JAVA MAGAZINE

Onafhankelijk tijdschrift voor de Java-professional

.nl. jug

02 > 2023

NLJUG 20 YEAR ANNIVERSARY!

Including brand new membership tiers

SEE PAGE 42

.nl. jug

›TEST AUTOMATION WITH ROBOT FRAMEWORK

›BOOST YOUR PRODUCTIVITY

›MIGRATE WITH OPENREWRITE

›JAVA 20 ALL THE NEW FEATURES


TIKKIE WON'T BE INNOVATIVE FOREVER
#STARTWITHUS
After successfully launching apps like Tikkie and Grip, it would be easy to say "Okay, we made it". But that's not how innovation works. Innovation is about constantly challenging yourself. Daring to acknowledge that something can always be better. Must be better. Because it's this mindset that makes the difference. And to make a difference we need you. Employees that will continue to motivate others, and us.
www.workingatabnamro.com
ABN·AMRO

{ COLOFON } JAVA MAGAZINE 02-2023

Content Manager & Project Coördinatie:
Stijn van de Blankevoort

Eindredactie:
Eveline Kroese

Auteurs:
Willem Cheizoo, Martin van Es, Erwin de Gier, Simone de Gijt, Martin Kanters, Maja Reißner, Gunter Rotsaert, Bert Jan Schrijver, Thamar Swart, Guus de Wit, Ivo Woltring

Redactiecommissie:
Stijn van de Blankevoort, Hanno Embregts, Julien Lengrand-Lambert, Helma Maassen, Elias Nogueira, Maja Reißner, Ivo Woltring, Thomas Zeeman

Vormgeving:
Wonderworks, Haarlem

Uitgever:
Martin Smelt

Traffic & Media order:
Marco Verhoog

Drukkerij:
Senefelder Misset, Doetinchem

Advertenties:
Richelle Bussenius
E-mail: richelle.bussenius@nljug.org
Telefoon: 023 752 39 22
Fax: 023 535 96 27
Marc Post
E-mail: mpost@reshift.nl
Telefoon: 023 543 00 08

Abonnementenadministratie:
Tanja Ekel

De prijs van een lidmaatschap van de NLJUG verschilt per membership tier die u afneemt. Een reguliere "base" membership van de NLJUG kost € 59,50 per jaar, waarbij Java Magazine gratis verschijnt. Naast het Java Magazine krijgt u gratis toegang tot de vele NLJUG workshops en het J-Fall congres. Het NLJUG is lid van het wereldwijde netwerk van JAVA user groups. Voor meer informatie of wilt u lid worden, **zie www.nljug.org**. Een nieuw lidmaatschap wordt gestart met de eerst mogelijke editie voor een bepaalde duur. Het lidmaatschap zal na de eerste (betalings)periode stilzwijgend worden omgezet naar lidmaatschap van onbepaalde duur, tenzij u uiterlijk één maand voor afloop van het initiële lidmaatschap schriftelijk (per brief of mail) opzegt. Na de omzetting voor onbepaalde duur kan op ieder moment schriftelijk worden opgezegd per wettelijk voorgeschreven termijn van 3 maanden.
Een lidmaatschap is alleen mogelijk in Nederland en België.
Uw opzegging ontvangen wij bij voorkeur telefonisch. U kunt de Klantenservice bereiken via: 023-5364401. Verder kunt u mailen naar **members@nljug.org** of schrijven naar NLJUG BV, Leden-administratie, Nijverheidsweg 18, 2031 CP Haarlem.
Verhuisberichten of bezorgklachten kunt u doorgeven via **members@nljug.org** (Klantenservice).

Wij nemen je gegevens, zoals naam, adres en telefoonnummer op in een gegevensbestand. De verwerking van uw gegevens voeren wij uit conform de bepalingen in de Algemene Verordening Gegevens-bescherming. De gegevens worden gebruikt voor de uitvoering van afgesloten overeenkomsten, zoals de abonnementenadministratie en, indien je daar toestemming voor hebt gegeven, om je op de hoogte te houden van interessante informatie en/of aanbiedingen. Je kunt uw persoonsgegevens opvragen om inzicht te krijgen in welke gegevens wij van je hebben, deze te corrigeren of, na beëin-diging van de abonnee-overeenkomst, te laten verwijderen. Stuur hiertoe een kaartje aan NLJUG BV, afd. klantenservice, Nijverheids-weg 18, 2031 CP Haarlem of een e-mail naar **members@nljug.org**.

VOORWOORD

20 years NLJUG

This year, it's our 20th anniversary. And when it's your birthday, you celebrate it with the ones you cherish. So come and join us at TEQnation (May 17th), J-Spring (June 21st) and J-Fall (November 9th)!

Furthermore, for most of us things change around our 20th birthday. It's the same for the NLJUG. No no, don't worry, no bad things, only good ones. We've introduced three new membership tiers: Base, Core, and Key. In a nutshell, our new tiers offer a range of benefits at different price points, making it easier than ever to choose the membership that fits your needs. As a regular member nothing changes for you, but would you like to know about the perks of upgrading to a higher level? Take a look at page number 42.

This year, twenty is not only a special number for a us. Because Java 20 has been released, which brings new and resubmitted Incubator and Preview features. Take a look at our Java version summary at page 10 to see what's it all about.

In this Java Magazine edition we'll touch upon various other subjects, such as: Integration tests with IntelliJ HTTP-client, co-creation between UX and software, migrating with OpenRewrite, Automating tests with Robot framework and more.

As you might've noticed, we are publishing more and more English written articles (don't worry, Dutch articles are also welcome 😊). So, do you have experience with a fun or interesting Java related tool and would you like to share your know-how? Don't hesitate and reach out to us via info@nljug.org. Even if it's your very first article. No worries, we would love to coach you!

See you soon at TEQnation and J-Spring!

V

Stijn van de Blankevoort
Content-/Communitymanager NLJUG
svdblankevoort@reshift.nl

SNOWFLAKE AND SNOWPARK: THE FUTURE OF THE CLOUD-BASED DATA PLATFORM

The exponential growth of data and the need to ingest, transform, and analyze it at scale present new challenges. Trying to make sense of this deluge of data using traditional technologies such as Hadoop, Hive, and Apache Spark creates challenges: often requiring complex infrastructure setup and maintenance, lack of scalability, data that ends up in silos, and security concerns.

By migrating certain Spark workloads to Snowflake, for example, organizations can take advantage of the Snowflake platform's near-zero maintenance, data-sharing capabilities, and built-in governance. With Snowpark, Snowflake's developer framework, data engineers, data scientists, and data developers can code in their familiar way with their language of choice, such as Java, and execute pipelines, ML workflows, and data apps faster and more securely, all in a single platform.

Snowflake: A Cloud-Based Data Platform

Snowflake is a cloud-based data platform that allows businesses of all sizes to store, manage, and analyze their data with ease. The platform is designed to handle all types of data, from structured to semi-structured and unstructured data. One of the key features of Snowflake is its ability to separate computing and storage, allowing you to scale each independently. This means you can easily increase your computing resources to handle complex workloads without having to worry about increasing your storage capacity.

Snowpark: The Future of Data Transformations in Snowflake

Snowpark is a new feature of Snowflake that allows developers to write code in their favorite programming languages, including JAVA, to perform complex data transformations directly in Snowflake. With Snowpark, developers can write and execute complex data transformations, without having to move data out of Snowflake and into a separate processing engine.

Leveraging the Power of JAVA with Snowpark

For JAVA developers, Snowpark offers a powerful new way to interact with Snowflake. With Snowpark, developers can write code in their favorite programming language, including JAVA, to perform complex data transformations and analytics within Snowflake. This means you can leverage your existing skills and experience to build powerful data-driven applications in Snowflake.

Snowpark supports the full range of JAVA features, including classes, functions, and methods. You can use familiar tools like Maven and Gradle to build, test, and deploy your Snowpark applications. Snowpark also supports popular JAVA libraries and frameworks, such as Apache Spark and Apache Arrow, giving you access to a rich ecosystem of tools and resources to build powerful data-driven applications.

For more information: https://snowflake.com

Learn More About Snowflake and Snowpark at Teqnation

If you're interested in learning more about Snowflake and Snowpark, be sure to attend Teqnation on May 16. At this developers' conference, you'll have the opportunity to hear from Snowflake's industry experts about the latest trends and innovations. You can learn more about Snowflake's capabilities, including demos on how to use Snowpark to build powerful data-driven applications. We hope to see you there!

INHOUD

Java 20

10 Java SE 20 was released March 21, 2023, so it should be generally available when this article is published. Most of the JEPs in this release have something to do with Pattern Matching or Virtual Threads and are resubmissions of already known Incubator and Preview features.

Migraties met OpenRewrite

18 Spring Boot 3 is onlangs uitgebracht en het is tijd om ons project naar deze nieuwste versie te upgraden. Het migreren van ons project naar Spring Boot 3 vereist onder meer Java 17 en de Jakarta Namespace in plaats van de Javax Namespace, wat het proces tijdrovend maakt.

Boost your productivity

38 As engineers we love to solve problems. Even better if we can solve problems with code. And the best solutions are those that have high quality code. We love this to the point where we spend hours upon hours on tests and refactoring. One place where we saw little of this love was in our engineering tools. We had many aliases and scripts lying around which we used, but the code for these was in a pretty bad shape. We decided to roll up our sleeves and dust off the old bash routines.

(NEW!)

NLJUG Membership tiers

42 NLJUG has recently introduced three new membership tiers - Base, Core, and Key - giving members more options to choose from. But what does it mean for you? Go check out page 42 for all the ins-and-outs.

SCHRIJVEN VOOR JAVA MAGAZINE?

Ben je een enthousiast lid van NLJUG en zou je graag willen bijdragen aan Java Magazine? Of ben je werkzaam in de IT en zou je vanuit je functie graag je kennis willen delen met de NLJUG-community? Dat kan! Neem contact op met de redactie, leg uit op welk gebied je expertise ligt en over welk onderwerp je graag zou willen schrijven. Direct artikelen inleveren mag ook.
Mail naar info@nljug.org en wij nemen zo spoedig mogelijk contact met je op.

INTEGRATIETESTEN MET DE INTELLIJ HTTP-CLIENT

In IntelliJ versie 2017.3 Ultimate introduceerde Jetbrains een nieuwe HTTP-client die de oude grafische REST-client verving (https://www.jetbrains.com/help/idea/http-client-in-product-code-editor.html). Het grote voordeel hiervan is dat de huidige HTTP-client werkt op basis van tekstbestanden. Dit maakt het aantrekkelijker in gebruik.

Erwin de Gier is Software Architect bij Trifork Amsterdam. Daarnaast is hij regelmatig te zien als spreker op diverse softwareconferenties.

Het is nu bijvoorbeeld mogelijk om de HTTP-request-scripts in te checken in versiebeheer samen met je code. In het zesjarige bestaan is de HTTP-client steeds volwassener geworden en zijn er steeds meer features toegevoegd. Dit maakt het nu een waardige oplossing om je HTTP REST-api tests in te schrijven, als alternatief voor tools zoals bijvoorbeeld Postman.

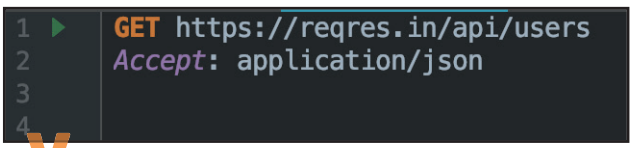
In Java Magazine nr. 4 van 2020 (deze kan je op nljug.org bij Java Magazine teruglezen) is reeds een introductie gegeven van de HTTP-client. Voor de voorbeelden maken we gebruik van een mock REST-service die gehost wordt op **https://reqres.in/** en gratis beschikbaar is.

{ VOORBEELD }

De HTTP-client werkt standaard met tekstbestanden met de extensie .http. In Listing 1 staat een voorbeeld van een eenvoudig `GET`-request. Deze request gebruiken we om een lijst met gebruikers op te halen.

L1
```
GET https://reqres.in/api/users
```

In de meest simpele vorm bestaat het commando uit de HTTP-request-methode (`GET`) en een URL (**https://reqres.in/api/users**). De HTTP-request kan worden uitgevoerd door op het 'play'-icoon te klikken, zie Afbeelding 1.



Afbeelding 1.

De request wordt uitgevoerd in het 'Services window'. In Listing 2 zie je de response.

L2
```
GET https://reqres.in/api/users

   HTTP/1.1 200 OK
   Date: Thu, 11 Jul 2019 11:12:43 GMT
   Content-Type: application/json; charset=utf-8
   ...

   {
     "page": 1,
     "per_page": 3,
     "total": 12,
     "total_pages": 4,
     "data": [
       {
         "id": 1,
         "email": "george.bluth@reqres.in",
         "first_name": "George",
         "last_name": "Bluth",
         "avatar": "https://s3.amazonaws.com/
uifaces/faces/twitter/calebogden/128.jpg"
       },
        ...
     ]
   }

   Response code: 200 (OK); Time: 98ms; Content
length: 539 bytes
```

Listing 3 toont een uitgebreider voorbeeld. Naast de HTTP-methode (`POST`) en de URL hebben we nu een header toegevoegd.

Daaronder staat onze post body in JSON. Op deze manier kun je meerdere headers en een body toevoegen. Dit maakt het ook mogelijk om te communiceren met services die authenticatie vereisen. Zo zou je bijvoorbeeld een basic authenticationheader kunnen toevoegen of een JWT-token.

L3
```
POST https://reqres.in/api/users
Content-Type: application/json
Authorization: Basic user password
Cookie: country=Netherlands; language=Dutch

{
    “name”: “Erwin”,
    “job”: “Developer”
}
```

In Listing 3 kun je ook zien dat je cookies kunt meegeven met een request. Cookies die door de server worden teruggegeven, worden automatisch meegegeven met de volgende request. Door middel van multipart-requests kun je files uploaden naar de server, Listing 4 toont hier een voorbeeld van.

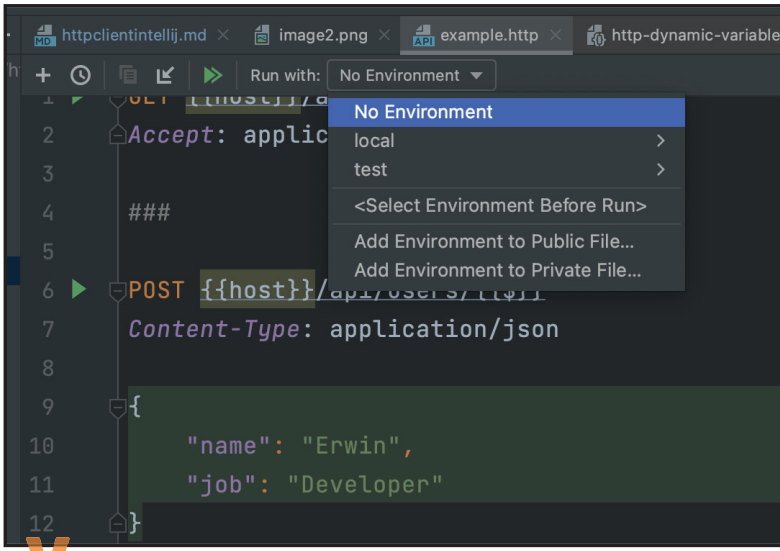
L4
```
### Upload an image to the server
POST {{url}}/media?name=Name for
{{$uuid}}&fileName=filename.png
Content-Type: multipart/form-data;
boundary=WebAppBoundary
Authorization: Bearer {{token}}

--WebAppBoundary
Content-Disposition: form-data; name=”file”;
filename=”test.png”

< test.png
--WebAppBoundary--
```

{ ENVIRONMENT VARIABELEN }

Vaak wil je requests maken naar verschillende omgevingen (lokaal, test, etc.). Het is mogelijk om gebruik te maken van variabelen in je script. Voeg hiervoor een JSON-bestand toe met de naam ‘http-client.env.json’. Een voorbeeld zie je in Listing 5. De environment configuratie bestaat uit JSON, waarbij je een lijst maakt van alle omgevingen. Elke omgeving kan weer meerdere key-value pairs bevatten.

