ЛАТУША А.В.,
КАТЄЛЬНІКОВ Д.І.
Вінницький національний технічний університет

KOTLIN І KTOR У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ: ПЕРЕВАГИ ТА ПРАКТИЧНІ АСПЕКТИ

Анотація: Розглянуто особливості використання Kotlin та Ktor у мікросервісній архітектурі. Представлено аналіз переваг та практичних аспектів впровадження Ktor у мікросервісні системи, зокрема, з акцентом на асинхронне програмування, продуктивність та зручність Kotlin.

Ключові слова: Kotlin, Ktor, мікросервісна архітектура, асинхронне програмування, продуктивність

Abstract: The article examines the features of using Kotlin and Ktor in microservice architecture. The analysis covers the advantages and practical aspects of implementing Ktor in microservice systems, with a particular focus on asynchronous programming, performance, and the convenience of Kotlin.

Keywords: Kotlin, Ktor, microservice architecture, asynchronous programming, performance

Зі збільшенням обсягів даних та складності обчислень, мікросервісна архітектура стала одним із провідних підходів у розробці сучасних веб-додатків, орієнтованих на масштабованість та розподілену обробку. Kotlin, як сучасна мова програмування, розширює можливості побудови таких систем завдяки поєднанню продуктивності та зручності для розробників [1]. Зокрема, фреймворк Ktor забезпечує потужний інструментарій для розробки асинхронних мікросервісів, що є надзвичайно важливим у контексті високих навантажень [2]. Метою роботи є визначення основних переваг використання Kotlin та Ktor у побудові мікросервісної архітектури, а також дослідження викликів, з якими стикаються розробники при інтеграції цих технологій.

Попри переваги Kotlin та Ktor для побудови мікросервісів, їх інтеграція в масштабовані розподілені системи досі має ряд невирішених аспектів. Проблема полягає в тому, як забезпечити ефективне виконання асинхронних операцій, організувати взаємодію сервісів та оптимізувати використання ресурсів під час обробки великих обсягів даних у реальному часі. Сучасні дослідження зосереджуються на перевагах мікросервісної архітектури, таких як легкість обслуговування, масштабованість та незалежність компонентів [3]. Використання Kotlin забезпечує створення продуктивного та стійкого коду для мікросервісів. Ktor, як асинхронний фреймворк, представлений як ефективний інструмент для побудови веб-додатків, однак комплексний аналіз саме його застосування у складних мікросервісних архітектурах потребує подальшого вивчення [2].

Розробка мікросервісів на Kotlin дозволяє створювати продуктивні рішення завдяки його синтаксичній зручності та підтримці асинхронного програмування через корутини [4,5]. Асинхронні операції є ключовим аспектом у побудові масштабованих рішень, оскільки дають змогу обробляти кілька запитів одночасно без блокування основного потоку. Ktor, як легкий та гнучкий фреймворк для веб-розробки на Kotlin, пропонує широкі можливості для реалізації асинхронних мікросервісів завдяки корутинам, які суттєво знижують навантаження на сервер. Крім того, Ktor надає інструменти для роботи з HTTP, WebSocket та іншими протоколами, що значно спрощує інтеграцію мікросервісів між собою та взаємодію з іншими компонентами системи [2].

Переваги Kotlin та Ktor у мікросервісній архітектурі включають високу продуктивність, легкість у супроводженні, масштабованість та інтуїтивно зрозумілу структуру коду. Проте серед викликів варто зазначити обмеження щодо взаємодії з іншими мовами та платформами, а також необхідність ретельного підходу до налаштування асинхронності для уникнення накладних витрат [3].

Отже, використання Kotlin та Ktor у мікросервісній архітектурі має низку значних переваг, включаючи продуктивність, гнучкість у налаштуванні та підтримку асинхронного програмування. Подальші дослідження можуть бути зосереджені на оптимізації інтеграцій між сервісами, що дозволить знизити накладні витрати та забезпечити стабільну роботу великих систем. Перспективи подальших досліджень включають аналіз продуктивності Ktor у різних конфігураціях та порівняння з іншими фреймворками для побудови мікросервісів.

