Politechnika Krakowska
im. Tadeusza Kościuszki

Podejście projektowe Eclipse MicroProfile

Kamil Osika
Jakub Klimek
Adam Zych
Paweł Śliwa
Wiktor Sokół

Geneza

- Zwiększenie popularności aplikacji mikro-serwisowych
- Optymalizacja kodu pod kątem architektury mikro-serwisowej

Czym jest Eclipse MicroProfile?

- Jest to zestaw specyfikacji do tworzenia aplikacji w architekturze mikroserwisowej.
  - Część pochodzi z Java EE, a część została stworzona na potrzeby Eclipse MicroProfile.
- Zapewnia przenośność aplikacji na wielu instancjach MicroProfile.
- Konkurencją jest Spring

Specyfikacje zawarte w MicroProfile z Java EE

- CDI (Context and Dependency Injection)
- JAX-RS (Java API for RESTful Web Services)
- JSON-B (JSON Binding)
- JSON-P (JSON Processing)
- Common Annotations

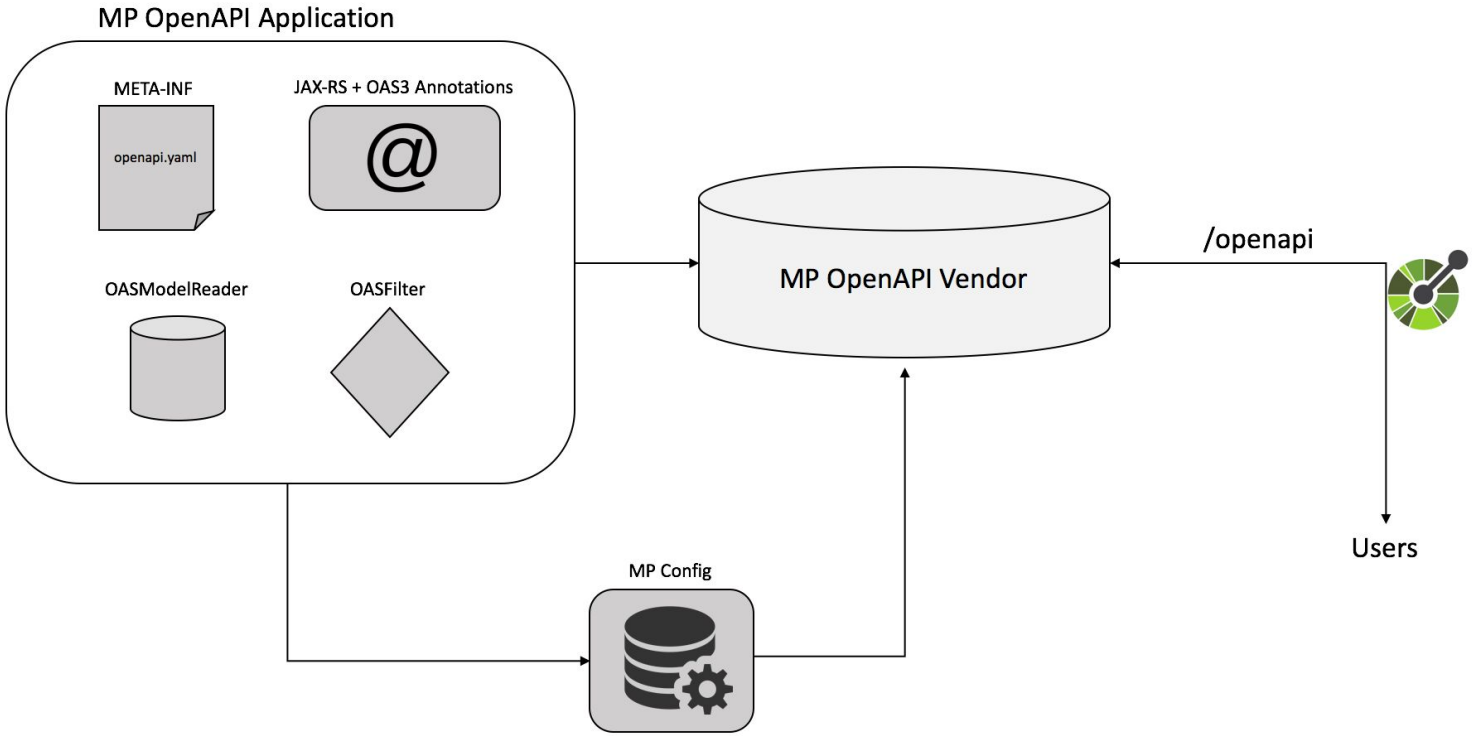
Poszczególne komponenty Eclipse MicroProfile