Bij het uitvoeren van de request kun je nu kiezen uit verschillende omgevingen (zie Afbeelding 2). Dit maakt de scripts goed herbruikbaar.

Secrets kunnen geplaatst worden in een bestand met de naam ‘http-client.private.env.json’. De variabelen hierin zijn een aanvulling van die uit het bestand ‘http-client.env.json’. Uiteraard voeg je dit



Afbeelding 2.

bestand toe aan het .gitignore-bestand.

L5
```
example.http:
GET https://{{host}}/api/users
Accept: application/json

http-client.env.json:
{
  “local”: {
    “host”: “http://localhost:8079”
  },
  “test”: {
    “host”: “https://reqres.in”
  }
}
```

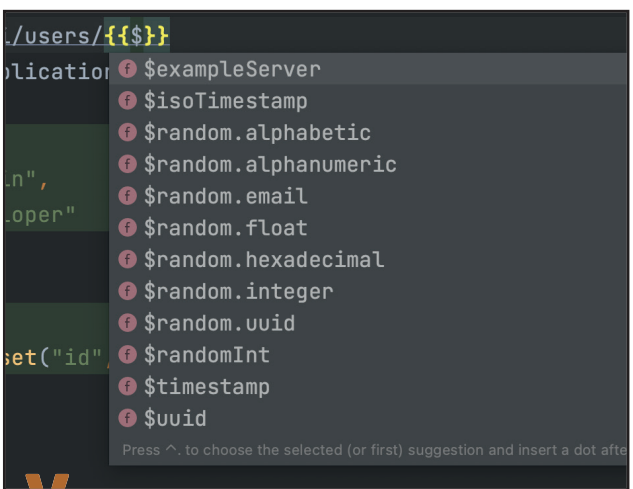
{ SCRIPTING }

Tot nu toe hebben we requests met headers en een body gespecificeerd en hebben we een configuratie opgezet voor meerdere omgevingen. In sommige gevallen wil je het resultaat van een request gebruiken voor een volgende request. In het voorbeeld in Listing 6 staan twee requests, met de `POST`-request maken we een gebruiker aan. In de response krijgen we een ID terug voor deze gebruiker. Onder de request-specificatie staat een stuk JavaScript tussen `<{%` en `%}`. In dit script hebben we toegang tot het client-object, waarop we globale variabelen kunnen setten. Ook hebben we toegang tot de response met de headers en de body. Hierdoor kunnen we een nieuwe globale variabele aanmaken met de naam ‘ID’ en de waarde van de ID uit de responsebody hieraan toewijzen. Deze variabele is vervolgens beschikbaar in de volgende requests en kan gebruikt worden als `{{id}}`. In de `PUT`-request gebruiken we dit ID om de gebruiker te wijzigen.

L6

```
POST https://{{host}}/api/users
Content-Type: application/json
{
    “name”: “Erwin”,
    “job”: “Developer”
}

> {%
  client.global.set(“id”, response.body.id);
%}

###
PUT https://{{host}}/api/users/{{id}}
Content-Type: application/json

{
    “name”: “Erwin”,
    “job”: “Architect”
}

> {%
  client.test(“Update should change the job name”,
function() {
      client.log(“Received response with
job”+response.body.job);
      client.assert(response.status === 200,
“Response status is not 200”);
      client.assert(response.body.job ===
‘Architect’, “Expected: Architect” + “\nreceived:
“ + response.body.job);
   });
%}
```

Het is ook mogelijk om tests te schrijven en assertions te doen in het scriptblok. We maken een test aan met de functie `client.test`. Hierin kun je de functie `client.assert` gebruiken om assertions uit te voeren. In het voorbeeld controleren we de statuscode en of de response de nieuwe jobnaam ‘Architect’ bevat. Het is handig om hier het verwachte en het daadwerkelijke resultaat te printen. Het resultaat verschijnt in de ‘Run window’ in het tabblad ‘Tests’. Tot slot kan de client nog gebruikt worden om te loggen.



Afbeelding 3.

Naast scripts die na een request worden uitgevoerd, is het sinds kort ook mogelijk om scripts uit te voeren voor de request. Deze scripts worden direct voor de request-definitie geplaatst. Je hebt toegang tot het request-object, waardoor je bij de request-variabelen, headers en environment kan. In de environment staan de environment variabelen. Listing 7 geeft hier van een voorbeeld.

{ CREATE ITEM }

L7

```
< {%
    request.variables.set(“id”, “identifier”);
%}
POST {{url}}/item
...
```

Naast het gebruik van variabelen in de scripts en in URL’s en request-bodies kun je ook gebruikmaken van een aantal interne functies, bijvoorbeeld `{{$uuid}}`, `{{$randomInt}}` en `{{$timestamp}}`, respectievelijk voor het generen van een random UUID, een random integer of de huidige timestamp. Er zijn echter nog veel meer mogelijkheden, bijvoorbeeld voor het genereren van een random e-mailadres. IntelliJ heeft context completion voor HTTP-files. Door te beginnen met `{{$` kan de hele lijst van mogelijkheden worden getoond (zie Afbeelding 3).

Het is ook mogelijk om scripts van een extern bestand in te lezen met: `> scripts/script.js`.

{ CI/CD INTEGRATIE }

Op ons project schrijven we voor elke feature HTTP-tests. De .http-bestanden bewaren we naast de code, zodat je tijdens het bouwen van de API de tests kunt uitvoeren. Ook kun je ze gebruiken om aan anderen uit te leggen hoe de API in elkaar zit en kunnen ze als voorbeeld dienen om te bepalen hoe de request-objecten moeten worden opgebouwd. Ze fungeren dus ook als documentatie. Na het committen worden de tests uitgevoerd op de feature branch (in Docker: **https://hub.docker.com/r/jetbrains/intellij-http-client**). Na de merge naar de main branch volgt een deploy op de testomgeving en worden de tests uitgevoerd tegen deze nieuwe versie. Het uitvoeren op onze CI/CD-omgeving doen we met behulp van de HTTP-Client Cli-runner **https://jb.gg/ijhttp/latest**. Deze CLI-runner is beschikbaar als een losse JAR en kan dus ook gebruikt worden buiten IntelliJ en door niet-ultimate gebruikers. Om de tests uit te voeren gebruik je het volgende commando:

Afbeelding 4.

```
./ijhttp example.http --env-file http-client.env.
json --env test
```

Hierin is `test` de aanduiding van de omgeving, zoals geconfigureerd in http-client.env.json. `example.http` is het bestand waarin onze HTTP-requests en tests staan.

Afbeelding 4 toont een deel van het resultaat van deze call. Per test wordt de request en response getoond. Ook wordt er een overzicht van de testresultaten gegeven. Het is mogelijk om het testresultaat in JUnit-formaat te genereren. Doordat we exact dezelfde tests in onze pipeline draaien als we beschikbaar hebben in IntelliJ, is het heel makkelijk om falende tests te debuggen. Je kunt immers in IntelliJ ook alle tests in een file uitvoeren, tegen een lokaal draaiende instantie van je applicatie in debug-modus. Hierdoor kun je snel achterhalen waar de fout zit en deze herstellen.

{ INTEROPERABILITY }

Een groot nadeel is dat de HTTP-client een feature is van de betaalde (Ultimate) versie van IntelliJ. Om er goed van gebruik te kunnen maken, zal iedereen dus een betaalde IntelliJ-versie moeten hebben. Gelukkig is het ook mogelijk om de HTTP-bestanden buiten IntelliJ te bewerken en uit te voeren. Naast de al genoemde HTTP-Client Cli-runner, is er ook een Visual Studio Code plugin (**https://marketplace.visualstudio.com/items?itemName=humao.rest-client**). Helaas kan de http-client.env.json niet gebruikt worden in Visual Studio Code en is het nodig om de omgevingsvariabelen aan de generieke settings.json toe te voegen (zie Listing 8).

L8
```
    “rest-client.environmentVariables”: {
    “local”: {
        “host”: “http://localhost:8079”
    },
    “test”: {
        “host”: “https://reqres.in”
    }
}
```

Hierna kunnen de requests worden uitgevoerd vanuit VSC). Als deze opties niet genoeg zijn, is het ook mogelijk om de HTTP-requests vanuit IntelliJ te exporteren als `curl`-commando's en vice versa.

{ CONCLUSIE }

De integratietests naast je code opslaan, zorgt ervoor dat er een lagere drempel is om deze uit te voeren en te onderhouden. Hierdoor wordt het ook makkelijker om de integratietests te schrijven als onderdeel van het bouwen van een feature.

De HTTP-tests kunnen eenvoudig worden uitgevoerd tegen een lokale instantie van de applicatie of een testomgeving. Ook kunnen ze dienen als voorbeeld voor partijen die je REST API consumeren.

Met de ijhttp library is het mogelijk om dezelfde tests uit de voeren in je CI/CD-pipeline. De bestanden zijn eenvoudig van opzet en kunnen makkelijk opgeslagen worden in versiebeheer. Voor ons zijn ze een goed alternatief voor tools als bijvoorbeeld Postman, daarom gebruiken we ze voor alle API-tests in ons project. <

JAVA 20

Java SE 20 was released March 21, 2023, so it should be generally available when this article is published. Most of the JEPs in this release have something to do with Pattern Matching or Virtual Threads and are resubmissions of already known Incubator and Preview features. This version doesn't contain any new main features. The most exciting innovation in this release is called 'Scoped Values' and is intended to widely replace thread-local variables.

In this article we will take a look at all the new and resubmitted Incubator and Preview features.

All code examples were tried out in a Docker container with OpenJDK 20 (Listing 1) [2].

L1
```
$ docker run -it --rm \
      -v $(pwd)/src:/src \
      openjdk:20-slim /bin/bash
$ cd /src/java20
```

{ PREVIEW AND INCUBATOR FEATURES }

429: Scoped Values (Incubator)

Just like Virtual Threads (see below), Scoped Values were developed as part of Project Loom [4].

Project Loom is intended to explore, incubate and deliver Java VM features, and APIs built on top of them for the purpose of supporting easy-to-use, high-throughput lightweight concurrency, and new programming models on the Java platform.

The Scoped Values feature in Java provides a way to define a value within a particular scope and ensures that it is used only within that scope. This feature makes it easier to manage data and reduces the risk of errors by limiting the scope of that data to only the areas where it is needed.

L2
```
package java20;
import jdk.incubator.concurrent.*;
public class JEP429 {
    private static final ScopedValue<String>
USERNAME = ScopedValue.newInstance();
    public static void main(String[] args) {
        ScopedValue.where(USERNAME, "Duke", () ->
System.out.println(USERNAME.get()));
        ScopedValue.where(USERNAME, "Java", () ->
System.out.println(USERNAME.get()));
    }
}
$ java --add-modules jdk.incubator.concurrent
--enable-preview --source 20 JEP429.java
WARNING: Using incubator modules: jdk.incubator.
concurrent
Duke
Java
```

Ivo Woltring is a Principal Expert and Codesmith at Ordina and likes to keep up with new developments in the software world.

432: Record Patterns (Second Preview)

This JEP aims to improve the expressiveness and readability of code that deals with records. A record pattern can be used with `instanceof` or `switch` to access the fields of a record without casting and calling accessor methods.

Type pattern matching was introduced in Java through JEP 394 in Java SE 17. The switch case statement was enhanced to work with pattern matching in JEP 406 and 420 in Java SE 18 [2,3].

L3
```
public class JEP432 {
    record Pair(Object x, Object y) { }
    record Point(int x, int y) {}
    enum Color { RED, GREEN, BLUE }
    record ColoredPoint(Point p, Color c) {}
    record Rectangle(ColoredPoint upperLeft,
ColoredPoint lowerRight) {}
    public static void noMatchExample() {
        Pair p = new Pair(42, 42);
        System.out.println("p instanceof
Pair(String s, String t) -> "
                + (p instanceof Pair(String s,
String t)));
    }
    static void printUpperLeftColors(Rectangle[]
```

```
r) {
        for (Rectangle(ColoredPoint(Point p, Color
c), ColoredPoint lr): r) {
            System.out.println(c);
        }
    }
    static void dump(Point[] pointArray) {
        // matches all Point instances
        for (Point(var x, var y) : pointArray) {
            System.out.println("(" + x + ", " + y
+ ")");
        }
    }
    public static void main(String[] args) {
        noMatchExample();
        System.out.println("---");
        printUpperLeftColors(new Rectangle[] {
        new Rectangle(new ColoredPoint(new
Point(1, 2), Color.RED),
        new ColoredPoint(new Point(3, 4), Color.
BLUE)),
        new Rectangle(new ColoredPoint(new
Point(5, 6), Color.GREEN),
        new ColoredPoint(new Point(7, 8), Color.
BLUE))
        });
        System.out.println("---");
        dump(new Point[] { new Point(1, 2), new
Point(3, 4) });
    }
}
$ java --enable-preview --source 20 JEP432.java
Note: JEP432.java uses preview features of Java SE
20.
Note: Recompile with -Xlint:preview for details.
p instanceof Pair(String s, String t) -> false
---
RED
GREEN
---
(1, 2)
(3, 4)
```

433: Pattern Matching for switch (Fourth Preview)
JEP 433 proposes a new feature that allows developers to use pattern matching in their switch statements. Essentially, this means that instead of just comparing a value to a series of constant values, developers can use more complex patterns to match against the value, including things like data types and structures like arrays or objects. This can make code more concise and easier to read (Listing 4), as developers can write more expressive code that directly matches against the data they are working with.

L4

```
return switch (o) {
    case null      -> "Oops";
    case Integer i -> String.format("int %d", i);
    case Long l    -> String.format("long %d", l);
    case Double d  -> String.format("double %f",
d);
    case String s  -> String.format("String %s",
s);
    case Point(int x, int y) p  -> String.
format("Point %s", s);
    default        -> o.toString();
};
```

See reference [2] for more code samples.

434: Foreign Function & Memory API (Second Preview)
This JEP proposes the addition of a new feature that allows developers to interface with native code and memory more efficiently. This means that Java applications can now use functions and data from other programming languages, such as C or C++, without the need for the complex and error-prone Java Native Interface (JNI).

The proposed API allows Java applications to directly access native code libraries and manage memory in a more efficient and controlled way (such as those provided by operating systems or third-party software vendors), without having to worry about compatibility issues or performance penalties.

In Listing 5 you can see how the C library `"strlen"` function is called to retrieve the length of a string. It is a 'nonsensical' example, but it does illustrate how it works. Most Java developers will probably rarely come into contact with the Foreign Function & Memory API.

L5

```
import java.lang.foreign.*;
import java.lang.invoke.MethodHandle;
public class JEP434 {
  public static void main(String[] args) throws
Throwable {
    // 1. Get a lookup object for commonly used
libraries
    SymbolLookup stdlib = Linker.nativeLinker().
defaultLookup();
    // 2. Get a handle to the "strlen" function in
the C standard library
    MethodHandle strlen = Linker.nativeLinker().
downcallHandle(
            stdlib.find("strlen").orElseThrow(),
            FunctionDescriptor.of(ValueLayout.
JAVA_LONG, ValueLayout.ADDRESS));
    // 3. Convert Java String to C string and
```

```
store it in off-heap memory
    try (Arena offHeap = Arena.openConfined()) {
        MemorySegment str = offHeap.
allocateUtf8String("Java Magazine Rockz!");
        // 4. Invoke the foreign function
        long len = (long) strlen.invoke(str);
        System.out.println("len = " + len);
    }
    // 5. Off-heap memory is deallocated at end of
try-with-resources
  }
}
$ javac --enable-preview --source 20 JEP434.java
Note: JEP434.java uses preview features of Java SE
20.
Note: Recompile with -Xlint:preview for details.
$ java --enable-preview --enable-native-
access=ALL-UNNAMED JEP434
len = 20
```

436: Virtual Threads (Second Preview)
Virtual Threads is an incubator feature introduced in Java SE 19 that allows for more efficient execution of concurrent code.

