



# MicroProfile a Quarkus

## Implementácia mikroslužby

Jaroslav Jankovič

# Zadanie

- Implementácia mikroslužby pre vytváranie poplatkov vo frameworku Quarkus
  - Úvod do Quarkus
    - Konfigurácia, nástroje, Developer Joy ...
  - Implementácia služby
    - Persistentná vrstva pomocou Hibernate (JPA)
    - Relačné databázové úložisko riešené ako embedded H2
    - Biznis vrstva pomocou CDI + JTA
    - REST vrstva pomocou JAX-RS
    - Použitie MicroProfile knižníc pre podporu mikroservisnej architektúry

# Príprava prostredia

- Príkazy môžete nájsť na <https://github.com/jjankovi/fmfi/blob/main/1.cvicenie.md>
- JDK 11+
  - <https://jdk.java.net/19/>
  - JAVA\_HOME environment property
- Maven 3.8.6 (Optional)
  - <https://maven.apache.org/download.cgi>
  - PATH environment property
- Vývojové prostredie:
  - VS Code (<https://code.visualstudio.com/>)
  - IntelliJ Idea (<https://www.jetbrains.com/idea/download/download-thanks.html?platform=windows&code=IIC>)

# Kostra aplikácie

- Na vytvorenie kostry použite <https://code.quarkus.io/>
- Parametre vyplňte nasledovne:
  - **Group:** sk.fmfi
  - **Artifact:** fee-service
  - **Build Tool:** Maven
  - **Extensions:**
    - RESTEasy Classic
- Stiahnite projekt ako zip a rozbaľte do svojho preferovaného adresára

# Prvé spustenie

- Spustite vývojové prostredie a otvorte projekt rozbalený zo zip archívu
- README.md
- Konfigurácia compiler tools (jdk, maven)
  - *mvn --version*
- Štruktúra projektu
  - pom.xml
  - src (java, test, resources)
  - Vygenerovaná REST služba, JUnit testy
- Spustenie developerského módu: <http://localhost:8080>



# Developer joy – Live Reload

- Otvorte hello REST API <http://localhost:8080/hello>
  - V službe **GreetingResource** zmeňte text a bez reštartu aplikácie skúste znovu zavolať službu
  - Spravte kompilačnú chybu a otestujte REST službu
  - Opravte kompilačnú chybu a vráťte hodnotu response na **Hello RESTEasy**
- Vytvorte novú REST službu AdvancedGreeting (bez vypnutia aplikácie)
  - Jedna GET metóda
  - Vystavená na URL **/greeting**, resp **/greeting?subject={subject}**
  - Nepovinný Query atribút **subject**
  - Vracať bude **plain text**
  - vráti text **Hello, World** prípadne **Hello, {subject}** (ak je parameter špecifikovaný)
  - Manuálne otestovanie

# Developer joy – Continuous testing

- V príkazovom riadku *press [r]*
- Otestovanie služby *hello*
  - Otvorte ***GreetingResource***, upravte hodnotu response metódy *hello* na *simple hello*
  - Spustite continuous testing
  - Upravte test ***GreetingResourceTest***, aby bol successfull
- Otestovanie služby *greeting*
  - Nová trieda ***AdvancedGreetingResourceTest***
  - Implementujte test ***testGreetingEndpointWithDefault***, ktorý pri volaní API nebude obsahovať nepovinný parameter
  - Implementujte test ***testGreetingEndpointWithParam***, ktorý pri volaní API bude obsahovať nepovinný parameter

## fee-service – model (1)

- Vytvorte adresárovú štruktúru ***src/main/java/sk/fmbi/model***
- Pridanie extensions:
  - *mvn quarkus:add-extension -Dextensions="hibernate-orm,hibernate-orm-panache,jdbc-h2"*
- Úprava application.properties:
  - *quarkus.datasource.db-kind=h2*
  - *quarkus.datasource.jdbc.url=jdbc:h2:mem:test*
  - *quarkus.hibernate-orm.database.generation=drop-and-create*
- Pridajte do projektu knižnicu *Lombok*:
  - **groupId:** org.projectlombok, **artifactId:** lombok, **version:** 1.18.24, **scope:** provided
- Aktualizujte si Maven závislosti