- Eclipse MicroProfile Config
- Eclipse MicroProfile Fault Tolerance
- Eclipse MicroProfile Health Check
- Eclipse MicroProfile JWT Authentication
- Eclipse MicroProfile Metrics
- Eclipse MicroProfile OpenAPI
- Eclipse MicroProfile OpenTracing
- Eclipse MicroProfile Rest Client

MicroProfile OpenAPI

Specyfikacja OpenAPI 1.0, ma na celu dostarczenie zestawu interfejsów Java oraz modeli programistycznych, które pozwolą programistom Java na natywne tworzenie dokumentów OpenAPI v3 z ich aplikacji JAX-RS.

Przegląd komponentów składających się na OpenAPI



Typowe adnotacje OpenAPI

@APIResponse - Opisuje pojedynczą odpowiedź z operacji API.

@APIResponseSchema - Wygodny skrótowy sposób określania prostej odpowiedzi za pomocą klasy Java, która w przeciwnym razie mogłaby zostać określona za pomocą @APIResponse.

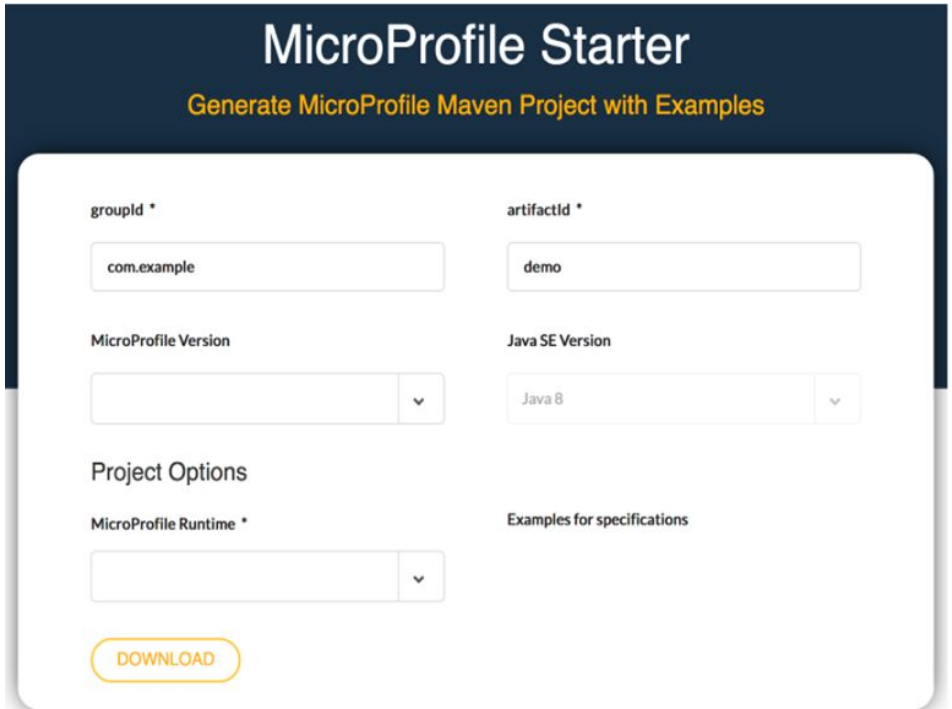
@Operation - Opisuje pojedynczą operację API na ścieżce.