In simple terms, it allows multiple threads of code to run simultaneously without using up unnecessary resources. This can improve the performance and responsiveness of Java applications, especially those that require frequent and complex interactions between threads.

Virtual threads are lightweight threads that can be created and managed more easily than traditional threads, and they are designed to be more efficient in terms of memory and CPU usage. This makes it possible to scale applications more easily, and to handle more concurrent requests without sacrificing performance or stability.

Virtual Threads are not maped 1:1 on an os thread, instead they are created and managed by the Java runtime. In the last Java Magazine a complete article was dedicated to this topic (see Java Magazine 2023-01).

437: Structured Concurrency (Second Incubator)
This incubator feature enables better management of concurrent tasks in Java programs. With structured concurrency, tasks are organized into 'scopes' to ensure that all tasks within a scope complete before the scope itself is considered complete.

This makes it easier to manage and control concurrent tasks, reducing the risk of problems such as race conditions and deadlocks. It also makes it easier to cancel or interrupt tasks within a scope without affecting other tasks, improving the overall stability and reliability of the program.

In simple terms, structured concurrency helps developers write more reliable and efficient code when dealing with multiple tasks that run concurrently.

In my article about Java 19 [2,3] a code example is provided.

438: Vector API (Fifth Incubator)
This JEP is a proposed enhancement that aims to provide a new set of vector operations that can better utilise modern hardware platforms, such as SIMD (Single Instruction Multiple Data) and AVX (Advanced Vector Extensions) instruction sets.

The Vector API is designed to enable accelerated computations on supported hardware without requiring any specific platform knowledge or code specialization. The API aims to expose low-level vector operations in a simple and easy-to-use programming model, allowing performance optimizations to be integrated seamlessly into existing Java code.

The proposed API includes several key features, including support for variable-length vectors, a new set of mathematical operations, and a range of predicate and masking functions for data selection and manipulation. The API also includes support for hardware-specific features such as vector masks, and cache control operations.

Overall this JEP is aimed at improving the performance of Java applications on modern hardware, as well as providing a more convenient and efficient way to utilize advanced vector processing capabilities.

The first incubator was introduced in Java SE 16 and a code sample can be found at reference [2].

{ CONCLUSION }

All the features mentioned in this article have potential and I am looking forward to using them. Let's hope that all of them will become official features in the next version (21), which will be a Long Term Support (LTS) version. <

{ REFERENCES }

1 https://openjdk.org/projects/jdk/20/
2 https://github.com/IvoNet/java-features-by-version
3 https://github.com/IvoNet/javamagazine
4 https://openjdk.org/projects/loom/
5 https://jdk.java.net/panama/

Stichting Devoxx4Kids Nederland ontvangt sponsorbedrag van adesso Nederland

Het tekort aan goed geschoold ICT personeel zal de komende jaren alleen maar groter worden. Om die trend te kunnen keren, is het noodzakelijk om kinderen al op jonge leeftijd kennis te laten maken met ICT, zodat meer kinderen na hun middelbare school de keuze kunnen maken voor een ICT gerichte opleiding.

De Stichting Devoxx4Kids Nederland heeft als doel het creëren van bewustwording van ICT bij kinderen. Dit doen ze door bedrijven te ondersteunen bij het organiseren van Devoxx4Kids dagen en door met het LITTIL platform professionals uit het vak in contact te brengen met scholen om daar ICT lessen te verzorgen. Dit alles wordt volledig gedaan door vrijwilligers. Eddy Vos van Devoxx4Kids is blij met de grote groep vrijwilligers die bij de stichting betrokken is. Naast vrijwilligers is de stichting ook afhankelijk van sponsoren die financiële middelen ter beschikking stellen.

adesso Nederland sponsort het werk van de Devoxx4Kids met een schenking van €2000. “Wij vinden het belangrijk dat kinderen op jonge leeftijd al de mogelijkheid krijgen om kennis te maken met ICT. Juist daarom zijn de Devoxx4Kids dagen en gastlessen

op scholen zo belangrijk”, aldus Jan Heuker, directeur van adesso Nederland.

“We zijn heel blij met de steun van adesso Nederland. Hopelijk zien meer bedrijven het belang van het werk van de stichting in, want financiële steun bij ons werk is hard nodig”, aldus Eddy Vos.

Voor meer informatie over de Stichting Devoxx4Kids Nederland en de mogelijkheid tot sponsoring, kan contact worden opgenomen met Eddy Vos (**eddy@devoxx4kids.nl**)

CO-CREATION BETWEEN UX AND SOFTWARE

What if we bring design, frontend and backend all together? How can we improve our flows by putting the users first?

To create an outstanding product with high value, everyone in the delivery process must understand the 'why'. This way, we all work towards the same purpose.

In this article we, Thamar (designer) and Simone (developer) will take you along in our journey. From our perspective, the two worlds of UX and software naturally overlap. This article will provide some approaches and tools which can help with the implementation of a process that supports close collaboration.

{ TEAM SETUP }

Every discipline comes with its own toolkit, knowledge and speciality in a specific field. By having the right disciplines at the table, we empower the setup of a successful team.

"From this purpose the team members will be chosen based on their personality, skillset and the type of project. If this all fits and the team purpose resonates with the new team member, we found a perfect match!"

There are many ways to describe team structures. In Table 1, you'll find a few options which best facilitate collaboration between multiple disciplines.

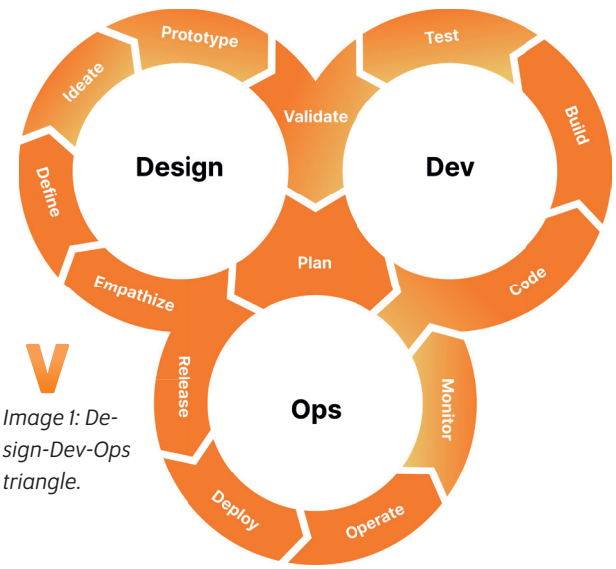
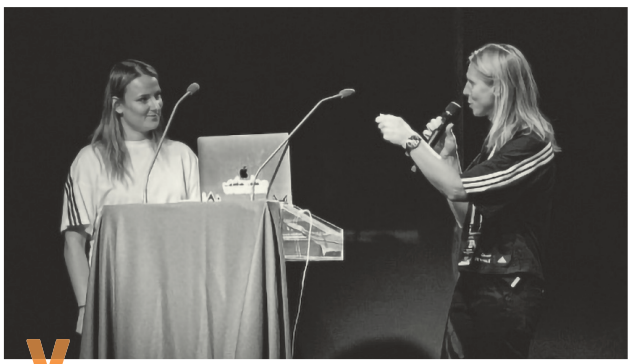


Image 1: Design-Dev-Ops triangle.



Thamar Swart (R) is currently the Global Director User Experience Design within adidas. With more than 15 years of experience in diverse team setups, she has seen how a project can be more innovative and successful when they start co-creating.
Simone de Gijt (L) is a Java and Kotlin Software Developer for OpenValue. In the 6 years that she works within IT she has seen the benefits in efficiency and market fit when trying to understand your user and thinking along in the design process.

{ THE PROCESS: DESIGN-DEV-OPS TRIANGLE }

Both design and software have their own process models. Design Thinking is commonly known as the primary model within design, whereas we in software are well-aware of the DevOps loop. Although these models are very helpful for the people within their respective discipline, they do not empower us to co-create. This is why we came up with the Design-Dev-Ops triangle. Let's take a look (Image 1).

In the triangle, you can find all the process steps of both the individual models. You read the model in the following way:

Every cycle starts with the **Plan** phase where all disciplines join the table.

- The very first cycle to enter will be the design cycle. So after the **Plan** phase, you start the cycle from **Empathize**.
- After the design cycle, the **Code** phase within the development cycle will be picked up.
- Last is the operation cycle, entered via the **Release** phase. By **Monitoring** the implementation we get insights that can lead to a new design cycle.

Team structure	Time to live	Purpose	UX/Dev division
Operations team	Permanent	Improve and maintain IT ecosystems, while empowering other teams by offering centralized solutions.	Only developers.
Process team	Permanent	Improve and maintain a complete, specific, business process from beginning to end.	Mixed team with designers and developers.
Feature team	Permanent	Improve and maintain at least one specific piece of functionality.	Only developers that have a 'single point of contact' with one designer. This designer is part of a design team.
Project team	Temporary	Quickly build something which can be released to production.	Depends on the product they are delivering. The team should always stay in contact with a dedicated designer of the team they are delivering for.

Table 1.

The phases in the cycle that have dual colors are the phases where the co-creation is most important.

{ DESIGN CYCLE }

The team starts together with the Plan phase. This is extremely important for the interdisciplinary connection of all the team members and to get a shared feeling and understanding of the 'why'. Not doing this in co-creation will most likely lead to:

› An isolated way of working between designers and developers. The 'what' being the most important point of reference during the development cycle, instead of the 'why'.

Visualization is key in generating an efficient and good collaboration. You'll find some tools to support co-creation in Table 2.

{ DEVELOPMENT CYCLE }

Again, the cycle starts with the **Plan** phase. Design is also included, as they are essential when you want to discuss the different iterations of implementing the end goal.

During this phase, the developers will also work out the tickets they need to deliver in more detail: the technical design. To get the most out of these designs, it is recommended to share your thoughts and designs with your design colleagues. However, this can be quite challenging due to the knowledge gap and the different vocabulary used. Some tools that can help the different disciplines understand each other are summarized in Table 3.

After the **Plan** phase, the developers will start writing their code. During the **Test** phase, the engineer who will perform the tests must have a direct communication channel with the designers. The tests should be created based on the designs that have been agreed upon during the design cycle.

Last but not least, it is recommended to let the designers play around with the tested and approved implementation before moving to production. They can do so in the recurring **Validate** phase. However, be careful of scope creep here. New wishes and requirements should enter the design cycle again.

{ OPERATIONS CYCLE }

Our experience is that the **Operations** cycle is often overlooked and undervalued. In order to give it the attention it deserves, we recommend having another **Plan** phase before moving on. One approach to consider is releasing the implementation as a beta version. This way, you release the feature to a small target group before you make it available to everyone.

Phase	Tool	Description
Empathize & Define	Journey map	Visualization of the user problems and/or opportunities.
Empathize & Define	Opportunity map	Generates a matrix of the effort compared to the value. Collaboration is a must. The designers will give input on the value and both disciplines look into the expected effort.
Ideate	Event storming	A tool you can use to get a better understanding of a user-flow or opportunity.
Prototype	Wireframes (UX) / Proof of concept (Software)	A prototype can be built with wireframes and other tools to the disposal of designers. However, the developers might be able to easily build a proof of concept in a specific branch, via mocks or behind feature flags.
Validate	Blueprint	Visualization of all the phases where user flows are combined with technical needs. Leading to all the tickets needed to be done to implement the solution.

Table 2.

Phase	Tool	Description
Plan	Process flow	Visualization of the user-flow, including the decision points, output, and integrations.
Plan	Sequence diagram	Shows the interaction between services when a request is being sent, including non-happy flows. Leading to more insight on what is happening in the backend and potential gaps, which might lead to different (functional) design choices and additional designs that need to be made.
Plan	Entity relationship	Helps to define which data is used and needed. Designers will be able to challenge these choices, preventing unnecessarily data from being saved.

Table 3.

Another approach can be to introduce A/B testing. Although it might sound like something you do in the development cycle, these tests are done in production with real customers. Some customers might get to see your new solution, while others are seeing another or the old implementation. This will give you some insight later on during the **Monitor** phase, which can be the input of a new design cycle.

After all the plans have been made, it is time to **Release** and Deploy to production. The developers are on fire and hopefully, if all goes well, they will keep an eye out to operate the existing code.

Metric	Description
User engagement	Shows how users are interacting with your content. Such as: time spent on site, pages per session, bounce rate, and click-through rate.
Heat maps	A visual representation of user activity on your website or application. Showing which areas are getting the most attention.
Conversion rate	The percentage of users who complete a desired action, such as making a purchase or filling out a form.
Customer satisfaction	Surveys or feedback mechanisms to see how satisfied users are with your product or service.
Traffic source	The channels through which users are coming to your website or application. Such as: via a search engine, social media, or referral traffic.
Customer lifetime value	The estimated value of a customer over the course of their relationship with your business.
Revenue metrics	Total revenue, revenue per user, and other financial metrics related to your business.

Table 4.

How they do that is by having monitoring in place.

The **Monitor** phase: think up front about which alerts, dashboards and logs should be present at which times? The requirements should be your first lead:

- Can you verify and monitor if the requirements are being fulfilled in production?
- Which information do you want to gain to provide input for future improvements?

As you can see, these starting points were initially designer driven. Now with the help of specific metrics we can really help each other to get more insight on how our feature is performing in production, but also get some leads on how to improve from here. In Table 4 you'll find some metrics to help you implement meaningful monitoring.

Spending dedicated time on quality monitoring will improve your product! Create a ticket to create some quality monitoring and consult with your designers. A dashboard hardly involves only design or development. It should be a co-creation!

{ KEY TAKEAWAYS }

Long story short: do you want to start today? First look at your team and the team structure. Does it fit with the 'why', does it fit with the team's purpose? Talented individuals != successful team. If you have the right team, incorporate design into your DevOps cycle. Be aware that the responsibility of the end product lies with the team. Responsibility for specific phases belongs to a discipline. In order to keep the responsibility where it belongs, you need to have trust in your team members. If not yet, start working on gaining that trust! When you start co-creating, find a language or a tool that works for all the disciplines. This won't be a one-size-fits-all. It will be agile; try something out, reflect on it and fine-tune it. And last but not least, give proper attention to the operations cycle. Your customer will thank you for it!

Image 2: Monitoring.

MIGRATIES MET OPENREWRITE

Spring Boot 3 is onlangs uitgebracht en het is tijd om ons project naar deze nieuwste versie te upgraden. Het migreren van ons project naar Spring Boot 3 vereist onder meer Java 17 en de Jakarta Namespace in plaats van de Javax Namespace, wat het proces tijdrovend maakt. Het goede nieuws is dat dit soort repetitieve taken geautomatiseerd kunnen worden, en daarvoor is OpenRewrite een handige tool.

Willem Cheizoo is een enthousiaste Java Developer bij JDriven. Met een innovatieve en teamgerichte aanpak werkt hij continu aan het leveren van kwalitatieve software.

Bij veel projecten gebeurt het updaten en aanpassen van nieuwe dependency's en library's nog altijd met de hand. Java-updates worden met regelmaat uitgesteld of updates naar nieuwere Spring Boot-versies worden op de lange baan geschoven. OpenRewrite is een tool die het mogelijk maakt om op grote schaal broncode-wijzigingen gecontroleerd door te voeren.

OpenRewrite bestaat in de basis uit recipes. Een recipe is gefocust op het aanpassen van één stuk code voor één specifiek probleem. Dit kan zijn:

- Een specifieke property in application.yml aanpassen.
- Een dependency upgraden in de pom.xml.
- Het wijzigen van een import, omdat de betreffende Class verplaatst is naar een andere package.

Een groot aantal recipes zijn op de website van Moderne [1] terug te vinden. Een voorbeeld van zo'n recipe via de link in de referenties [2].