## fee-service – model (2)

- Vytvorte java triedu **Fee** do model package
- Anotujte triedu ako JPA entitu ( *@Entity*)
- Pridajte Lombok anotácie, **@Getter**, **@Setter**, **@NoArgsConstructor**
- Primárny kľúč **id** typu **Long** s anotáciami **@Id** a **@GeneratedValue**
- Pridajte ďalšie atribúty (použite anotáciu **@Column**):
  - **transactionId** typu String
  - **acno** typu String
  - **amount** typu BigDecimal
  - **postingDate** typu LocalDateTime

## fee-service – repository

- Vytvorte adresárovú štruktúru ***src/main/java/sk/fmbi/repository***
- Vytvorte CDI triedu ***FeeRepository*** s aplikačným scope ( *@ApplicationScope*)
  - Trieda bude implementovať rozhranie ***PanacheRepository<Fee>***
- Pridajte custom repository metódu ***listForAcno***
  - Jeden parameter ***acno***
  - Metóda bude vracať ***List<Fee>***
  - V implementácii použite parent metódu ***list***, ako query parameter použite ***acno***

## fee-service – backend interface

- Vytvorte adresárovú štruktúru *src/main/java/sk/fmbi/service*
- Vytvorte rozhranie **FeeService**
- Pridajte predpis metódy **createFee**
  - Parametre **String transactionId**, **String acno**, **BigDecimal transactionAmount**
  - Navráťová hodnota je novovytvorená entita **Fee**
- Pridajte predpis metódy **getAllFees**
  - Bez parametrov, vracajúca **List<Fee>**
- Pridajte predpis metódy **getFeesForAcno**
  - Parameter **String acno**
  - Navráťová hodnota je **List<Fee>**

## fee-service – backend bean (1)

- Vytvorte CDI beanu **FeeServiceBean** typu **@RequestScope**
  - Rovnaký priečinok ako rozhranie **FeeService**
  - Implementuje rozhranie **FeeService**
- Pridajte privátnu final premennú **feeRepository**
- Pridajte konštruktor
  - S anotáciou **Inject**
  - S inicializáciou premennej
- Implementujte metódy **getAllFees**, **getFeesForAcno**
  - Pomocou anotácie **@Transactional** označte metódy ako **netransakčné**
  - Pre načítanie všetkých poplatkov použite repository metódu **findAll + list**

## fee-service – backend bean (2)

- Pridajte podporu logovania do triedy **FeeServiceBean**
  - Použite *java.util.logging.Logger*
- Implementujte metódu **createFee**
  - Označte metódu ako **transakčnú**
  - Pridajte zmysluplnú logovaciu správu na začiatku metódy (názov operácie, parametre)
  - Hodnota poplatku – 0.01 €, resp 2 € v prípade transakcie väčšej ako 10000 €
  - Persistujte novú entitu pomocou repository a vráťte z metódy ako návratovú hodnotu



## fee-service – REST

- Vytvorte adresárovú štruktúru ***src/main/java/sk/fmbi/resource/dto***
- Vytvorte DTO objekt **FeeDTO** s atribútmi transactionId, acno, amount
- Vytvorte REST službu **FeeResource**:
  - Base URL služby bude **fee** (použité anotáciu **@Path**)
  - CDI závislosť na **FeeService** (cez konštruktor ako v CDI FeeServiceBean)
- Vytvorte metódu **GET** pre načítanie poplatkov
  - Response type **JSON**
  - nepovinný Query parameter **acno** riadi, aká biznis metóda sa použije
- Vytvorte metódu **POST** pre vytvorenie nového poplatku
  - DTO objekt ako parameter
  - Typ **JSON** ako request aj response type

## fee-service – JUnit test

- Pridanie mock knižnice
  - **groupId:** io.quarkus, **artifactId:** quarkus-junit5-mockito, **scope:** test
- Vytvorenie testovacej triedy ***FeeServiceTest***
  - Cesta *src/test/java/sk/fmfi/feeservice/service*
  - Anotácia ***@QuarkusTest***
- Pridanie závislostí
  - CDI závislosť na ***FeeService***
  - Mock závislosť (***@InjectMock***) na ***FeeRepository***
- Vytvorte 2 testy pre otestovanie poplatkov s rôznymi výškami

# fee-service – Integračný test

- Vytvorenie testovacej triedy ***FeeResourceIT***
  - Cesta *src/test/java/sk/fmfi/resource*
  - Anotácia ***@QuarkusTest***
- Vytvorte test **testCreateAndFetchFee** (inšpirujte sa ***GreetingResourceTest***)
  - V prvej fáze otestujte vytvorenie poplatku (použite metódu **body**)
  - Assert na HTTP response kód 200
- Pridajte správny header **Content-Type**
- Pridanie extension pre JSON serializáciu
  - *mvn quarkus:add-extension -Dextensions="resteasy-jackson"*
- Pridajte zavolanie GET metódy a otestujte odpoveď
  - HTTP response kód
  - HTTP response body (použite **body**, resp **body.prettyPrint**)