@Parameter - Opisuje pojedynczy parametr operacji.

start.microprofile.io

- Generate MicroProfile projects
- Visual Studio Code/IntelliJ Idea plugin
- Command line tooling

```
<dependency>
  <groupId>org.eclipse.microprofile</groupId>
  <artifactId>microprofile</artifactId>
  <version>1.3</version>
  <type>pom</type>
  <scope>provided</scope>
</dependency>
```



Current MicroProfile implementations

Przykładowy kontroller

```
@Path("/api/books") // just a basic JAX-RS resource

@Counted // track the number of times this endpoint is invoked

@RequestScoped

public class BooksController {

 @Inject //use CDI to inject a service

 private BookService bookService;

 @GET

 @RolesAllowed("read-books")

 // uses common annotations to declare a role required

 public Books findAll() {

  return bookService.getAll();

 }

}
```

Przykładowy serwis

Duża konfigurowalność

```
@Inject

@ConfigProperty

private int maxBooks
```

```
@ApplicationScoped
public class BookService {
 @Inject
 // JPA is not provided out of the box, but most providers support it at
 // some level.  worst case, create your own producer for the field
 private EntityManager entityManager;
 @Inject
 // use configuration to control how much data you want to supply at
 // a given time
 @ConfigProperty(name = "max.books.per.page", defaultValue = "20")
 private int maxBooks;
 public Books getAll() {
  List < Book > bookList = entityManager
   .createQuery("select b from Book b", Book.class)
   .setMaxResults(maxBooks) // use that configuration to do a paginated look
   .getResultList();
  return new Books(bookList);
 }
}
```

Zabezpieczenia

```
@RolesAllowed("read-books")
```

```
@LoginConfig(authMethod = "MP-JWT", realmName = "admin-realm")
```

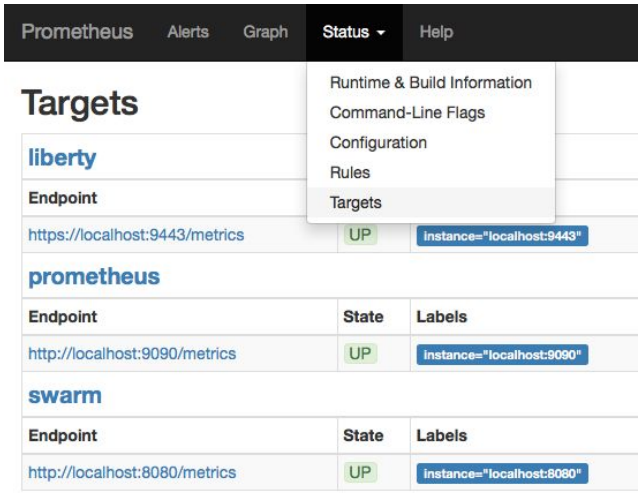
```
@Inject

private JsonWebToken jsonWebToken;
```

```
boolean createAny = jsonWebToken.getClaim("create.books.for.other.authors");
if (!createAny) {
    throw new NotAuthorizedException("Cannot create book, wrong author");
}
```

Monitorowanie

```
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>microprofile-metrics</artifactId>
</dependency>
```

By monitorować system z poziomu GUI wymagana jest instalacja Prometeusza




```
{
    "base": {
        "classloader.totalLoadedClass.count": 16987,
        "cpu.systemLoadAverage": 1.22,
        "thread.count": 141,
        "classloader.currentLoadedClass.count": 16986,
        "jvm.uptime": 52955,
        "memory.committedNonHeap": 131727360,
        "gc.PS MarkSweep.count": 3,
        "memory.committedHeap": 503316480,
        "thread.max.count": 143,
        "gc.PS Scavenge.count": 20,
        "cpu.availableProcessors": 8,
        "thread.daemon.count": 123,
        "classloader.totalUnloadedClass.count": 2,
        "memory.usedNonHeap": 117340624,
        "memory.maxHeap": 503316480,
        "memory.usedHeap": 139449848,
        "gc.PS MarkSweep.time": 428,
        "memory.maxNonHeap": -1,
        "gc.PS Scavenge.time": 220
    }
}
```

Obsługa błędów

@Retry

// Retry indicates that this should trigger retry the method call several times in case the remote server call results in an exception

@CircuitBreaker

// CircuiBreaker wraps the call in a circuit breaker which opens after several failures and closes again after some time

@Fallback(fallbackMethod = "getCachedAuthor")

// Fallback indicates that we should fall back to the local cache

// if the method fails even after several retries
// or the circuit is open

Ciekawostki

1) Czy w usłudze Fault Tolerance parametry adnotacji Fault Tolerance, takie jak Retry, są konfigurowalne?

2) Jakie są domyślne źródła konfiguracji w MicroProfile Config?

3) W jaki sposób można włączyć funkcję Open Tracing w MicroProfile dla wywołań JAX-RS?

Dziękujemy za uwagę!