Het directe resultaat van het configureren en uitvoeren van een recipe is gewijzigde code. Indien de code van het project al volledig up-to-date is, resulteert het dus in géén enkele wijziging. De recipes zijn namelijk idempotent, wat betekent dat we ze meerdere keren kunnen uitvoeren met hetzelfde eindresultaat.

Om de kracht van OpenRewrite te laten zien, zullen we een oudere versie van de Spring Petclinic Sample application die nog op Spring Boot 1.5 draait geautomatiseerd in drie stappen upgraden naar Spring Boot 3.0 met Java 17, JUnit 5 en de nieuwe Jakarta Namespaces.

{ PROJECT UITCHECKEN }

Allereerst zullen we een check-out doen van een verouderde branch van het project, zie Listing 1. Deze uitgecheckte versie draait nog op Spring Boot 1.5 en Java 8, zoals terug te zien is in de pom.xml. We maken ook direct een nieuwe 3.0.x branch, zodat we de wijzigingen om naar Spring Boot 3 te komen in de nieuwe branch kunnen committen.

L1
```
git clone https://github.com/spring-projects/
spring-petclinic.git
cd spring-petclinic
git switch -c 3.0.x 1.5.x
```

{ STAP 1: MIGRATIE NAAR SPRING BOOT 2 }

Het project migreren van Spring Boot 1 naar Spring Boot 2 is het eerste dat we zullen doen. Direct naar Spring Boot 3 migreren kan ook, maar voor dit artikel doen we het in stapjes. Hiervoor configureren we de OpenRewrite-plugin in ons project. Dit is tijdelijk en zullen we aan het einde weer ongedaan maken.

We roepen de `init`-goal van de `rewrite-maven-plugin` aan (Listing 2), zodat in onze pom.xml de correcte configuratie van de plugin geplaatst wordt. Dit kan alléén wanneer OpenRewrite nog niet in onze pom.xml geconfigureerd is. Wanneer de plugin al wel geconfigureerd zou zijn, hadden we veel eenvoudiger `rewrite:configure` als goal kunnen gebruiken.

In dit artikel zien we een aantal versienummers staan (zoals in Listing 2) die tijdens het schrijven van het artikel de meest recente versies waren.

L2
```
./mvnw org.openrewrite.maven:rewrite-maven-
plugin:4.42.0:init \
 -Ddependencies=org.openrewrite.recipe:rewrite-
spring:LATEST \
 -DactiveRecipes=org.openrewrite.java.spring.
boot2.SpringBoot1To2Migration
```

OpenRewrite is nu als plugin geconfigureerd, wat terug te vinden is in de pom.xml van ons project. We zijn nu feitelijk klaar om het rewrite-proces te starten, door gebruik te maken van de `run`-goal van de OpenRewrite-plugin. In Listing 3 zien we hoe we dit proces kunnen starten. Effectief zal dit code wijzigen op basis van de geconfigureerde `activeRecipes`.

L3
```
./mvnw rewrite:run
```

De plugin zal in de logs laten zien welke recipe geleid heeft tot welke wijziging. Een kleine snippet hiervan zien we in Listing 4. Dat betekent dat we exact kunnen achterhalen waar bepaalde wijzigingen vandaan komen.

{ TESTS REPAREREN }

Om te controleren of onze migratie geslaagd is, kunnen we een volledige build draaien, zie Listing 5.

L5
```
./mvnw verify
```

We zien in de logs van Listing 6 dat een tweetal tests gefaald zijn. De ene failure wordt veroorzaakt door een wijziging in `javax.validation`. De andere door een gewijzigde default media type in Spring Boot.

L6
```
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   ValidatorTests.
shouldNotValidateWhenFirstNameEmpty:42
expected: "may not be empty"
but was : "must not be empty"
[ERROR]   VetControllerTests.
testShowResourcesVetList:67 Content type
expected: <application/json;charset=UTF-8>
but was: <application/json>
[INFO]
[ERROR] Tests run: 41, Failures: 2, Errors: 0,
Skipped: 1
```

Met een eenvoudige aanpassing in de tests kunnen we dit probleem oplossen, zoals te zien is in Listing 7 en 8. Deze aanpassing moet echter wel handmatig worden gedaan.

L7
```
src/test/java/org/springframework/samples/
petclinic/model/ValidatorTests.java
- assertThat(violation.getMessage()).
isEqualTo("may not be empty");
+ assertThat(violation.getMessage()).
isEqualTo("must not be empty");
```

L8
```
/src/test/java/org/springframework/samples/
petclinic/vet/VetControllerTests.java
- actions.andExpect(content().
contentType("application/json;charset=UTF-8"))
+  actions.andExpect(content().
contentTypeCompatibleWith("application/
json;charset=UTF-8"))
```

{ STAP 2: MIGRATIE NAAR JAVA 17 }

Ons einddoel is een gemigreerde applicatie op Spring Boot 3. Om naar Spring Boot 3 te kunnen migreren, dienen we eerst naar Java 17 te gaan.

Met het toepassen van een enkele recipe kunnen we ons project vrij eenvoudig migreren naar Java 17 (Listing 9), zelfs wanneer ons project nog op Java 8 zit. Dat komt omdat onderliggende recipes

L4
```
[WARNING] Changes have been made to pom.xml by:
[WARNING]     org.openrewrite.maven.ExcludeDependency
[WARNING]     org.openrewrite.maven.ExcludeDependency
[WARNING]     org.openrewrite.maven.AddDependency
[WARNING] Changes have been made to src/main/java/org/springframework/samples/petclinic/system/
CacheConfiguration.java by:
[WARNING]     org.openrewrite.java.spring.BeanMethodsNotPublic
```

ook worden meegenomen, waaronder `org.openrewrite.java.migrate.Java8toJava11`.

De kracht van OpenRewrite is dat een recipe kan bestaan uit een hiërarchie van meerdere recipes samen. De eventuele afhankelijkheden tussen recipes worden vastgelegd in de recipes zelf. We kunnen natuurlijk ook in één keer de recipe `org.openrewrite.java.spring.boot3.SpringBoot2To3Migration` gebruiken. Dit bevat de migratie naar Spring Boot 2, Java 17 maar ook naar Spring Boot 3. In dit artikel is er bewust gekozen om het stap voor stap uit te leggen.

L9
```
./mvnw rewrite:configure \
  -Ddependencies=org.openrewrite.recipe:rewrite-migrate-java:LATEST \
  -DactiveRecipes=org.openrewrite.java.migrate.UpgradeToJava17
```

Wanneer we nu nogmaals de `rewrite:run`-goal van de plugin uitvoeren (Listing 3), zal deze ons weer netjes rapporteren wat er gewijzigd is door welke recipe (Listing 10).

Met een kleine tussenstap naar Java 11, hebben we in twee stappen ons project geupgrade naar Java 17.

{ STAP 3: MIGRATIE NAAR SPRING BOOT 3 }

Met de migratie naar Java 17 zijn we klaar om te migreren naar Spring Boot 3. Hiervoor kunnen we het `recipe org.openrewrite.java.spring.boot3.SpringBoot2To3Migration` gebruiken (Listing 11). Hiervoor hebben we deze keer wel een andere dependency nodig, dit is de dependency waarin de bewuste recipe zit.

Net zoals in de eerdere stappen draaien we hierna de goal `rewrite:run` (Listing 3) en geeft de plugin in de logs ons exact aan wat er gewijzigd is (Listing 12).

L10
```
[WARNING] Changes have been made to pom.xml by:
[WARNING]     org.openrewrite.java.migrate.UpgradeToJava17
[WARNING]         org.openrewrite.java.migrate.Java8toJava11
[WARNING]             org.openrewrite.java.migrate.jacoco.UpgradeJaCoCoMavenPluginVersion
[WARNING]                 org.openrewrite.maven.UpgradePluginVersion: {groupId=org.jacoco,
artifactId=jacoco-maven-plugin, newVersion=0.8.8}
[WARNING]             org.openrewrite.java.migrate.JavaVersion11
[WARNING]                 org.openrewrite.maven.ChangePropertyValue: {key=java.version, newValue=11,
addIfMissing=false}
[WARNING]         org.openrewrite.java.migrate.JavaVersion17
[WARNING]             org.openrewrite.maven.ChangePropertyValue: {key=java.version, newValue=17,
addIfMissing=false}
```

L12
```
[WARNING] Changes have been made to pom.xml by:
[WARNING]     org.openrewrite.java.spring.boot3.UpgradeSpringBoot_3_0
[WARNING]         org.openrewrite.java.spring.boot3.MavenPomUpgrade
[WARNING]             org.openrewrite.maven.UpgradeParentVersion: {groupId=org.springframework.boot,
artifactId=spring-boot-starter-parent, newVersion=3.0.0-SNAPSHOT}
[WARNING]         org.openrewrite.java.migrate.jakarta.JavaxMigrationToJakarta
[WARNING]             org.openrewrite.java.migrate.jakarta.JavaxPersistenceToJakartaPersistence
[WARNING]                 org.openrewrite.maven.AddDependency: {groupId=jakarta.persistence,
artifactId=jakarta.persistence-api, version=3.x, onlyIfUsing=javax.persistence.*}
[WARNING]             org.openrewrite.java.migrate.jakarta.JavaxValidationMigrationToJakartaValidation
[WARNING]                 org.openrewrite.maven.RemoveDependency: {groupId=javax.validation,
artifactId=validation-api}
[WARNING]             org.openrewrite.java.migrate.jakarta.JavaxXmlBindMigrationToJakartaXmlBind
[WARNING]                 org.openrewrite.maven.AddDependency: {groupId=jakarta.xml.bind,
artifactId=jakarta.xml.bind-api, version=3.x, onlyIfUsing=javax.xml.bind..*}
[WARNING]                 org.openrewrite.maven.AddDependency: {groupId=org.glassfish.jaxb,
artifactId=jaxb-runtime, version=3.x, scope=runtime, onlyIfUsing=javax.xml.bind..*}
[WARNING]             org.openrewrite.java.migrate.jakarta.EhcacheJavaxToJakarta
[WARNING]                 org.openrewrite.maven.ChangeDependencyClassifier: {groupId=org.ehcache,
artifactId=ehcache, newClassifier=jakarta}
```

Afbeelding 1.

```
./mvnw rewrite:configure \
 -Ddependencies=org.openrewrite.recipe:rewrite-
spring:LATEST \
 -DactiveRecipes=org.openrewrite.java.spring.
boot3.SpringBoot2To3Migration
```

L11

{ STAP 4: PLUGIN OPSCHONEN }

In de pom.xml van het project hebben we de OpenRewrite-plugin geconfigureerd. We hebben de keuze om deze te laten staan of eventueel op te ruimen. Dit kan met één enkel commando (Listing 13).

```
./mvnw rewrite:remove
```

L13

{ CONCLUSIE }

OpenRewrite maakt het upgraden en migreren van Java-applicaties eenvoudig door middel van recipes. De hiërarchie van recipes (zie Afbeelding 1) wordt automatisch uitgevoerd, waardoor inhoudelijke kennis niet nodig is om te weten welke andere recipes eerst uitgevoerd moeten worden.

Het automatiseren van de migratie van Spring Boot 1.5 naar Spring Boot 3 met de recipes is gemakkelijk te scripten en kan op meerdere projecten worden toegepast. Ook vanuit een build-server is dit gemakkelijk te doen. Daarnaast is het mogelijk om zelf recipes te maken voor shared library's, gedeelde componenten of een eenduidige migratiestrategie.

OpenRewrite kan teams enorm veel tijd besparen bij het upgraden van projecten en verlaagt de drempel om te migreren naar de laatste functionaliteiten en dependency-versies. Daarnaast verkleint het de risico's op fouten.

Het stappenplan is ook terug te vinden in mijn blog [3].

{ LINKS }

1 https://public.moderne.io/
2 https://public.moderne.io/recipes/org.openrewrite.java.spring.boot3.UpgradeSpringBoot_3_0
3 https://blog.jdriven.com/2022/03/major-migrations-made-easy-with-openrewrite/

POWERED BY nl.jug

J·SPRING nl.jug 2023

Save the date for the NLJUG Java Spring Conference

June 21st, 2023 at The Media Plaza in Jaarbeurs Utrecht

GET YOUR TICKET AT: JSPRING.NL/TICKETS/


Nataliia Dziubenko


Ana Maria Mihalceanu


Ixchel Ruiz


Bouke Nijhuis


Roy van Rijn


Frank Delporte


Grace Jansen

More info such as the line-up will be announced on the official website: jspring.nl

MAIN SPONSOR

CHILIT

CO-SPONSORS

ABN·AMRO

Rabobank

PARTNERS

PICNIC

profit4cloud

Rijksoverheid

topicus


We think there is a place for everyone within the bank. Including employees with autism.
Discover IT & how Rabobank creates an inclusive culture at **rabobank.jobs**

Rabobank

DEVITJOBS.NL

Java Software Developer

Salarisindicatie:
60.000 - 85.000 EUR

Locatie:
Nieuwegein

Technologieën:
Spring Boot, Jenkins, Maven, Quarkus, Java

ORDINA

Java Developer

Salarisindicatie:
55.000 - 85.000 EUR

Locatie:
Amsterdam

Technologieën:
AWS, Backend, Docker, ElasticSearch, Git, Java, Kafka, Kubernetes, Maven, MongoDB, PostgreSQL, RabbitMQ, Spring, Terraform, Cloud

PIC NIC

Java Developer

Salarisindicatie:
45.000 - 75.000 EUR

Locatie:
Utrecht

Technologieën:
Java, Fullstack, Frontend

Java ontwikkelaar (HODOR)

Salarisindicatie:
80.000 - 90.000

Locatie:
Utrecht

Technologieën:
Maven, Docker, Angular, Java, DevOps, CI/CD, Hibernate, Spring, SQL, JSON

KVK HOUVAST VOOR ONDERNEMERS

Find all these vacancies and more

at Devitjobs.nl

NLJUG INNOVATION AWARD 2023

JAVA DEVELOPERS MAKEN HET VERSCHIL
IN INNOVATIEVE TECHNOLOGISCHE ONTWIKKELINGEN

Is jouw bedrijf een innovator en heb jij een succesvol, innovatief, verantwoord en/of omvangrijk project? Dan is dit jouw kans om in de schijnwerpers te staan! De NLJUG Innovation Award wordt voor de vijfde keer uitgereikt tijdens J-Fall en we zijn op zoek naar het meest innovatieve project. Elke NLJUG businesspartner dat een project inschiet, krijgt tevens 2 J-Fall conferentietickets! Maak kans op een timeslot binnen het programma van J-Fall en inschrijving door je project voor donderdag 14 september 2023 in te dienen!

Winnaars

- **2018** Yolt
- **2019** Tennet
- **2021** Picnic
- **2022** Omoda

INSCHRIJVING

Vanaf nu tot en met 14 september 2023 kan een project en/of persoon worden voorgedragen. Ken jij een bedrijf, project en/of persoon die dit jaar de NLJUG Innovation Award zou moeten winnen?

Schrijf je in op nljug.org/nljug-innovation-award/

Introduction to the Data Cloud

KEYNOTE **SPEAKER: ANDRE MOLENAAR**

"I am extremely optimistic with this great opportunity at Snowflake, to co-create value addition for our partners, their customers, and the customers' customer in turn. I look forward to share my passion for the Snowflake technology with the Teqnation audience."

André joins us with deep experience in Advanced Analytics and Big Data technology. After completing his degree in Information Science at the University of Amsterdam, Andre worked for renowned technology leaders like Oracle, Cloudera and AWS.

The introduction to The Data Cloud tells the Snowflake story across 7 pillars, to make the Data Cloud truly different from anything that's been available in the past.

We've taken the learnings from familiar technologies, building on the shoulders of giants, to deliver a new approach that removes many of the challenges and limitations of the past. This has resulted in a powerful solution where you can bring all data together, have a single engine to access it all, and support all types of workloads, all governed at the core.

Additionally, all that is now available globally - connecting your teams, across regions, and clouds through a seamless experience. It delivers all this as a simple, self-managed platform that you can trust just works, even as we expand support into flexible and powerful ways to program against the Data Cloud with SQL, Java, Python, and more.

