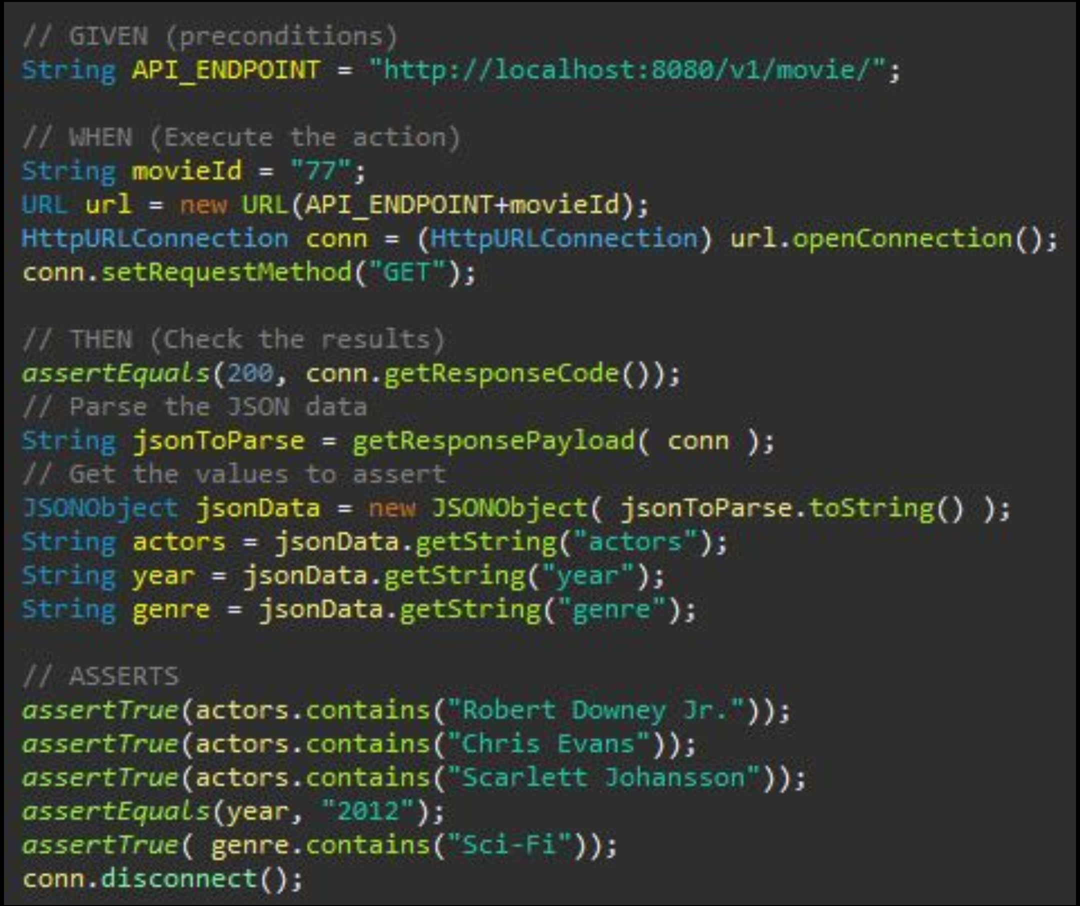
Not focused on final user interactions

Performance Test



Black box testing

Counter example (using java)



```java
// GIVEN (preconditions)
String API_ENDPOINT = "http://localhost:8080/v1/movie/";

// WHEN (Execute the action)
String movieId = "77";
URL url = new URL(API_ENDPOINT+movieId);
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");

// THEN (Check the results)
assertEquals(200, conn.getResponseCode());
// Parse the JSON data
String jsonToParse = getResponsePayload( conn );
// Get the values to assert
JSONObject jsonData = new JSONObject( jsonToParse.toString() );
String actors = jsonData.getString("actors");
String year = jsonData.getString("year");
String genre = jsonData.getString("genre");

// ASSERTS
assertTrue(actors.contains("Robert Downey Jr."));
assertTrue(actors.contains("Chris Evans"));
assertTrue(actors.contains("Scarlett Johansson"));
assertEquals(year, "2012");
assertTrue( genre.contains("Sci-Fi"));
conn.disconnect();
```

Given:
the movies API exposed

When:
I search the movie: Avengers

Then:
I expect to get:

Response code: 200

"Robert Downey Jr., Chris Evans, Scarlett Johansson" in the list of actors

2012 as the release year

Sci-Fi as the genre

Black box testing tools

SoapUI by SMARTBEAR

APACHE JMeter™

REST-assured

Assertible

POSTMAN

Katalon

apigee

REST-assured

REST Assured

What is it?

