jQAssistant

Link: https://jqassistant.org/

In eigenes Projekt Einbinden

```
<plugin>
  <groupId>com.buschmais.jqassistant</groupId>
  <artifactId>jqassistant-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>scan</goal>
        <goal>analyze</goal>
      </goals>
      <configuration>
        <warnOnSeverity>MINOR</warnOnSeverity>
        <failOnSeverity>MAJOR</failOnSeverity>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Verwendung

`mvn jqassistant:help`

Zeigt alle *goals*

`mvn jqassistant:scan`

Scannt das Target-Verzeichnis (`mvn verify` sollte schon ausgeführt worden sein)

`mvn jqassistant:server`

Startet den *Neo4j* Server

Neo4j Browser öffnen: http://localhost:7474/

Browser-UI

`<shift-enter>`	Multiline Editig
`<ctrl-enter>`	Ausführen

JQAssistant Abfragen:

Wie viele Klassen gibt es im Projekt?

```
MATCH (t:Class) return count(t)
```

Alphabetische Liste aller Klassen im Projekt:

```
MATCH (n:Class) RETURN n.fqn, n.fileName ORDER BY n.fqn ASC
```

Liste der Entwickler:

```
MATCH (n:Developer) return distinct n.name
```

Liste aller Enums:

```
MATCH (n:Enum) RETURN n.name, n.fqn ORDER BY n.name ASC
```

Welche Klassen verwenden den einen bestimmten Enum?

```
MATCH (c:Type)-[:DEPENDS_ON]->(n:Enum)
WHERE n.name = "Production"
RETURN c.fqn
```

Welche Enums werden nicht verwendet?

```
MATCH (n:Enum)
WHERE NOT (:Type)-[:DEPENDS_ON]->(n)
RETURN n.fqn
```

Welche Klassen implementieren welche Interfaces?

```
match (c:Class)-[:IMPLEMENTS]->(i:Interface) return c.fqn, i.fqn
```

Welche Interfaces werden durch keine Klasse implementiert?

```
match (i:Interface)
where not (:Class)-[:IMPLEMENTS]->(i:Interface)
return i.fqn
```

Welche Typen verwenden diese Interfaces?

```
match (t:Type)-[:DEPENDS_ON]->(i:Interface)
where not (:Type)-[:IMPLEMENTS]->(i)
return t.fqn, i.name
```

Welche Typen sind im Package xy?

```
MATCH (p:Package) -[:CONTAINS]->(c)
WHERE p.fqn = "org.slf4j.impl"
RETURN c.fqn
```

Was verwenden die Typen im Package xy?

```
MATCH (p:Package) -[:CONTAINS]->()-[:DEPENDS_ON]->(c)
WHERE p.fqn = "org.slf4j.event"
RETURN c.fqn
```

Wer verwendet die Typen im Package xy?

```
MATCH (c)-[:DEPENDS_ON]->(m)<-[:CONTAINS]-(p:Package)
WHERE p.fqn = "org.slf4j.event"
RETURN m.name, c.fqn
```

Als Knoten:

```
MATCH (c)-[:DEPENDS_ON]->(m)<-[:CONTAINS]-(p:Package)
WHERE p.fqn = "org.slf4j.event"
RETURN m, c
```

Alternative Syntax:

```
MATCH (c)-[:DEPENDS_ON]->(m), (p:Package)-[:CONTAINS]->(m)
WHERE p.fqn = "org.slf4j.event"
RETURN m.name, c.fqn
```

Welche Typen in Package xy werden nicht verwendet?

```
MATCH (p:Package)-[:CONTAINS]->(m)
WHERE p.fqn = "org.slf4j.event"
AND NOT ()-[:DEPENDS_ON]->(m)
RETURN m.name
```

Welche Typen werden nicht verwendet?

```
MATCH (t:Type) WHERE NOT ()-[:DEPENDS_ON]->(t) RETURN t.fqn
```

Welche Klassen verwenden *interne* Klassen?

```
match (t:Type)-[:DEPENDS_ON]->(c:Class)
where c.fqn CONTAINS 'internal'
return t.fqn
```

Welche Klassen deklarieren die meisten Methoden?

```
MATCH
  (t:Type)-[:DECLARES]->(m:Method)
RETURN
  t.fqn AS Type, count(t) AS DeclaredMethods
ORDER BY
  DeclaredMethods DESC
LIMIT 20
```

Welche Klassen deklarieren die meisten Felder?

```
MATCH
  (t:Type)-[:DECLARES]->(m:Field)
RETURN
  t.fqn AS Type, count(t) AS DeclaredFields
ORDER BY
  DeclaredFields DESC
LIMIT 20
```

Welche Klassen verwenden am häufigsten den gleichen Typ?

```
MATCH
  (t:Class)-[x:DEPENDS_ON]->(n:Type)
RETURN
  t.fqn, x.weight, n.fqn
ORDER BY
  x.weight desc
LIMIT 20
```

Welche zyklische Abhängigkeiten zwischen Klassen gibt es?

```
MATCH
  (t1:Type)-[:DEPENDS_ON]->(t2:Type),
  cycle=shortestPath((t2)-[:DEPENDS_ON*]->(t1))
RETURN
  t1.fqn as Type, nodes(cycle) as Cycle
```

Anreicherung der vorhandenen Daten

Alle Packages unter `src/main` mit dem Label `Component` markieren:

```
MATCH
    (:Main:Artifact)-[:CONTAINS]->(root:Package)-[:CONTAINS]->(component:Package)
WHERE
    root.fqn = "org.slf4j"
SET
    component:Component
RETURN
    component.fqn as Component
ORDER BY
    component.name desc
```

Gewichtete Beziehung `DEPENDS_ON_COMPONENT` zwischen `Component` Packages hinzufügen:

```
MATCH
    (c1:Component)-[:CONTAINS*]->(t1:Type),
    (c2:Component)-[:CONTAINS*]->(t2:Type),
    (t1)-[:DEPENDS_ON]->(t2)
WHERE
    c1 <> c2
WITH
    c1, c2, count(*) as weight
MERGE
    (c1)-[d:DEPENDS_ON_COMPONENT]->(c2)
SET
    d.weight = weight
RETURN
    c1.fqn as Component, c2.fqn as ComponentDependency, d.weight as Dependency
```

Zyklische Abhängigkeiten zwischen Packages:

```
MATCH
    (c1:Component)-[:DEFINES_COMPONENT_DEPENDENCY]->(c2:Component),
    cycle=shortestPath((c2)-[:DEFINES_COMPONENT_DEPENDENCY*]->(c1))
RETURN
    c1 as Component, nodes(cycle) as Cycle
```