And finally, it unlocks possibilities for your business whether it's consuming the most relevant data, services, and apps, or even opening up new revenue streams through the Marketplace.

Using these pillars as our guide, we'll showcase both what makes the Snowflake platform unique, and how companies like yours are benefiting from the Data Cloud.

Who let the data pipelines out?

DEEP-DIVE **TATIANA PETRACHE**

Have you ever tried to collaborate on a data project with your team, only to have it devolve into a chaotic mess of multiple copies of data and governance nightmares? Well, fear not!

With the power of Snowpark and Snowflake you can streamline your data engineering and data science use cases and bring your data pipelines to your data, where they belong. All this while maintaining your sanity and data governance.

How? Snowpark brings DataFrame-style programming and functions available for Python, Scala and Java to help you build and deploy data pipelines at scale. And if you're still not convinced, just wait until you see the demo we have planned!

Tatiana Petrache, Solution Engineer at Snowflake
Tatiana has over 12 years of experience in the design, development and deployment of data projects ranging from Data Warehousing, Business Intelligence, Big Data and Machine Learning. Applied experience in building and scaling data science products. Enthusiastic about simplifying Data Science in the cloud using Snowflake. She is naturally curious with a wide range of interests in various topics about technology, culture, neuroscience or music.

Maintaining and repairing IT Systems with AI

Artificial Intelligence for IT Operators (AIOps) is helping Rabobank teams to detect and correct incidents in their application landscape and has proven to deliver superhuman performance on some maintenance task. Using state-of-the-art machine learning technology Rabobank is working on its vision to deliver rock-solid services with self-healing capabilities.

In this talk Joep Daandels will share how teams are implementing AI in their daily operations and lay out how we leverage data from existing monitoring systems and business processes to build a self-healing application.

Don't miss this chance to get a glimpse into the future and see how AI can enhance the performance of your teams.

Rabobank

Joep Daandels, Rabobank, DevOps Engineer

Joep is an enthusiastic DevOps Engineer from Maaskantje, The Netherlands. He graduated as a software engineer from the Avans University of Applied Sciences and is working for Rabobank since 2010. Joep has been involved in several major projects for the bank (SEPA, Rabobank Order Manager, Instant Payments) in the payment solutions domain and has been focused at non-functional requirements, like performance and stability. In recent years he specialized himself in machine learning and developed a state-of-the-art deep-learning solution for anomaly detection to enhance IT operations with AIOps. With his team Joep is currently working on the vision to deliver self-healing IT applications to automatically remediate potential issues in the IT landscape. In his spare time Joep loves to relax and read a good paper of study some new interesting technology.

SELMA HELPS 1.3 MILLION PEOPLE MAKE ENDS MEET

How many people do you want to help?

Selma Kubat, IT project manager at Netcompany, helped to build the system that pays out national and social pensions to over 1.3 million people.

Do you want to work on IT projects that make a real difference to society?

Take a look at our open roles:

netcompany

In control

Psychology and computer science, a match made in heaven or recipe for disaster? In this talk you will learn more about how psychological effects impacts software development. You may think, is there even an impact since it is such an exact discipline? Yes, there definitely is! For there are requirements to gather, expectations to manage, colleagues to work with and of course writing code still remains human work.

{ A FEW EXAMPLES }

Developing software is complex and Agile/Scrum is chock-full of group processes and interactions. There is definitely a lot of psychology going on. I will share what I learned at Spotify and I will point at a harmful side effect of the continuous improvement cycle in scrum and how to avoid it.

And did you ever have to request something technical like a mail server or access through a firewall? More and more you will be faced with a DIY (Do-It-Yourself) process. It sounds great but there is a catch. In the session you will learn the pitfalls of outsourcing a complex process to the consumer.

Finally two psychological constructs: Groupthink and Flow. Groupthink can lead to false consensus. I will explain why it is dangerous and the steps you can take to protect you and your team. Flow, on the other hand, is a great psychological concept that can help you. You will learn how to recognize it and how to use it to your advantage.

This is just a touch of what you can expect from this session. The session is interesting and relevant for everyone. It does not matter which programming language you use and what role you have, as long as you are involved in the software development process.

Now, if you want to know whether it is a match made in heaven or recipe for disaster... Come to the session and decide for yourself!

ABN·AMRO

Explore what Apryse has to offer

Do you want to create, edit, and convert digital documents in your applications or company workflow? Are you interested in building powerful solutions with ease using elegant, well-documented, cross-platform APIs? Then the Apryse Developer Suite might be the perfect solution for you.

Apryse, formerly known as PDFTron and now including iText, takes document solutions to the next level, making work better and life simpler.

As a global leader in document processing technology, Apryse gives developers, enterprises, and small businesses the tools they need to reach their document goals faster and easier. Our product portfolio includes server, mobile and web SDKs, iText, and XODO. Apryse technology works with all major platforms and a wide variety of unique file types. And are trusted by over 20,000 innovative start-ups, governments, and Fortune 500 companies, including Microsoft, Canon, and IBM. And no matter who you are, we have a solution for you.

Would you be interested in hearing more? Check out apryse.com for more info or to book your free trial. Or come say hi to us at TEQnation on 17 May!

Our developers will be present at the biggest multitrack Developer Conference of the Netherlands. And we happily invite you to stop by our booth to say hi if you're also attending – or to try your luck with our code challenge and win one of our amazing prizes!

See you there?

apryse

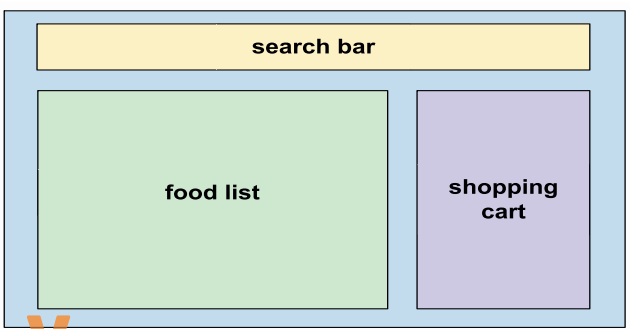
MICRO FRONT-ENDS IN DE PRAKTIJK

Misschien heb je al gehoord van micro front-ends, maar wil je graag weten hoe je het kan gebruiken om een volgende stap te maken in jouw schaalbare front-end applicatie. In dit artikel zal eerst ingegaan worden op uitleg over verschillende vormen van micro front-ends, zoals die al jaren bestaan en ook van een modernere aanpak op basis van module federation. De verschillende oplossingen zijn aangevuld met best practices, pitfalls en praktijkvoorbeelden.

Martin van Es is eind jaren '90 begonnen met het maken van websites en is alweer ruim 18 jaar professioneel bezig als software developer.

Stel je voor dat je aan een groot softwareproject werkt met een team van developers. Zolang alle code uit één bron komt, bijvoorbeeld één repository of één deliverable, dan spreken we over een 'monoliet'. Of het nu front-end is of back-end, er komt al snel een moment dat de grootte van de codebase zelf een uitdaging gaat vormen voor het onderhoud. Bij API-development kan dit worden opgelost door de codebase te splitsen in microservices, elk met een eigen verantwoordelijkheid en vaak onafhankelijk op te schalen in de infrastructuur. De grens om een monoliet te splitsen in microservices is vaag, maar het kan zeker helpen als je met meerdere teams tegelijk aan een codebase werkt.

Micro front-ends is een verzamelnaam van oplossingen om kleinere front-ends te gebruiken in één applicatie. Ze kunnen daarom gebruikt worden om een monolitische front-end codebase te splitsen en daarmee vergelijkbare voordelen te halen als bij microservices: de verantwoordelijkheid kan worden verdeeld over meerdere teams, en bij sommige oplossingen kunnen meerdere technische stacks door elkaar worden gebruikt (bijvoorbeeld React en Angular).



Afbeelding 1: micro front-end diagram.

Micro front-ends zijn niet nieuw: hoewel de term pas een paar jaar gebruikt wordt, zijn sommige oplossingen die onder deze noemer vallen al veel gebruikt in de praktijk of zelfs al verouderd. Als je in een enterprise-omgeving aan front-end code hebt gewerkt, dan heb je waarschijnlijk al met micro front-ends te maken gehad.

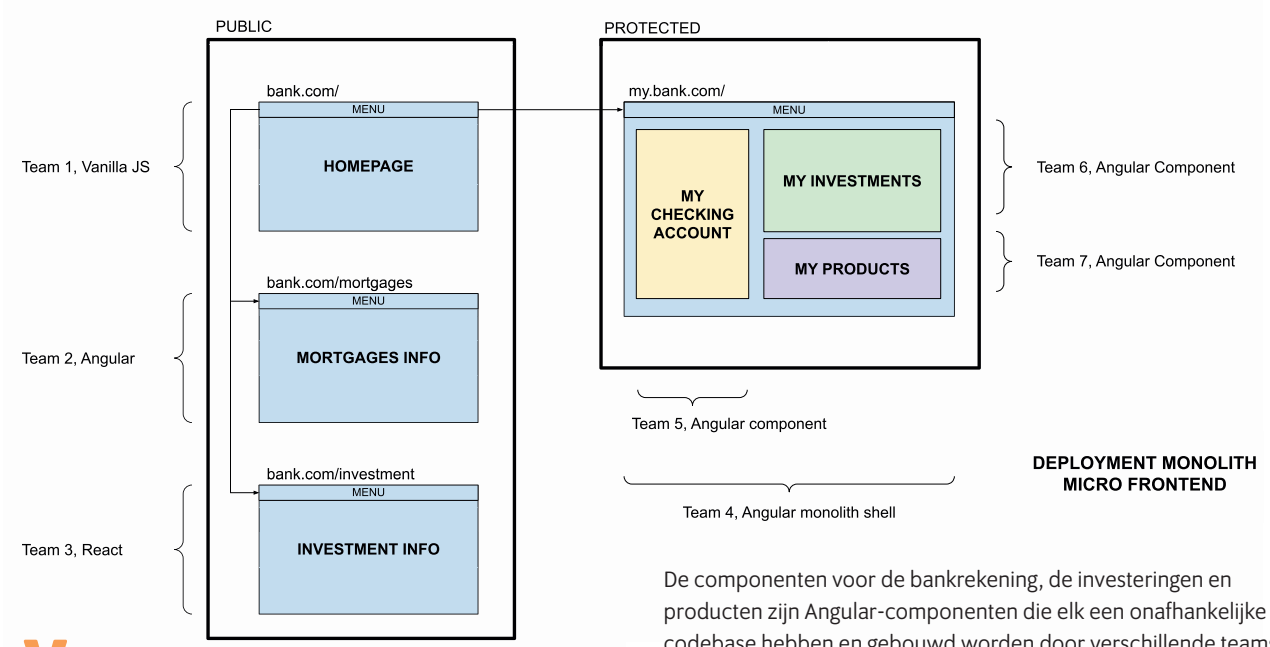
Neem het voorbeeld in Afbeelding 1 als uitgangspunt; een webshop waar je eten kan bestellen.
Functioneel bestaat dit uit drie componenten: een zoekbalk, een lijst van etenswaren en een winkelmandje waar je jouw bestelling mee kan plaatsen. Daarnaast is er nog de code die deze aan elkaar lijmt, de 'schil'.

{ HYPERLINKS }

Je zou ervoor kunnen kiezen om losse pagina's te maken voor de zoekbalk, de lijst en het winkelmandje en deze met hyperlinks aan elkaar te knopen. Dit is in principe al een micro front-end: de pagina's kunnen los worden onderhouden en eenvoudig in verschillende stacks worden geïmplementeerd. Je ziet echter wel nadrukkelijk dat de pagina wordt herladen bij het navigeren en communicatie tussen de componenten is niet mogelijk (ze staan immers niet op dezelfde pagina).

{ IFRAMES }

Om het zichtbaar herladen weg te nemen en communicatie tussen de componenten mogelijk te maken, zou je ervoor kunnen kiezen om de componenten met iframes op één pagina in te laden. De isolatie is hetzelfde als bij losse pagina's dus je kan prima React en Angular in verschillende iframes inladen. Daarnaast kan JavaScript in elk iframe bij de Document Object Model van de andere iframes en zo is het mogelijk om tussen de iframes te communiceren.



Afbeelding 2: micro front-end diagram – hyperlinks en monoliet.

De huidige implementatie van het iframe-element kennen we sinds HTML5, dus al meer dan 10 jaar en daarom wordt het ondersteund door alle browsers.

Helaas hebben iframes een slechte reputatie gekregen, en niet zonder reden: het kan misbruikt worden voor clickjacking en daarom zijn er allerlei veiligheidsrestricties opgelegd, waardoor het steeds lastiger wordt om een configuratie van iframes werkend te houden. Ook het maken van een responsive ontwerp is beperkt, omdat CSS features zoals media queries en flexbox niet werken over iframes heen.
Een uitzonderlijk voordeel van iframes is het insluiten van bronnen van derde partijen die controle willen houden over hun bron, zoals bijvoorbeeld Youtube-videos. Dit wordt nog steeds via iframes gedaan, maar kan voor de afnemer wel betekenen dat er uitzonderingen moeten worden toegevoegd aan de Content Security Policy om frames van deze derde partij toe te staan.

{ DEPLOYMENT MONOLITH }

Een veel gebruikte oplossing voor enterprise-omgevingen met een grote front-end codebase, is het ontwikkelen van componenten per team en deze in de buildpipeline te combineren tot één applicatie. Dit zou je een 'deployment monoliet' kunnen noemen. Laten we hiervoor kijken naar een wat complexer voorbeeld dan de voorgenoemde winkel, namelijk een bank (Afbeelding 2).
Het rechterdeel is de persoonlijke bankieromgeving voor eindgebruikers. De 'schil' wordt hier een 'monolith shell' genoemd. De schil is een normale front-end applicatie, in dit geval in Angular.

De componenten voor de bankrekening, de investeringen en producten zijn Angular-componenten die elk een onafhankelijke codebase hebben en gebouwd worden door verschillende teams. Het resultaat wordt steeds gebuild en als deliverable naar een NPM-registry verzonden. De schil importeert deze afhankelijkheden op dezelfde manier, net zoals elke derde-partij NPM-module. De integratie van de componenten wordt dus in de buildfase van de schil gedaan en wordt dan als een monoliet opgeleverd, vandaar de naam deployment monoliet.

Zoals je kan zien, staan aan de linkerkant ook pagina's die met hyperlinks aan elkaar zijn verbonden. Verschillende micro front-end oplossingen kunnen dan ook prima door elkaar heen worden gebruikt.

{ WEB COMPONENTS }

Bij een deployment monoliet zit je dus wel vast aan één (versie van) een framework. Dus als je meerdere versies van Angular zou willen gebruiken in één applicatie, of bijvoorbeeld Angular en React, dan is dit niet voldoende. Je zou er dan voor kunnen kiezen om de micro front-ends te verpakken in een Web Component.

Web Components zijn gemaakt om framework agnostische eigen elementen toe te voegen aan een applicatie, maar omdat Web Components browsernative zijn, kan het ook gebruikt worden om een component te maken van bijvoorbeeld een React-component. De React-afhankelijkheden zitten dan in dit Web Component. Dit is gelijk ook het grootste nadeel: als je meerdere Web Components op de pagina laadt, dan worden deze afhankelijkheden potentieel ook meerdere malen ingeladen. Dit is een redelijk populaire oplossing voor micro front-ends. Bijvoorbeeld Angular biedt de mogelijkheid om componenten te verpakken als Web Components onder de naam 'Angular Elements'.

{ MODULE FEDERATION }

Om meerdere (versies van) frameworks tegelijk te gebruiken en geen afhankelijkheden te dupliceren, kan je sinds een tijdje ook

gebruikmaken van Module Federation. Hierbij worden de componenten runtime, dus in de browser, geïntegreerd in plaats van in het buildproces.

