УДК 004.42

Сабадош Володимир,
аспірант кафедри ІУСТ,
ДВНЗ «Ужгородський національний університет»
Коцовський Владислав,
доцент кафедри ІУСТ, к. т. н., доцент,
ДВНЗ «Ужгородський національний університет»
Ужгород, Україна

ВІД YAML ДО ГОТОВОГО МІКРОСЕРВІСУ: ІНСТРУМЕНТ ГЕНЕРАЦІЇ ШАБЛОННОГО КОДУ З ПІДТРИМКОЮ ГЕКСАГОНАЛЬНОЇ АРХІТЕКТУРИ

Мікросервісна архітектура [1] набула широкої популярності у корпоративному середовищі завдяки своїй здатності масштабувати окремі частини системи незалежно одна від одної, підвищити гнучкість розробки, полегшити розгортання та підтримку. Згідно зі статистикою, у багатьох відомих корпоративних проєктах (наприклад, Netflix, Amazon, Spotify) [2] застосунки складаються з десятків або навіть сотень мікросервісів.

Для того щоб пришвидшити розробку і підтримку таких систем, доцільно використовувати інструменти, які автоматично створюють типову структуру мікросервісів із дотриманням єдиної архітектури та розмежуванням відповідальностей між модулями. Ці інструменти дозволяють зменшити витрати на старт проєкту, забезпечити узгодженість архітектурних підходів і підвищити якість результату.

У відповідь на цю потребу з'явилася низка генераторів, зокрема Spring Initializr, OpenAPI Generator, AsyncAPI Generator та JHipster. Вони полегшують створення REST API, Kafka-слухачів, DTO та конфігураційних компонентів. Однак жоден із них не забезпечує повноцінного створення проєктів згідно з принципами Гексагональної архітектури [3], що включає ізоляцію доменної логіки, чітке

розмежування портів та адаптерів. Це призводить до потреби в подальшому ручному рефакторингу або адаптації коду під архітектурні стандарти команди.

Тому розробка інструмента, який здатен генерувати готову до розробки структури мікросервісу з підтримкою Гексагональної архітектури, є актуальним і практично значущим завданням, що сприяє автоматизації проєктування, підвищенню якості коду та відповідності принципам Domain-Driven Design (DDD) [4].

Design First підхід [5] стає дедалі популярнішим у розробці сучасних мікросервісів. Він передбачає, що специфікація API створюється на етапі проєктування ще до написання бізнес-логіки або серверного коду. Це дозволяє узгодити очікування між командами backend, frontend, QA та DevOps, забезпечуючи прозору комунікацію через єдине джерело правди — YAML-опис інтерфейсів.

Найпоширенішими форматами таких специфікацій є OpenAPI [6] (для REST API) та AsyncAPI [7] (для подієво-орієнтованих систем, зокрема Kafka). Ці формати підтримуються великою кількістю інструментів генерації серверного та клієнтського коду, документації, мок-серверів тощо. Приклад зазначених специфікацій проілюстрований на рис. 1.

```yaml
asyncapi: 2.0.0
info:
  title: User Created Events
  version: '1.0.0'
channels:
  user/created:
    subscribe:
      operationId: handleUserCreated
      message:
        payload:
          $ref: '#/components/schemas/UserCreated'
```

```yaml
openapi: 3.0.1
info:
  title: Users API
  version: 1.0.0
paths:
  /users:
    get:
      summary: Отримати користувачів
      responses:
        '200':
          description: Список користувачів
```

Рисунок 1.

Приклади специфікацій: OpenAPI (ліворуч) та AsyncAPI (праворуч).

Розроблений генератор реалізований на Java та використовує шаблони для OpenAPI Generator і AsyncAPI Generator. На вхід подаються:

- файл OpenAPI (`openapi.yaml`),
- файл AsyncAPI (`asyncapi.yaml`),
- назва майбутнього мікросервісу.

На основі цих даних формується повноцінний проєкт на Java з використанням фреймворку Spring Boot рис.2. Генератор обробляє вхідні специфікації та створює такі модулі:

- REST API (адаптери in)
- Kafka Listener (адаптери in)
- DTO-класи
- Сервісний шар (application services)
- Конфігураційні компоненти
- Заготовки портів (інтерфейсів) для зовнішніх залежностей (адаптери out).

Усі модулі автоматично структуруються у відповідності до принципів гексагональної архітектури.

```
src/
└── main/
    ├── java/
    │   └── com/example/user/
    │       ├── domain/          // бізнес-логіка
    │       ├── application/     // сервіси
    │       ├── adapters/
    │       │   ├── in/          // REST, Kafka вхідні адаптери
    │       │   └── out/         // Бази даних, API клієнти
    │       └── config/          // Spring конфігурації
    └── resources/
        ├── openapi.yaml
        └── asyncapi.yaml
```

Рисунок 2.

Структура проєкту з дотримання Гексагональної архітектури

Для генерації REST API інструмент використовується OpenAPI генератор: openapi-generator-cli generate -i openapi.yaml -g spring -o generated-rest -c config.json.

Для асинхронних API (Kafka) застосовується AsyncAPI Generator: ag asyncapi.yaml @asyncapi/java-spring-template -o generated-kafka.

Розроблений інструмент дозволяє зменшити час старту нового мікросервісу, підвищити узгодженість структури проєктів, а також сприяти впровадженню

підходу API-First і Test-Driven Development. Генерація шаблону з підтримкою гексагональної архітектури дозволяє зосередитися на доменній логіці та швидко впроваджувати нові рішення у production-середовище.

Список використаних джерел та літератури

1. Richardson C. Microservices Patterns. Manning Publications, 2018. 520 p.
2. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2021. 616 p.
3. Cockburn A. Hexagonal Architecture [Електронний ресурс]. – Режим доступу: https://alistair.cockburn.us/hexagonal-architecture/
4. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. 560 p.
5. Zoref E. Swagger Design First Approach [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@eyalzoref_26637/swagger-design-first-approach-in-microservices-development-with-spring-boot-maven-and-swagger-eb8525cb55f2
6. OpenAPI Specification [Електронний ресурс]. – OpenAPI Initiative. – Режим доступу: https://swagger.io/specification/
7. AsyncAPI Specification [Електронний ресурс]. – AsyncAPI Initiative. – Режим доступу: https://www.asyncapi.com/