# fee-service – Config

- Do **FeeServiceBean** pridajte premennú **minimalFeeLimit**
  - Typ premennej bude **int**
  - Premenná bude obsahovať anotáciu **@ConfigProperty** s názvom **minimal.fee.limit**
- Upravte službu **createFee**
  - Použite premennú **minimalFeeLimit** na odlíšenie väčšieho a menšieho poplatku
- Do aplikačnej konfigurácie pridajte custom property
  - **minimal.fee.limit=\${minimal\_fee\_limit}**
  - Spustite aplikáciu
- Do developerského profilu aplikačnej konfigurácie (použite prefix **%dev**) pridajte custom property s hodnotou 10000 a spustite aplikáciu
- Spustite test **FeeServiceTest** – aký je výsledok testu?
  - Do testovacej verzie (použite prefix **%test**) konfigurácie pridajte chýbajúcu property a zopakujte test

# fee-service – OpenApi

- Pridanie extensions:
  - `mvn quarkus:add-extension -Dextensions="quarkus-smallrye-openapi"`
- Zapnite Swagger UI aj pre produkčný mód
  - `quarkus.swagger-ui.always-include=true`
- Otvorte Swagger UI cez DEV UI
  - url swagger: <http://localhost:8080/q/swagger-ui/>
  - vytvorte nový poplatok
  - vyhľadajte všetky poplatky
  - vyhľadajte poplatky pre zadaný účet
- Pomocou konfigurácie ***quarkus.smallrye-openapi***. \* upravte definíciu služby ***FeeResource***:
  - info-title, info-description, info-version
- Pomocou ***org.eclipse.microprofile.openapi.annotations***. \* upravte metódu ***createFee***:
  - @Operation (summary, description)
  - @RequestBody (description)



# fee-service – Security

- Pridanie extensions:
  - `mvn quarkus:add-extension -Dextensions="elytron-security-properties-file"`
- Upravte **FeeResource** tak, aby metódu pre načítanie poplatkov mohol volať len používateľ s rolou *user* (použite anotáciu *RolesAllowed*)
- Spustite aplikáciu a otestujte načítanie poplatkov, aký je výsledok?
- Do aplikačnej konfigurácie pridajte nového užívateľa s potrebnou rolou:
  - Nové aplikačné parametre budú mať prefix: `quarkus.security.users.embedded`
  - Použite nasledujúce premenné
    - `.enabled = true`
    - `.plain-text = true`
    - `.users.[user name] = [user_password]`
    - `.roles.[user name] = user`
- Upravte **FeeResource** tak, aby metódu pre vytvorenie poplatku mohol volať len používateľ s rolou *admin*. Otestujte danú metódu s užívateľom s rolou *user*
- Na vyriešenie problému upravte jednoriadkovou zmenou aplikačnú konfiguráciu

# fee-service – Health

- Pridanie extensions:
  - `mvn quarkus:add-extension -Dextensions="smallrye-health"`
- Spustite aplikáciu v developerskom móde
  - Skontrolujte výsledky jednotlivých health check procedúr: `/q/health/[ live / ready / started ]`
- Pridajte triedu **SimpleHealthCheck** pre otestovanie liveness
  - Package `sk.fmfi.health`
  - Implementujúcu rozhranie **HealthCheck**
  - Pridajte custom health check pomocou triedy **HealthCheckResponse** s negatívnym výsledkom
  - Skontrolujte health check aplikácie – aký je výsledok?
  - Pridajte anotáciu **Liveness**

# fee-service – Metrics

- Pridanie extensions:
  - `mvn quarkus:add-extension -Dextensions="smallrye-metrics"`
- Spustite aplikáciu v developerskom móde
  - Skontrolujte výsledky jednotlivých metrics rozhraní cez Dev UI
- Pridajte application metrics do triedy ***FeeResource***
  - Metóda **getFees** bude meraná pomocou `@Timed`
    - Nastavte atribút `absolute=true, unit=MetricUnits.MILLISECONDS, name, description`
  - Metóda **createFees** bude meraná pomocou `@Counted`
    - Nastavte atribút `absolute=true, name, description`
- Skontrolujte existenciu aplikačných metrík