In tegenstelling tot Web Components is Module Federation expliciet ontwikkeld met het integreren van onafhankelijke componenten als doel. Daarom biedt het bijvoorbeeld een oplossing voor het ontdubbelen van afhankelijkheden. Stel, er zijn twee componenten die gebruikmaken van React en aangeleverd worden via Module Federation. Het is dan mogelijk om dit aan te geven in de configuratie. De module bundler (bijvoorbeeld Webpack of Vite) zorgt er dan voor dat React één keer wordt ingeladen in de browser en dat beide componenten hierbij kunnen.

Module Federation is het afgelopen jaar sterk in opkomst. Hoewel het sinds het verschijnen van Webpack 5 mogelijk is om met de hand Module Federation in te regelen, wordt het nu ook omarmd door ondersteunende tooling. Voor enterprise front-ends is het meestal wenselijk om gebruik te maken van standaardconfiguraties voor het valideren en builden van de codebase. Hoewel de Angular CLI en Create React App (nog) geen ondersteuning bieden voor Module Federation, worden er grote stappen gemaakt door Nx, een platformonafhankelijke toolkit voor front-end monorepos.

Voordat je begint met het op grote schaal gebruiken van Module Federation moet je wel goed nadenken over een interface, waarmee de componenten met elkaar kunnen communiceren. Hiervoor zijn nog geen standaarden. Je zal dus zelf een contract moeten bedenken voor bijvoorbeeld het uitwisselen van applicatie brede state en navigatie tussen componenten. Dit geldt overigens net zo goed voor Web Componenten, maar dat is dan ook niet ontwikkeld met het encapsuleren van framework-componenten in gedachten.

{ TOOLING }

Ongeacht de gekozen techniek, is het wel aan te raden om gebruik te maken van bepaalde ondersteunende tooling om een schaalbare applicatie te maken met micro front-ends. Aangezien de micro front-ends onafhankelijk zijn van elkaar en mogelijk bij verschillende teams in gebruik zijn, kan versiebeheer natuurlijk in losse repository's gedaan worden.

Meestal is het echter minder bewerkelijk om gebruik te maken van een monorepo, bijvoorbeeld op basis van Nx. Het voordeel hiervan is dat linten, testen en builden maar één keer moet worden ingericht voor de monorepo en dit vereist dus minder onderhoud.

Daarnaast verzorgt Nx tijdens het ontwikkelen het koppelen van de componenten. Als de componenten in losse repository's zouden zitten, en er is een afhankelijkheid tussen, dan zou je dat handmatig met npm-link moeten koppelen.

In een Nx-monorepo kan je meerdere apps hebben, en meerdere library's. Nx kent verschillende type library's met verschillende toepassingen. UI-library's zijn voor UI-componenten (dus zonder businesslogica), zoals een invoerveld of een knop. Data Access library's zijn voor servicecalls en applicatie brede state. Feature library's bevatten een afgebakend stuk businesslogica. Hiermee biedt Nx handvatten voor een conventie voor het splitsen (of voorkomen) van een monoliet. Micro front-ends zouden dan thuishoren in een app, of eventueel in een feature library.

Verdiep je bij het gebruik van Nx ook in Affected Projects. Zeker naarmate er meer apps en library's bij komen, is het de beste manier om builds sneller te laten verlopen. Er wordt bij een affected build door Nx gekeken naar de geschiedenis in versioncontrol en alleen die apps en library's worden gebuild waar dat nodig voor is.

Nx biedt ook ondersteuning voor Storybook. Hiermee kan een webapp worden gegenereerd met een overzicht van de ontwikkelde componenten in een UI-library.

Als er met verschillende teams in één monorepo wordt gewerkt, is het nuttig om CODEOWNERS-bestanden toe te voegen. Hiermee kan je per directory aangeven wie er reviews moeten doen op pull-requests op wijzigingen in code in deze directory. Bijvoorbeeld in het geval van een monorepo met meerdere apps, kan in de root van app 1 een CODEOWNERS-file staan met de teamleden van team 1, die app 1 beheren. Als een andere ontwikkelaar een pull-request aanmaakt, waarbij code in de app 1-directory is geraakt, dan moet deze pull-request ook worden gereviewd door iemand in team 1.

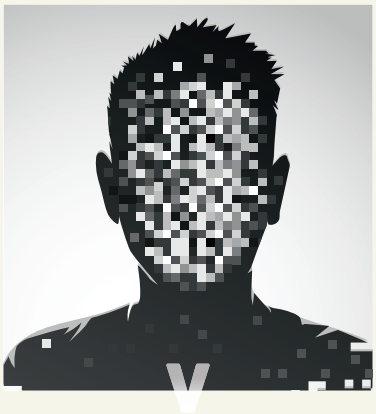
Naarmate front-end applicaties steeds complexer worden, verwacht ik dat de rol van micro front-ends steeds meer onderdeel wordt van het curriculum voor front-end developers. Met name Module Federation en het ecosysteem daaromheen (zoals Nx) wordt nog niet effectief toegepast. Ik hoop dat er op een redelijk korte termijn abstracties komen voor Module Federation. Dat zal de drempel voor goed gebruik lager maken en daarmee de deur openzetten naar applicaties op een moderne stack, die bedoeld is om moderne schaalbare applicaties te maken. <

{ REFERENTIES }

- https://mdworld.nl
- https://www.w3.org/html/wg/spec/the-iframe-element.html
- https://developer.mozilla.org/en-US/docs/Web/Web_Components
- https://angular.io/guide/elements
- https://webpack.js.org/concepts/module-federation/
- https://www.angulararchitects.io/en/aktuelles/a-software-architects-approach-towards/

COLUMN

IT Driven earth



By λ

8 Billion people are living on a flat earth. To make that work you need a lot of machinery. And we of all people know that IT is the key to make that work. We are part of this small community of 6 billion people that anxiously try to hide the truth about this. All people in IT next to all people working in (or related to someone working in) government, science, education, media, airways, shipping, farming, mining and so on. But how do we make this work and keep it a secret? I'll share my insights here.

So welcome all flat earth patriots. I know I'm talking to an intimate crowd of people who know the truth and can keep a secret for 2 billion others (sort of). We of all people know that everything is driven by IT and IT needs to be coded. That is why we need to be in on it.

But over time a lot of information gets lost and because we cannot share this information on the public internet, we need some way to stay up-to-date. So, I chose this medium to share my knowledge and thoughts with you. If you have more information that we should add or questions about this all, then please contact me. But please do any online communication in ROT13 to make sure nobody is able to read it!

But let me start with introducing myself. Okay, I can't tell you my name, obviously, that would be too dangerous, so you can call me Lambda. I started learning math in primary school and in 8th grade got the first steps in calculus. Around 8 was also the time I became first class citizen. My first coding was basic on a commodore 64. But when I started working in the real world, Java seemed to be inevitable. To be honest things really became interesting for me when flatMap came around... But this column is not about me, it is about our big secret.

I don't exactly remember when learning the truth, but I think the flatness of all the games on the commodore 64 were already a hint. It is not that strange that Minecraft is a big hit, because it is hiding the truth in plain sight, right? When did you find out the real truth? And who told you to stay quiet about it?

Anyway, so why write to you here? Well, let me tell you, Java is very important in this context. I mean we needed Sun for Java to come to life. But then the Oracle came. You might think it was the Oracle of Delphi, but deep in our hearts, we all know that's not true.

After some digging I found that it might originally be the Oracle of Dodona [1], which is still recruiting and training people in Belgium [2]. We all know that you can ask Oracle anything, but what it gives you is tools that aren't answers, but basically puzzles you need to solve. And asking questions about those tools most of the time keeps you in the dark.

But Oracle then acquired Sun and kept it shining. But that was not the first interesting acquisition...

Let's have a look at the list of acquisitions and see what we can find. Have you ever heard of NetForce? Tom Clancy wrote a book about this secret FBI agency that basically is responsible for controlling the content of the internet. January 16th 2002, Oracle acquired NetForce.

To gain more control over light, both night and day, Oracle bought Hyperion [3,4] in 2007, which is basically the force controlling Sun and Moon to circle around under the dome we call sky. And let's not think about the power Oracle gained by acquiring BEA [5]. 2019 is the year Oracle acquired Oxygen...

I know... there is a lot to talk about and too little space and time (although spacetime does not exist of course, but I'll come back to that). It was nice getting to know you and talk to you. But if we speak again, let's make sure we know we are speaking from the same background without sharing it over and over again. If we meet, just great me with 'flatMap' and we will both know.

So flatMap...

LINKS

1 https://nl.wikipedia.org/wiki/Orakel_van_Dodona
2 https://dodona.ugent.be/
3 https://nl.wikipedia.org/wiki/Hyperion_(mythologie)
4 https://en.wikipedia.org/wiki/Hyperion_(Titan)
5 https://en.wikipedia.org/wiki/Bea

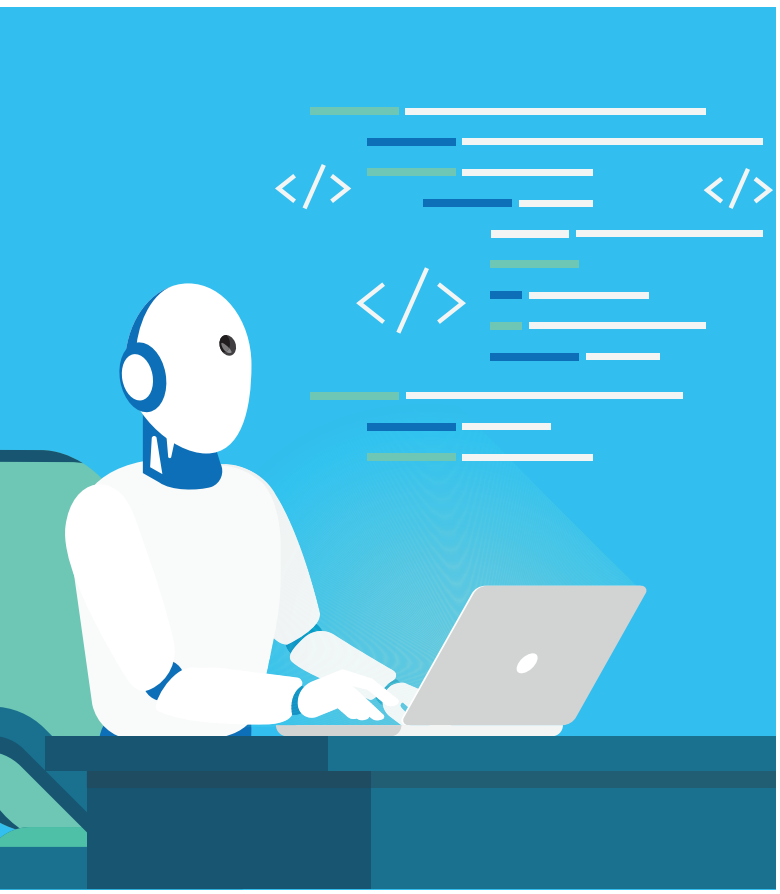
TESTAUTOMATISERING MET ROBOT FRAMEWORK

Robot Framework is een opensource 'test automation framework' die zijn oorsprong vindt in Nokia Networks. Aangezien Robot Framework (RFW) ook genoemd staat op de Java Roadmap 2022 [1], is het een goed moment om dit test-framework nader te bekijken.

Gunter Rotsaert (@mydevlprplanet) is een Systems Engineer bij TriOpSys en deelt graag kennis via zijn blog.

{ WAAROM ROBOT FRAMEWORK? }

RFW-testcases hebben een eenvoudige syntax en zijn 'keyword driven'. Dit betekent dat ze zeer goed leesbaar en begrijpelijk zijn. Dit zorgt er ook voor dat je geen aparte testdocumentatie nodig hebt voor het beschrijven van fysieke testgevallen. De RFW-testcases zijn platte tekstbestanden, waarmee ze uitermate geschikt zijn om opgenomen te worden in een versiebeheersysteem, zoals Git (met alle voordelen van dien).



RFW is gebaseerd op Python, maar hierdoor is er zeker geen afhankelijkheid van de programmeertaal waar je applicatie in geschreven is. Je kan namelijk elke applicatie testen, zolang deze maar voldoende generieke interfaces heeft die je kan aanroepen vanuit je testcases. Daardoor is RFW erg krachtig en flexibel. Omdat RFW gebaseerd is op Python, is het framework eenvoudig uitbreidbaar en zijn de mogelijkheden zowat onbeperkt, zoals later in dit artikel uitgelegd wordt.

Als je kijkt naar de testpiramide, dan situeert RFW zich vooral op het niveau van de systeemintegratietesten. Er is ook een integratie met Selenium en Playwright. Hierdoor is het ook mogelijk om end-to-end testen uit te voeren. Verder is er een breed toepassingsgebied. Zo wordt RFW gebruikt voor het testen van luchtvaartsystemen, dataprocessingsystemen, Internet of Things systemen, enz.

Bovendien is RFW opensource en kan het gebruikt worden zonder licentiekosten.

{ BASISFUNCTIONALITEIT }

Als je meer wil leren over RFW, dan is de officiële website van RFW [3] een goed startpunt. In dit artikel zullen een aantal basisconcepten uitgelegd worden, maar RFW omvat veel meer functionaliteit dan in een artikel uitgelegd kan worden. De officiële documentatie is van goede kwaliteit, legt helder de concepten uit en de website bevat ook links naar interessante library's en ander referentiemateriaal.

De voorbeeldapplicatie [2] die gebruikt wordt, is een standaard RESTful-applicatie om customers te beheren.

L1

```
*** Settings ***
Documentation   Basic test for Customer
Library   Collections
Resource  customer.resource


*** Variables ***
&{CUSTOMER_JOHN_DOE}  firstName=John  lastName=Doe  email=john.doe@mail.nl


*** Test Cases ***
Add Customer John Doe
    [Documentation]  Add a Customer
    [Setup]  Verify Customers Length  0
    ${customer}=  Add Customer  ${CUSTOMER_JOHN_DOE}
    Verify Customer Exists  ${customer}
    [Teardown]  Delete Customer  ${customer}[customerId]


*** Keywords ***
Verify Customer Exists
    [Documentation]  Verify whether provided Customer exists
    [Arguments]  ${customer}
    Verify Customers Length  1
    Get Customer  ${customer}[customerId]

Verify Customers Length
    [Documentation]  Verify length of Customers
    [Arguments]  ${expected_size}
    ${customers}=  Get Customers
    ${length}=  Get Length  ${customers}
    Should Be Equal As Integers  ${expected_size}  ${length}
```

De testen die gebruikt worden in deze en de volgende paragrafen zijn terug te vinden op GitHub [4].

De basistest (zie Listing 1) zal een customer toevoegen en verifiëren of deze ook succesvol is toegevoegd. Dit is voldoende om een aantal concepten en onderdelen toe te lichten:

- 'Settings': wordt gebruikt om metadata, zoals documentatie vast te leggen en om externe library's toe te voegen (net als imports in Java).
- 'Variables': laten je toe om constanten te definiëren die verder in de testcases gebruikt kunnen worden.
- 'Test Cases': deze sectie bevat de testcases zelf. De testcases in een bestand behoren automatisch tot een 'Test Suite' (een Test Suite kan echter ook bestaan uit meerdere bestanden of directory's).
- 'Keywords': keywords zijn herbruikbaar door meerdere testcases, vergelijkbaar met methods in Java.

{ TEST CASES }

Neem eerst een kijkje naar de testcase zelf. Deze bestaat ook uit een aantal onderdelen:

- 'Documentation': de documentatie behorende bij deze testcase, dit kan een tekstuele algemene omschrijving bevatten van de test.
- 'Setup': een setup waar je de randvoorwaarden voor de testcase in vastlegt. In deze testcase wordt eerst gecontroleerd of de lijst van 'customers' wel leeg is.
- Een sectie voor de testcase zelf. Hier is het van belang om in goed leesbare taal de testcase vast te leggen. In de test wordt 'customer John Doe' aangemaakt en vervolgens gecontroleerd of deze inderdaad opgehaald kan worden.
- 'Teardown': de Teardown wordt gebruikt om ervoor te zorgen dat de applicatie weer in dezelfde situatie terechtkomt als voor het starten van de testcase. De Teardown wordt altijd uitgevoerd, of de test nu succesvol is of niet.