REST Assured is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs.

REST Assured is open source, which makes it easily accessible to everyone, therefore, becoming one of the most popular REST API validation tools.

Installation

Maven

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>3.0.2</version>
    <scope>test</scope>
</dependency>
```

Gradle

```
testCompile 'io.rest-assured:rest-assured:3.0.2'
```

REST Assured examples

EXAMPLES

GET User with RestAssured.

POST Create new user, validating the id and creation_date returned by the API

POST Create entity, GET: the new id and make a new API request to get the info.

DELETE the newly created entity

Performance GET all the movies.

Let’s code

So… Why should I use REST Assured?

Counter example

```
@Test
public void testGetUser() {
    try {
        String API_ENDPOINT = "http://localhost:8080/v1/user/";

        String userID = "67";

        URL url = new URL(API_ENDPOINT + userID);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Content-Type", "application/json");

        assertEquals(200, conn.getResponseCode());

        String jsonToParse = getResponsePayload(conn);

        JSONObject jsonData = new JSONObject(jsonToParse);
        String firstName = jsonData.getString("firstName");
        String lastName = jsonData.getString("lastName");
        String country = jsonData.getString("country");
        String email = jsonData.getString("email");

        // ASSERTS
        assertEquals(firstName, "Dorian");
        assertEquals(lastName, "McCrainor");
        assertEquals(country, "Indonesia");
        assertEquals(email, "dmccrainor1u@sina.com.cn");

        conn.disconnect();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

REST Assured

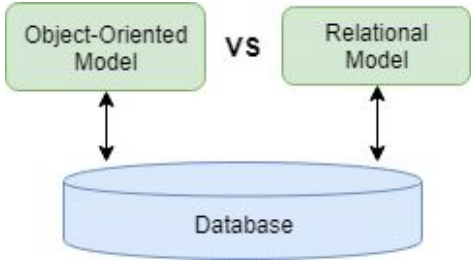
```
@Test
public void testGetUser() {
    Long idUser = 67L;
    String firstName = "Dorian";
    String lastName = "McCrainor";
    String country = "Indonesia";
    String email = "dmccrainor1u@sina.com.cn";

    GetUserResponseDTO userDTO = given(spec).when()
            .get("/user/{id}", idUser)
            .then().statusCode(200)
            .extract().as(GetUserResponseDTO.class);

    assertEquals(userDTO.getId(), idUser);
    assertEquals(userDTO.getFirstName(), firstName);
    assertEquals(userDTO.getLastName(), lastName);
    assertEquals(userDTO.getCountry(), country);
    assertEquals(userDTO.getEmail(), email);
    assertThat(userDTO.getCreateTime(), notNullValue());
}
```

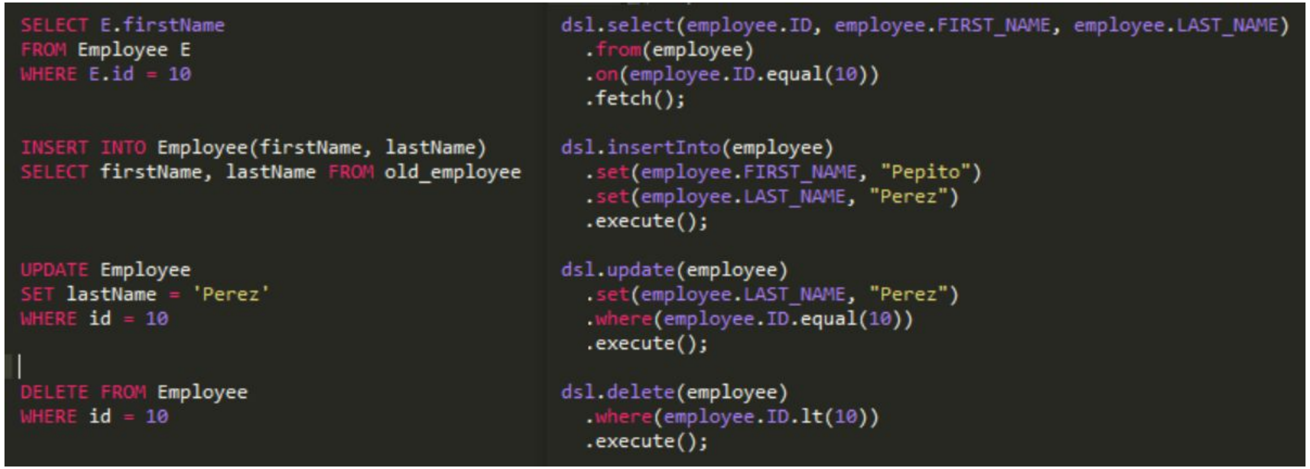
White box testing

JOOQ

What is it?