Список використаних джерел
1. Kotlin Coroutines Guide. Kotlin Documentation. URL: https://kotlinlang.org/docs/coroutines-overview.html (Last accessed: 10.11.2024).
2. Ktor: Build Asynchronous Servers and Clients in Kotlin. URL: https://ktor.io/ (Last accessed: 10.11.2024).
3. Newman, S. Building Microservices: Designing Fine-Grained Systems. 1st Edition. Sebastopol, CA, USA : O'Reilly Media, 2015. 278 p.
4. Josh Skeen, David Greenhalgh. Kotlin Programming. Atlanta, GA, USA : Big Nerd Ranch Guides, 2018. 480 p.
5. Pierre-Olivier Laurence, Amanda Hinchman-Dominguez, Mike Dunn, G. Blake Meike. Programming Android with Kotlin: Achieving Structured Concurrency with Coroutines 1st Edition. Sebastopol, CA, USA : O'Reilly Media, 2021. 355 p.

ЛІЩИНСЬКА Л.Б.,
ВНТУ

ОСНОВНІ ПІДХОДИ ДО ПОБУДОВИ СИСТЕМИ АДАПТИВНОГО ТЕСТУВАННЯ ЗНАНЬ

Анотація. Проаналізовано основні підходи для побудови системи адаптивного тестування знань.
Ключові слова: адаптивне тестування, машинне навчання, класифікація.

Традиційне тестування у вигляді стандартизованих тестів фіксованою довжин сьогодні, переростає у нові ефективні форми адаптивного тестування, що базується на відмінних від традиційних теоретико-методологічних підходах [1]. У зв'язку з цим, розробка методів і програмних засобів для систем адаптивного тестування знань є актуальними задачами різноманітних сфер діяльності, які потрібують об'єктивної та ефективної оцінки знань.

В загальному алгоритм адаптивного тестування знань складається з таких кроків:

1. З набору завдань вибирається найбільш підходяще (за певними параметрами) для користувача завдання.
2. Користувач вирішує завдання правильно чи неправильно.
3. Оцінка користувача оновлюється на підставі цієї відповіді.
4. Дані кроки повторюються до тих пір, поки згідно певна умова виконується. Така умова називається критерієм зупинки тестування. Як тільки вона задовольняється тестування вважається завершеним.

Оскільки на початку процесу тестування системі невідомо про рівень знань користувача поки він не відповість щонайменше на перше запитання, тестування починається з середнього рівня складності, вважаючи рівень підготовки користувача як «середній». Наступне завдання системи – якомога швидше пристосуватись (адаптуватись) до рівня користувача, для найефективнішої оцінки його знань.

В загальному випадку для розробки системи адаптивного тестування знань необхідні такі компоненти.

Набір тестових завдань, які відкалібровані за складністю. Банк завдань повинен калібруватись відповідно до певної психометричної моделі.

Точка входу у тест. В основному всі системи адаптивного тестування знань припускають, що кожен користувач, який починає тестування має «середній» рівень знань, але якщо користувач використовує систему повторно, є можливість починати тестування з іншого рівня.

Логіка вибору завдання з тестового набору. Кожна система тестування знань імплементує власну логіку для найбільш точного підбору завдань під рівень знань користувача.

Алгоритм підрахунку результатів. Після кожного вирішеного завдання, не важливо чи успішно, оцінка користувача змінюється в ту, чи іншу сторону. В результаті проходження N завдань система складає результуючу оцінку, яка, за потреби, може бути приведена до будь-якої системи оцінювання [2].

Критерій визначення завершення тестування. Система може пропонувати користувачеві завдання до тих пір, доки вона не зможе адекватно оцінити рівень його знань. Саме момент, коли оцінка стає «адекватною» і є таким критерієм. В кожні системі тестування даний критеріїв може кардинально відрізнятись. В класичному тестуванні, зазвичай, процес тестовая зупиняється коли