L2

```
*** Keywords ***
Add Customer
    [Documentation]  Add a Customer
    [Arguments]  ${json}
    ${headers}=  Create Dictionary  Content-Type=application/json
    ${resp}=  POST  url=${CUSTOMER_ENDPOINT}  headers=${headers}  json=${json}  expected_status=200
    RETURN    ${resp.json()}
```

{ KEYWORDS }

De testcase maakt gebruik van een aantal keywords, zoals 'Verify Customers Length' en 'Verify Customer Exists'. Hier komt de kracht van RFW naar voren. Je kan herbruikbare onderdelen, zoals 'Verify Customers Length' vatten in keywords om zo duplicatie tegen te gaan, maar je kan deze ook gebruiken om meer leesbare testen te creëren en zo ervoor zorgen dat je testcase zelf op een hoog abstractieniveau blijft en makkelijker te begrijpen is voor je klant. In de keywords zie je met Arguments nog een extra onderdeel terugkomen. Arguments zijn niets meer dan argumenten die je kan meegeven aan een keyword, net zoals je de argumenten in een Java-method meegeeft.
Verder zie je dat hier gebruikgemaakt wordt van een aantal built-in keywords [5] die standaard zonder specifieke import beschikbaar zijn, zoals 'Get Length' en 'Should Be Equal As Integers'.

{ SETTINGS }

In de 'Settings' zie je twee soorten imports. De import van 'Collections' die voorafgegaan is door 'Library'. Dit betekent dat dit een import is van een Python-library, in dit specifieke geval van een built-in RFW-library [6]. De built-in RFW-library's bieden je al heel wat functionaliteit die je zonder meer kan gebruiken in je testcases.

De import van `customer.resource` is een import van een resource-bestand. In een resource-bestand kan je keywords verzamelen die je kan delen over meerdere testbestanden. Dit resource-bestand bevat dezelfde onderdelen als een testcasebestand, behalve het onderdeel 'Test Cases'.

In Listing 2 wordt er gebruik gemaakt van de RequestsLibrary [7] om de API van de applicatie te benaderen.

{ TEST UITVOEREN }

Een test uitvoeren kan met behulp van een eenvoudig commando:

```
$ robot basic_test.robot

==============================================
Basic Test :: Basic test for Customer
==============================================
Add Customer John Doe :: Add a Customer | PASS |

----------------------------------------------

Basic Test :: Basic test for Customer | PASS |
1 test, 1 passed, 0 failed

==============================================

Output: /./javamagrobotframework/output.xml
Log: /.../javamagrobotframework/log.html
Report: /.../javamagrobotframework/report.html
```

Zoals je in de output kan zien, levert dit drie automatisch gegenereerde testrapportages op.

{ DATA-DRIVEN TESTS }

Naast de keyword-driven testcase, zoals hierboven beschreven is, is het ook mogelijk om een data-driven testcase te maken. Dit kan bijzonder handig zijn wanneer je eenzelfde soort testcase over een grote dataset wil uitvoeren. Een voorbeeld hiervan vind je terug in Listing 3.

De testcase 'Create Customers' maakt nu gebruik van een template 'Create Customer' die als keyword is vastgelegd. Het template heeft een dictionary als argument. De testcase roept nu geen keywords aan, maar bevat per regel de input voor het argument van het keyword 'Create Customer'. Wanneer je deze testcase uitvoert, zie je dat elke regel met data uitgevoerd is alsof het een afzonderlijke aanroep van het keyword zou zijn.

{ EIGEN LIBRARY'S MAKEN }

RFW biedt heel wat built-in library's en er zijn ook heel wat opensource initiatieven die RFW-library's aanbieden. Daarnaast kan je heel wat voor elkaar krijgen met behulp van keywords.

Maar wat als je iets voor elkaar wil krijgen waar nog geen library voor bestaat of wat met keywords lastig te ontwikkelen is? In dat geval kan je je eigen library maken en het eenvoudigste is om deze te ontwikkelen met Python. Dit betekent tevens dat de RFW-mo-

L3

```
*** Test Cases ***
Create Customers
    [Documentation]  Create several customers
    [Template]  Create Customer
    firstName=John  lastName=Doe  email=john.doe@mail.nl
    firstName=Foo  lastName=Bar  email=foo.bar@mail.nl
    [Teardown]  Delete Customers


*** Keywords ***
Create Customer
    [Documentation]  Create a Customer
    [Arguments]  &{customer}
    ${resp_json}=  Add Customer  ${customer}
    Verify Customer Exists  ${resp_json}
```

L4

```
from typing import Dict

class CustomerLibrary:

    @staticmethod
    def verify_customer_in_list(customers, email: str="") -> bool:
        """
        Verify whether a customer with the email is present in the list.
        """
        if not customers:
            return False
        for customer in customers:
            if (email and customer["email"] == email):
                return True
        return False
```

gelijkheden onbeperkt zijn, zolang je ze ook kan ontwikkelen met Python.

Neem het volgende voorbeeld, waarbij je wil controleren of een customer met een bepaald e-mailadres bestaat. Je kan waarschijnlijk het keyword 'Verify Customer In List' ontwikkelen binnen RFW, maar de vraag is of dit de RFW-code wel goed leesbaar maakt.

In dit geval kan je een eigen CustomerLibrary.py aanmaken (zie Listing 4). Het enige wat je in het RFW-script hoeft te doen, is de library toevoegen aan de 'Settings'-sectie. De naam van de Python-method `verify_customer_in_list` kan je vervolgens als keywod 'Verify Customer In List' gebruiken. Dit maakt dat het toepassingsgebied van RFW groot is, zodat je er nagenoeg elk type applicatie mee kan testen. <

{ REFERENTIES }

1 https://github.com/devoxx/JavaRoadmap2022
2 https://github.com/mydeveloperplanet/javamagrfwbackend
3 https://robotframework.org/
4 https://github.com/mydeveloperplanet/javamagrobotframework
5 https://robotframework.org/robotframework/latest/libraries/BuiltIn.html
6 https://robotframework.org/robotframework/latest/libraries/Collections.html
7 https://marketsquare.github.io/robotframework-requests/doc/RequestsLibrary.html

BOOST YOUR PRODUCTIVITY

Engineer your engineering tools

As engineers we love to solve problems. Even better if we can solve problems with code. And the best solutions are those that have high quality code. We love this to the point where we spend hours upon hours on tests and refactoring. One place where we saw little of this love was in our engineering tools. We had many aliases and scripts lying around which we used, but the code for these was in a pretty bad shape. We decided to roll up our sleeves and dust off the old bash routines.

Guus de Wit is a Java/Kotlin developer at Blue4IT. With a passion for puzzles and a background in science, he strives for elegant solutions to complex problems.

Martin Kanters is a Java/Kotlin developer at JPoint in Utrecht and Apache Maven committer. Working together in teams on complex applications keeps him going to work every morning with a big smile.

After a couple of months of spending spare time on our new engineering tools, we figured we had something that could benefit the whole company. We presented our new product to the business and voilà! We now have a working product which we actively develop and maintain during sprints. It is an extensible command line application written in Kotlin that can be used to interact with the tools you use on a daily basis. We'll explain how it works and we will also provide some tips on how to start up a similar project yourselves!

{ THE PROBLEM }

We use many different tools during our daily work as software developers. Think of Git servers, ticket backlogs, logging tools etc. Switching between those can be cumbersome, as you constantly have to context switch which might lead you to lose focus. Not only that, but they can be notoriously slow, making it even more infuriating to have to work with them. We actually took this to the test and measured how long it took to create a pull request in Azure DevOps. It took about 30 seconds...

Now, 30 seconds is not that bad. But creating a pull request is not something you only do once. Let's say you create about two per day. Over the course of a year that adds up to about half a work day. Half a day lost, just because our tools are slow! And creating pull requests is only one example. We can save days, or even weeks of time by making this process shorter. You can do the math for a company consisting of hundreds of developers.

{ OUR SOLUTION }

Of course you can already imagine where this is going. Repeating slow, boring and tedious steps just calls for automation! So we started our journey by automating anything in sight. We wrote custom bash scripts, aliases and Alfred workflows on the fly (Alfred is an alternative for Spotlight on MacOS with features like hotkeys, keywords and text expansion) (Image 1).

The results were glorious! Except...

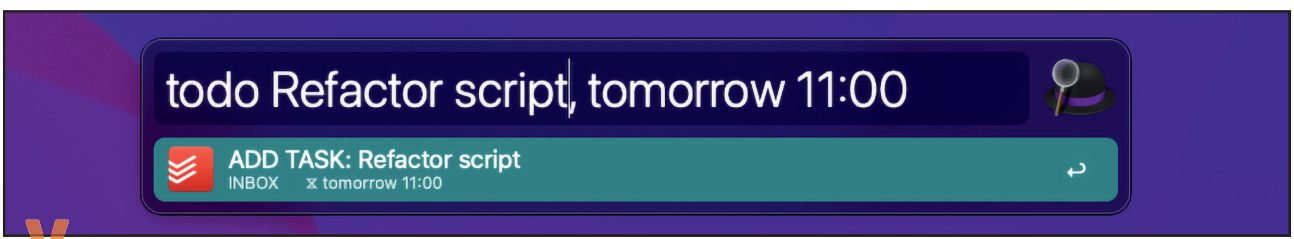


Image 1. Example of an Alfred workflow which adds a task in Todoist.

We didn't really have any standards in place. The automations worked, but mostly just 'on my machine'. They were pretty much all static scripts and hard to customize. And unit tests? Those weren't even on the radar. The result was a messy pile of unrelated pieces of code, scattered around our home folders, which we could hopefully all still find when we had to instant message them to a new team member.

So, we wanted to make it better. We collected the automations we had and came up with a plan to streamline the development process. Instead of writing separate scripts, we made one CLI that connects to all the systems and tools we wanted to automate. With that, we had a central place to put advanced features like auto suggestions based on API calls and context awareness by inspecting in which Git repository you currently are. Concluding with the catchy name RET [1] for the application, creating a pull request is now as easy as executing this command:

```
ret pr create
```

```
2023-02-20 19:59:34,593 WARN  [io.qua.dep.dev.tes.JunitTestRunner] (Test runner thread) Unable to lo
ad annotation type kotlin.jvm.internal.SourceDebugExtension cannot determine if it is @Testable


--
All 170 tests are passing (0 skipped), 1 test was run in 195ms. Tests completed at 19:59:34 due to c
hanges to StackOverflowCommandTest.class.
Press [space] to restart, [e] to edit command line args (currently 'stackoverflow kotlin concatenate
 list'), [r] to re-run, [o] Toggle test output, [:] for the terminal, [h] for more options>
```

Image 2. The Quarkus development mode live reloads and runs relevant tests on code changes, and allows for invoking the PicoCLI application directly.

On top of that, any Alfred workflow we need for quick access to the automations can just call the CLI underneath. Do you want an IntelliJ plugin? You guessed it, just create a keyboard shortcut that calls the CLI underneath. Creating these integrations became a bliss, and using it is even better. Every time we use RET to create a PR, it feels so delightful to just press a few buttons and poof, it's there!

{ HOW WE BUILT IT }

Since we wanted to build a CLI, but we are also JVM programmers, we looked into the possibilities in this ecosystem. Quickly we found the combination of Quarkus, PicoCLI and GraalVM. First, we want to briefly introduce these technologies before we show how they work together.

Quarkus [2] is an open-source framework backed by RedHat which started to become big when Kubernetes' popularity was on a rise. In that time, there were no good ways to build Java applications that were fit for running on Kubernetes or serverless, i.e. with a quick startup time, low memory footprint and a low time to first request. Quarkus gets you started quickly with sensible defaults and they provide an interactive developer mode featuring hot code reload, continuous testing, and more.

PicoCLI [3] is a great library for creating CLI applications on the JVM. Register commands, options, flags, et cetera using either annotations or by dynamically calling methods on the API. Quarkus offers an extension which allows you to inject dependencies in the commands and directly invoke the commands from the developer mode.

GraalVM [4] allows us to compile our CLI tool into a native application. This ensures a super fast startup time and low memory footprint. Quarkus provides an integration with GraalVM, allowing you to build real applications into native images.

{ HOW IT WORKS TOGETHER }

Using those techniques, it was pretty easy to build a professional CLI tool out of it following the same code quality standards we always adhere to.

In the following snippet you can see how easy it is to create a basic subcommand for querying StackOverflow.

```
@Command(name = "stackoverflow")
class StackOverflowCommand(private val
browserUtils: BrowserUtils) : Runnable {

   @Parameters
   private val queryParts = emptyList<String>()

   override fun run() {
       val query = queryParts.joinToString(" ")
       val encodedQuery = URLEncoder.encode(query,
Charset.defaultCharset())
       val url = "https://www.stackoverflow.com/
search?q=$encodedQuery"
       browserUtils.openUrl(url)
   }
}
```

The `@Command` and `@Parameters` annotation are from PicoCLI, while Quarkus registers the command as a bean and powers the dependency injection for our `BrowserUtils` bean. As this is registered as a subcommand under `ret`, this allows us to invoke `ret stackoverflow kotlin concatenate list`.

By running `mvn quarkus:dev` we start the developer mode. It continuously compiles the code, re-runs the relevant tests automatically with each change, allows us to invoke our command line application using `e` and with space you re-run the last invoked command.

Finally, by enabling the Maven profile 'native' when invoking the package phase, a native image will be created using GraalVM. The profile is provided out of the box by Quarkus. (Image 2.)

{ USAGE }

For new users of RET, the setup process is now as easy as executing the following commands in succession:

```
# brew tap ret <our custom homebrew Git repo>
# brew install ret
# ret configure
```

This last command would guide you through the initial configurations of the application (i.e. by providing your email address, git repositories, access tokens and other personalized settings). After that, you're all set up to go!

We have users from many different teams across our organization, and are still growing daily. Currently, three tools have been integrated (Azure DevOps, Splunk, Signal FX), which will grow once we are finished with the plugin architecture we are developing. This will make it easier to add new features users have been requesting for.

{ STARTUP APPROACH }

Sounds cool, right? You might think: "How did they manage to get time in the sprint to work on this project?". We also never thought we would get time for this. However, we managed to do it by approaching our project as a startup.

In short, it went something like this. First, we had the idea and we developed a vision for the new engineering tools. We experimented and invested our 'unicorn' time. Unicorn is our dedicated time where we can work on anything related to innovation. Then, we started using the tools ourselves (eating our own dog food). We pitched the idea to our business manager and explained what the benefits could be. Since the organization regards developer happiness and efficiency highly, we all thought it was a great fit. She was enthusiastic and gave us time in the sprints to work on it.

In the weeks after, we developed the project further and invested into documentation and an easy installation process. We asked developers from other teams to beta test the tools and collected feedback. In the end, we released the product in the wild at an internal event. Currently we see about 100 ret calls a day, saving about an hour of lost time. And we have no plans for slowing down!

The whole process was really interesting. We had never done something like this before and are really surprised how much we have accomplished!

Our next steps are two-fold: integrate more tools and make it open-source. But before this can happen, we need to remove some company-specific code and ensure that other developers can easily create and contribute new functionality to RET. The plugin architecture helps to accommodate that. We are super stoked for making this work, because we really believe in open-source!

{ HOW TO GET STARTED }

Of course, this whole process took a lot of time and effort, so we would advise you to start small. You can already start today with the first steps of engineering your engineering tools. One of the key things is to share your tools with your colleagues and let them contribute. You can just create a new Git repository and put your existing tools there. Those can be things like scripts, command line aliases, Alfred workflows, Postman collections, but even bookmarks or common documentation can already be extremely useful to others.