jOOQ generates Java code from your database
and lets you build type safe SQL queries through its fluent API





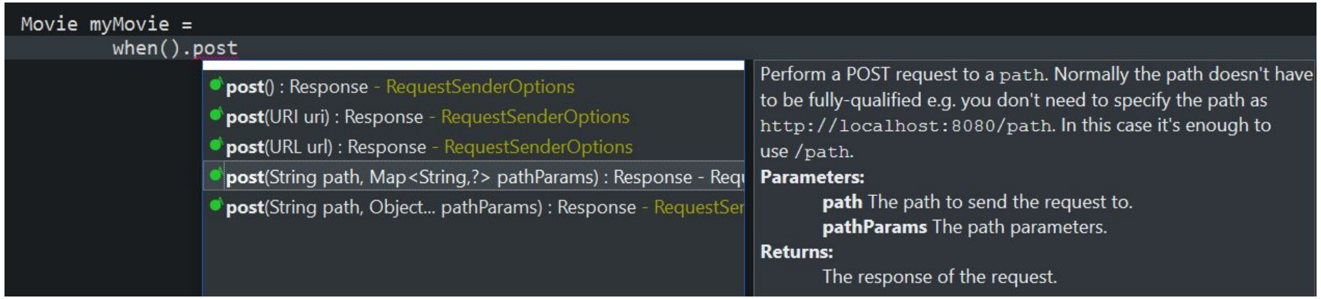
```
SELECT E.firstName
FROM Employee E
WHERE E.id = 10


INSERT INTO Employee(firstName, lastName)
SELECT firstName, lastName FROM old_employee



UPDATE Employee
SET lastName = 'Perez'
WHERE id = 10


DELETE FROM Employee
WHERE id = 10
```

```
dsl.select(employee.ID, employee.FIRST_NAME, employee.LAST_NAME)
  .from(employee)
  .on(employee.ID.equal(10))
  .fetch();

dsl.insertInto(employee)
  .set(employee.FIRST_NAME, "Pepito")
  .set(employee.LAST_NAME, "Perez")
  .execute();

dsl.update(employee)
  .set(employee.LAST_NAME, "Perez")
  .where(employee.ID.equal(10))
  .execute();

dsl.delete(employee)
  .where(employee.ID.lt(10))
  .execute();
```

JOOQ

Why use it?

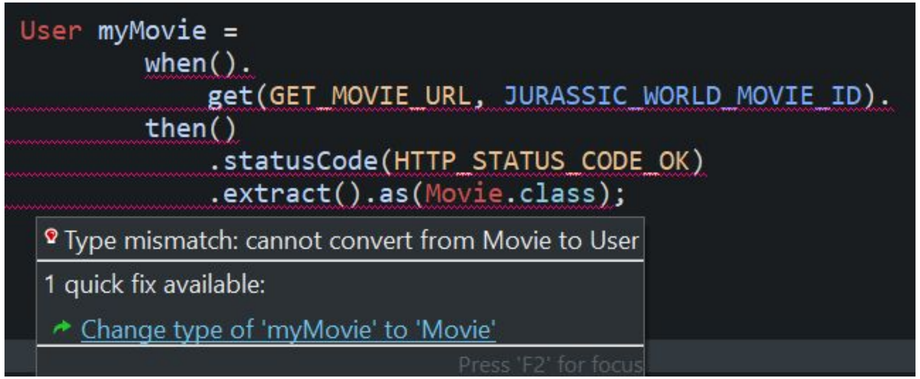
- Database First
- SQL centered
- Typesafe SQL
- Multi-Tenancy
- Standardization
- Query Lifecycle
- Procedures
- Code Generation
- Active Records



```
Movie myMovie =
        when().post
```

post() : Response - RequestSenderOptions
post(URI uri) : Response - RequestSenderOptions
post(URL url) : Response - RequestSenderOptions
post(String path, Map<String,?> pathParams) : Response - Requ
post(String path, Object... pathParams) : Response - RequestSer

Perform a POST request to a `path`. Normally the path doesn't have to be fully-qualified e.g. you don't need to specify the path as `http://localhost:8080/path`. In this case it's enough to use `/path`.
Parameters:
path The path to send the request to.
pathParams The path parameters.
Returns:
The response of the request.



```
User myMovie =
        when().
            get(GET_MOVIE_URL, JURASSIC_WORLD_MOVIE_ID).
        then()
            .statusCode(HTTP_STATUS_CODE_OK)
            .extract().as(Movie.class);
```