When you let colleagues contribute, it ensures that they will improve the tools and that more people get a unified way of working. Also, they can share their own tools which you can then use. Try to find the biggest pain points you and your colleagues run into daily, and try solving those.

After the collection phase, you can start professionalizing. Ensure that each tool has some documentation, perhaps create some tests for it, if possible. Maybe generate a static site out of the docs and put that online somewhere. Host your tools in a Git repository so that they can be installed using Homebrew or Chocolatey. The possibilities are endless.

If you also want to pitch it to your organization, try to measure the usage and benefits. Imagine that you can tell your business manager that your product results in a reduction of five minutes a day for twenty developers. Furthermore, this also makes them happier and improves the engineering culture. It's hard for a business manager to not see value in that!

{ CONCLUSION }

Investing time in your engineering tools is not only fun, it can greatly benefit your company if every developer gets to share the pie. Collaborating on the scripts and shortcuts makes them easier to start being used and of better quality. Start easy by collecting what's already there and if you want to professionalize, pick a framework that suits your programming style and language preferences. Before you know it, you don't want to stop engineering your new engineering tools! <

{ REFERENCES }

1 Rabobank Engineering Tools
2 https://quarkus.io/
3 https://picocli.info/
4 https://www.graalvm.org/

Exciting news!

NLJUG has recently introduced three new membership tiers - Base, Core, and Key - giving members more options to choose from. NLJUG is a Java community dedicated to developing and making the platform accessible for knowledge sharing and networking. With over 4500 members and 80 partners, NLJUG is a professional platform that aims to remain inclusive for everyone, including students. However, the growing costs of events, no-shows and the production of the Java Magazine pose challenges. To address this, NLJUG is introducing the Core, and Key memberships, while keeping the regular (base) membership accessible and unchanged. If you don't want the extra benefits of the Core & Key memberships, as described below, nothing will change for you. But, if you don't want to miss out on these amazing benefits, then upgrade your membership today to unlock perks & discounts!

BECOME AN NLJUG MEMBER

STUDENT	NLJUG BASE	NLJUG CORE	NLJUG KEY
FREE	€ 59,50 *	€ 125,00 *	€ 279,00 *
Free entrance to all NLJUG events (subject to availability) Food and drinks included	Free access to J-Fall (subject to availability) Food and drinks included	Priority access to the J-Fall conference Food and drinks included	VIP access to the J-Fall conference Food and drinks included
Free Java Magazine subscription (4 digital issues a year)	Free Java Magazine subscription (4 issues a year)	Free Java Magazine subscription (4 issues a year)	Free Java Magazine subscription (4 issues a year)
NLJUG newsletter containing the most relevant knowledge and meetups	NLJUG newsletter containing the most relevant knowledge and meetups	NLJUG newsletter containing the most relevant knowledge and meetups	NLJUG newsletter containing the most relevant knowledge and meetups
	Paid access to the J-Spring conference	50% discounted access to the J-Spring conference	Free access to J-Spring conference (worth € 169,-**)
	Free access to NLJUG University sessions	Free access to NLJUG University sessions	Free access to NLJUG University sessions
		50% discounted acces to J-Fall Pre-con Masterclasses	Free access to J-Fall Pre-con Masterclasses (worth € 129,-**)
		Additional exclusive discounts on courses and training organized by our partners	Additional exclusive discounts on courses and training organized by our partners
			Access to J-Spring and J-Fall VIP Lounges*** incl. reserved free parking
			Exclusive access to the NLJUG Core Slack channel, where you can discuss the future of our community
			NLJUG Key Member Swag (including NLJUG Core Member Hoody, Core Member pin and much more)
			NLJUG Key Member visible on conference badges

* 9 % VAT included, billed anually
** 21 % VAT included
*** VIP area with food and drinks

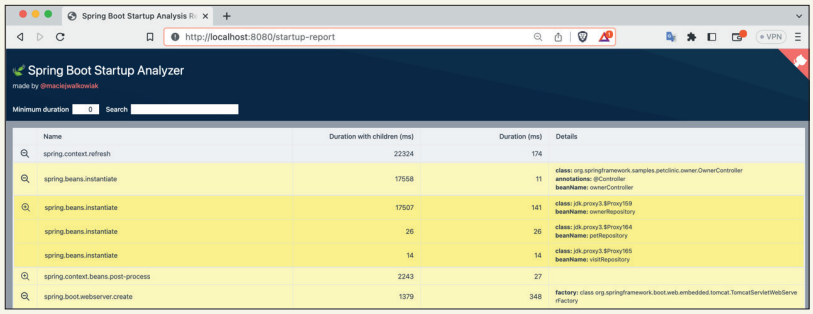
Advantages of becoming an NLJUG member:

- Free access to J-Fall
- Payed access to J-Spring
- Free access to the NLJUG University sessions
- 4 times a year receive the (printed) Java Magazine
- Discounts on partner events

HTTPS://NLJUG.ORG/LIDWORDEN/

.nl. jug

BYTE SIZE



{ SPRING BOOT STARTUP REPORT }

Let's face it, we've all seen this - more often than we like to admit. A beautiful Spring Boot app, but why on earth does it take so bloody long to Boot? What is it doing? The good news is: you'll get a nice visualisation of all that heavy lifting. Add Maciej Walkowiak's spring-boot-startup-report module as an optional dependency. Restart your app one more time and review the startup report. The bad news is: it will not fix the underlying cause of the slow startup... see https://bit.ly/sb-sr.

› **Maarten Mulders - @mthmulders**

{ GOTTA CATCH 'M ALL! }

Are you searching for a state of the art Android demo to learn the newest tech? Pokedex is a repository made by Jaewoong Eum which demonstrates modern Android development with Hilt, Material Motion, Coroutines, Flow, and Jetpack libraries (Room, ViewModel) based on MVVM architecture. And it's about Pokemons, so what else? **https://github.com/skydoves/Pokedex**

› **Julien Lengrand-Lambert**

{ MAVEN DAEMON }

An easy way to speed up a Maven build is by running it with the Maven Daemon (mvnd). Among other things, mvnd uses classloader caching, multiple long-living background processes (like Gradle) and a native executable to speed up your builds. I've seen performance gains of up to 40% on my projects. See **https://github.com/apache/maven-mvnd** for all the details!

› **Hanno Embregts - @hannotify**

{ MANAGING DOTFILES }

Many developers manage their user-specific application config (also known as dotfiles) in a Git repo. This allows for keeping track of changes and synchronizing the dotfiles across different machines. But dotfiles can live all over your home directory, so managing them in a single repo can be hard. To take the next step, you could use 1) a Git bare repo and 2) aliasing Git commands to send them to that repo. See **https://bit.ly/3nCjPBX** for more!

› **Dylan van Gils - @Dylan57401792**

HOW TO WRITE A BYTE SIZE' BYTE SIZE

Do you have a tip to share, an opinion, or simply want to share something short? That's what the byte size format is for! Byte sizes are short bits of information, in a Twitter format: you have roughly 280 characters, and space enough for a link, or a picture! Send us your byte sizes by @JavaMagazineNL on Twitter or by mailing **info@nljug.org**!

› **Java Magazine Editorial Committee**

FIRST8 | CONCLUSION

IN C<>DE WE TRUST

First8 Conclusion is specialist op het gebied van maatwerkapplicaties vanuit de Java & Open Source techniek.

WIL JIJ:

- werken in teams van passievolle Java specialisten?
- een gezonde balans tussen werk en privé?
- werken met en bijdragen aan open source?
- werken bij een intiem bedrijf met de faciliteiten van een groot bedrijf?
- jezelf blijven ontwikkelen?
- meewerken aan Masters of Java?
- genieten van bordspelavonden en pizzasessies?

Bekijk onze vacatures en ontdek wat First8 Conclusion voor jou kan betekenen.

Jonkerbosplein 52 – 16e verdieping
6534 AB Nijmegen

Edisonbaan 15
3439 MN Nieuwegein

www.conclusion.nl/first8

V

COLUMN

Write more, publish less

Maja Reißner *is*
Teacher cyber security.

Here's something new: advice from a stranger. Well, it's actually more advice to myself and a wish for the whole world.

So what's the reasoning behind this? I like writing so I can order and sharpen my thoughts and opinions. Although I haven't read any profound research about it, I'm sure it's not a personal thing but generic, because I've read this by lots of writers before. Therefore, I may easily state: writing is beneficial to ordering and sharpening thoughts and opinions. That covers the first part of the advice: "write more".

If you read this sentence on 'beneficial writing' carefully, it is a different statement than in the sentence before because it states a fact and not an opinion anymore. While it is fine to write it down for my personal notes, I shouldn't publish it without further explanation or context. Otherwise it is not clear to the reader that I very much believe in what I say but actually have no evidence, but merely personal experience and anecdotes.

At the same time, if I choose to make it clear that something is my own opinion it may appear to be my unique thought. I'm certain that although I cannot remember reading about the benefits of writing before having that opinion myself, it is very likely that I actually first read it and only (knowingly or unknowingly) agreed in hindsight.

Now, if I were publishing this in something very short like a tweet, it could be catchy like:
"Writing is beneficial to ordering and sharpening thoughts and opinions. Spring is in the air, go start blogging!"

Sounds nice. And I hate it. As is, this is written as a fact and it's not. I'm not saying it's wrong, but it's stated as a fact although I don't know if it's true. And how on earth would a reader know if it's true?

Of course I read more than I write and the amount of wrong facts I read on the internet really hurts me. I would so very much appreciate people spending more time on precise wording, doing proper research and adding references to their work. It's hard work and it costs a lot of time and energy to do research and to articulate things well. Still, ideally you write this text once and people will read and profit from it several times.

This doesn't mean you shouldn't write. We all should write. And only publish the clever bits.

Note 1: There are people who enjoy the diversity of thoughts and believe that there's beauty and benefit in hearing everyone. I disagree. I'm spammed by the sheer amount of information out there and I rather read one excellent, well-researched article or profound opinion every 4 months than 20 braingasms every day.

Note 2: I skipped the real reasons why writing is beneficial. It won't fit in this column. Stay tuned and check out the next magazine for more on writing.

Note 3: I've run this text through ChatGPT to improve its readability. The result was more readable. It was also very boring. I decided to keep the original.

VAN HET BESTUUR

Een update vanuit het NLJUG-bestuur met dit keer een vooruitblik naar een aantal events dit jaar.

Het conferentieseizoen is weer geopend! Bij het schrijven van deze bestuurscolumn sta ik op het punt om op het vliegtuig naar Atlanta te stappen voor de DevNexus-conferentie. DexNexus is natuurlijk geen J-Fall, maar het komt in de buurt ;-).

{ TEQNATION EN J-SPRING }

Gesproken over conferenties: de voorbereidingen voor de eerste events dit jaar vanuit de NLJUG zijn in volle gang! Op 17 mei is TEQnation in DeFabrique in Utrecht **(https://teqnation.com**) en een maand later is het tijd voor J-Spring, op 21 juni in de Jaarbeurs, ook in Utrecht (**https://jspring.nl**). De eerste sprekers voor J-Spring zijn al vastgelegd en ik kan alvast verklappen dat het weer een fantastische editie belooft te worden. Heb je nog geen kaartje? Bestel er dan snel één, want ze gaan hard.

{ NLJUG 20 JAAR }

Dit jaar is er een mijlpaal voor de NLJUG: we bestaan 20 jaar! Een greep uit het oorspronkelijke persbericht (maart 2003) over de oprichting: *"Nederland kent tot heden een kleine groep Java-gebruikers die zich verenigd hebben in "Dutch Melange". Deze organisatie staat volledig achter de NL-JUG en zal derhalve opgaan in de nieuwe Java User Group. Het oprichten van de NL-JUG komt voort uit een wereldwijd harmonisatie proces van Java user groups"*. Tijdens J-Spring en zeker tijdens J-Fall dit jaar, zullen we op meerdere manieren aandacht besteden aan dit bijzondere moment. Zorg dus dat je erbij bent!

{ NLJUG SPEAKER ACADEMY }

Ook dit jaar bieden we vanuit de NLJUG weer kosteloos onze speaker academy aan. In twee avonden word je door ervaren sprekers en leden van de J-Fall programmacommissie meegenomen in de wereld van 'spreken op conferenties'. We helpen je met het kiezen van onderwerpen, schrijven van proposals voor een call-for-papers, voorbereiden van presentaties en het oefenen van spreken voor een publiek. Uit dit programma zijn al veel goede sprekers gekomen en we zien geregeld deelnemers van de speaker academy terug in de top 10 best beoordeelde sessies van J-Fall! De eerste speaker academy sessie zal na J-Spring plaatsvinden. Houd **www.nljug.org** en je mailbox in de gaten voor meer informatie.

{ TOT SLOT }

We zijn altijd op zoek naar interessante artikelen voor Java magazine. Heb je een onderwerp waarvan je niet zeker weet of het interessant genoeg is? Of kun je hulp gebruiken bij het schrijven of reviewen van het artikel? Geen probleem, de redactiecommissie van Java magazine helpt je daarbij. Interesse? Neem even contact op via **info@nljug.org**.

En noteer alvast in je agenda: J-Fall is dit jaar op donderdag 9 november in Pathé Ede. Op 8 november is er weer een pre-conference met een aantal workshops, de Masters of Java en 's avonds een diner met alle deelnemers en sprekers. Zie je daar ;-)

Groeten,

Bert Jan Schrijver
(bestuurslid stichting NLJUG)

OOK DIT JAAR BIEDEN WE VANUIT DE NLJUG WEER KOSTELOOS ONZE SPEAKER ACADEMY AAN.

FIRST8 | CONCLUSION

MASTERS OF JAVA 2023

Funprogging contest voor alle Java Developers

De Masters of Java is een roemruchte “funprogging contest” (gebaseerd op Java SE), die toegankelijk is voor iedere Java ontwikkelaar. In deze wedstrijd wordt niet de API-kennis, maar de echte programmeer-vaardigheid getest. In voorgaande edities heeft Masters of Java al heel wat verhitte strijd opgeleverd!

Er zijn 5 zeer diverse programmeeropdrachten voor de teams. Deze moeten binnen de tijdslimiet worden opgelost. Voor elke seconde die je overhoudt, scoor je een punt. Ook kun je extra punten scoren met bepaalde testen. Het team met de meeste punten wint.

Een team bestaat uit maximaal 2 ontwikkelaars en het team dat aan het einde van de dag de meeste punten heeft, mag zich “Master of Java 2023” noemen. Er zal naast eeuwige roem natuurlijk gestreden worden voor mooie en spectaculaire prijzen. De afgelopen jaren is de beslissing steeds gevallen tijdens de laatste opdracht. Het belooft dus een zinderende wedstrijd te worden!

Schrijf nu in voor Masters of Java

EEN NLJUG EVENT

nl. jug

2023 MASTERS OF JAVA nl. jug

Wanneer: woensdag 8 november
Tijd: 13.00u tot 17.30u
Waar: Van der Valk Hotel, Veenendaal
Kosten: Gratis

Als trotse hoofdsponsor zorgt First8 Conclusion elk jaar voor 5 uitdagende opdrachten en een robuuste game server. Met veel enthousiasme, bevlogenheid en passie werken wij aan deze en andere gave projecten. Iets voor jou? We maken graag kennis met je.

Benieuwd naar eerdere opdrachten?
Check https://github.com/First8/mastersofjava

Eventbrite

www.first8.nl

Aanmelden: https://nljug.org/masters-of-java-2023/

CHILIT

VOOR DEVELOPERS MET PIT!

Ben jij een developer die méér brengt? Méér waard is? Méér pit heeft? Ontdek dan wat CHILIT voor jou kan betekenen.

Zo geniet jij straks van;

- ☑ een vast salaris,
- ☑ schaalbare bonussen,
- ☑ veel vrijheid,
- ☑ volop zekerheid!

www.chilit.nl

Chilit B.V.
Utrechtseweg 75, 3702AA Zeist

088 224 5480, hello@chilit.nl
Check onze beloningsstructuur op chilit.nl