Type mismatch: cannot convert from Movie to User
1 quick fix available:
Change type of 'myMovie' to 'Movie'
Press 'F2' for focus

JOOQ

Editions

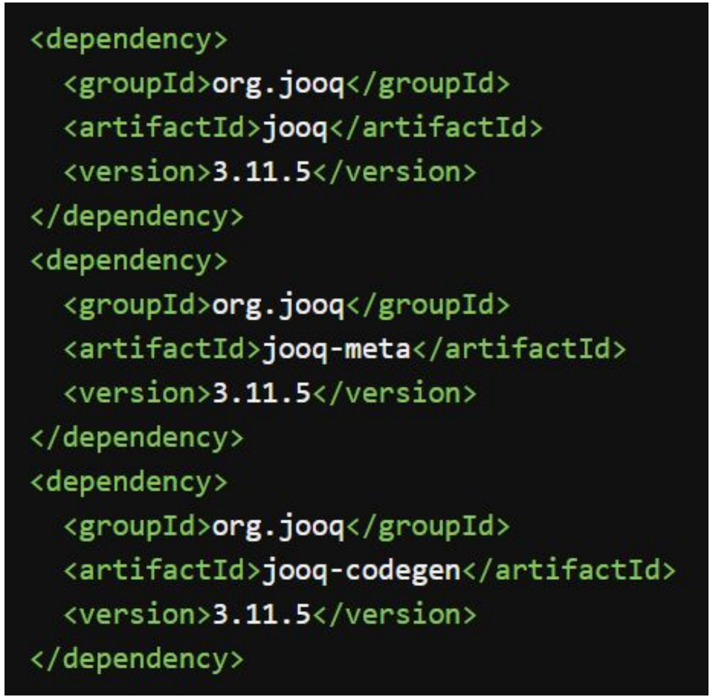
Free - Open Source

Paid - Express, Professional, Enterprise

Installation

Download ZIP

Maven



```xml
<dependency>
  <groupId>org.jooq</groupId>
  <artifactId>jooq</artifactId>
  <version>3.11.5</version>
</dependency>
<dependency>
  <groupId>org.jooq</groupId>
  <artifactId>jooq-meta</artifactId>
  <version>3.11.5</version>
</dependency>
<dependency>
  <groupId>org.jooq</groupId>
  <artifactId>jooq-codegen</artifactId>
  <version>3.11.5</version>
</dependency>
```

Let’s code

EXAMPLES

GET: comparison between the DB records and the API entities

POST: Create an entity and validate the insert
GET: the new id and query the DB.

DELETE: the entity create

Data driven test: Get multiple DB records and for each make request to the API validating the info

JOOQ

Generation Tool

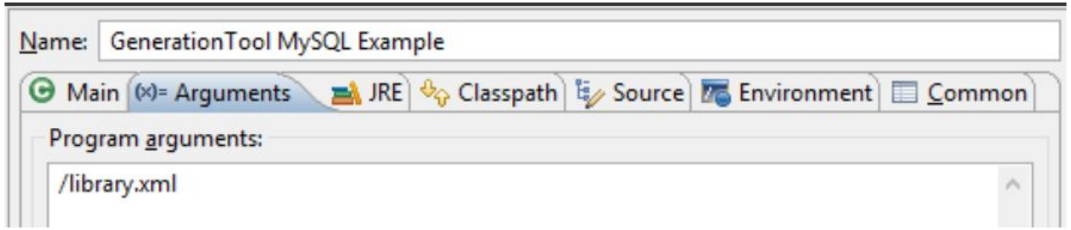
Command Line

```
C:\>java -cp jooq-3.11.5.jar;...;[JDBC-driver].jar;. org.jooq.codegen.GenerationTool /[XML file]
```

Maven

```
<plugin>
  <groupId>org.jooq</groupId>
  <artifactId>jooq-codegen-maven</artifactId>
  <version>3.11.5</version>

  <executions>
    <execution>
      <id>jooq-codegen</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>generate</goal>
```

Eclipse



Name: GenerationTool MySQL Example

Program arguments:

/library.xml

API Testing Recommendations

① Write as few end-to-end tests as possible

② Focus on personas and user journeys

③ Choose your ends wisely

④ Rely on infrastructure-as-code for repeatability

⑤ Make tests data-independent

https://martinfowler.com/articles/microservice-testing/#testing-end-to-end-tips

¿Questions?

THANKS!