```
recipe = [{‘name’: ‘Молоко’, ‘quantity’: 200}, {‘name’: ‘Цукор’, ‘quantity’: 50}]
print(f"Калорійність страви: {calculate_calories(recipe)} ккал")
```

Результат програми: *Калорійність страви: 277.5 ккал*

Висновки. Розробка програми для кухарів на Python демонструє значний потенціал для автоматизації кулінарних процесів. Використання бібліотек pandas, matplotlib та інших дає змогу створити функціональний інструмент, що задовольнить потреби як професійних кухарів, так і любителів. Програма зменшує час, витрачений на планування страв, сприяє дотриманню здорового харчування і полегшує управління ресурсами на кухні [4].

Список використаних джерел

1. Lutz M. Learning Python. O’Reilly Media. 2013. 1643 p.
2. VanderPlas J. Python Data Science Handbook. O’Reilly Media, 2016. 548 p.
3. Taibi D., Lenarduzzi V., Pahl C. Processes, Motivations, and Issues for Migrating to Microservices Architectures. *IEEE Software*. 2017. P. 22–32.
4. Документація Python. URL: https://docs.python.org. (дата звернення: 01.12.2024).

УДК 004.415:004.432.4

Ілик В. В., здобувач вищої освіти,
Сеник І. О., асистент кафедри
інформаційних технологій

АРХІТЕКТУРА МІКРОСЕРВІСІВ ДЛЯ МАСШТАБНИХ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ

Донецький національний університет імені Василя Стуса, м. Вінниця

В епоху цифрової трансформації компанії потребують архітектурних рішень, які забезпечують швидке реагування на зміни, гнучкість у розвитку та високу масштабованість. Мікросервісна архітектура стає все популярнішою через можливість адаптації до змінного навантаження та здатність задовольняти вимоги високонавантажених систем.

Останнім часом мікросервіси почали широко застосовуватися у великих компаніях (Netflix, Amazon, Uber) для розробки високонадійних, продуктивних систем.

Порівняння з традиційним монолітним підходом, де додаток розробляється як єдине ціле, ускладнює гнучкість та масштабування.

Мета роботи – проаналізувати основні переваги та виклики впровадження мікросервісної архітектури у клієнт-серверні системи, виявити умови, за яких вона є найефективнішою;

1) дослідити ключові принципи мікро сервісної архітектури;

2) визначити, як мікро сервіси можуть забезпечити масштабованість та продуктивність;

3) розглянути виклики, з якими стикаються компанії, що впроваджують мікросервіси [1].

Основи мікросервісної архітектури

Мікросервіси – це підхід до розробки програмного забезпечення, в якому додаток розбивається на набір невеликих незалежних сервісів, кожен з яких виконує окремі функції.

Основні принципи:

- Кожен сервіс має чітко визначену функціональність.
- Сервіси взаємодіють через стандартизовані API.
- Кожен мікросервіс незалежно розгортається, тестується та оновлюється.
- Порівняння з монолітною архітектурою: Монолітний підхід об’єднує всі функціональні частини програми в одне ціле, що робить такі системи складними для масштабування, підтримки та оновлення.

Переваги мікросервісної архітектури

Масштабованість:

- Завдяки розділенню додатка на окремі сервіси можна масштабувати тільки ті компоненти, які потребують ресурсів, що забезпечує ефективне використання ресурсів.
- Підтримка горизонтального масштабування кожного сервісу на окремих серверах або контейнерах.

Гнучкість:

- Дає змогу використовувати різні технології, фреймворки, та мови програмування для окремих сервісів, залежно від вимог.

Незалежність компонентів:

- Розгортання та оновлення окремих сервісів відбувається незалежно, що знижує час простою додатка і підвищує швидкість оновлень.

Продуктивність команд:

- Підхід підвищує продуктивність розробницьких команд, які можуть незалежно працювати над окремими мікросервісами, зменшуючи залежності між командами та час узгоджень [2].

Виклики мікро сервісної архітектури

Складність комунікації:

- Мікросервіси вимагають чіткої комунікації між сервісами через API, що призводить до підвищення складності розробки.
- Потрібне використання протоколів комунікації, як-от HTTP / REST або gRPC, для передачі даних.

Моніторинг і підтримка:

- Більша кількість сервісів означає складнішу систему моніторингу та управління. Необхідне впровадження інструментів, як-от Prometheus, Grafana для моніторингу стану кожного сервісу.
- Забезпечення стабільності та швидке виявлення проблем, особливо в умовах постійного оновлення.

Управління даними:

- Розподілені сервіси мають окремі бази даних, що ускладнює синхронізацію даних між ними. Потрібне впровадження механізмів для забезпечення цілісності та консистентності даних.

- Розробка системи управління транзакціями та роботи з даними між мікросервісами.

Безпека:

- Кожен сервіс повинен бути захищений, оскільки множинні точки доступу до API збільшують вразливість системи.
- Впровадження аетентифікації та авторизації для кожного сервісу, наприклад, через OpenID Connect або OAuth2.

Приклади використання мікросервісної архітектури в клієнт-серверних системах

Netflix:

Завдяки мікросервісній архітектурі Netflix змогла досягти високої масштабованості та стабільності, обробляючи мільярди запитів щодня.

Реалізація незалежних сервісів для кожної функції (рекомендації, пошук, авторизація тощо) дала змогу швидко оновлювати та масштабувати окремі компоненти.

Uber:

Система Uber використовує мікросервіси для забезпечення швидкого обслуговування замовлень та координації водіїв і пасажирів.

Архітектура знижує затримки та допомагає обробляти великий потік запитів у реальному часі.

Amazon:

Впровадження мікросервісів допомогло Amazon забезпечити стабільну роботу інтернет-магазину та надає можливість команді розробників оновлювати окремі сервіси, не впливаючи на загальну роботу системи [3].

Мікросервісна архітектура пропонує ефективний підхід до розробки масштабованих клієнт-серверних систем, що дає змогу забезпечити швидкий розвиток додатків та легкість оновлення.

Основні переваги: висока гнучкість, можливість індивідуального масштабування сервісів, покращення продуктивності команд, незалежне розгортання та оновлення.

Головні виклики: забезпечення надійної комунікації, управління даними, безпека і складність моніторингу.

Рекомендується використовувати мікросервіси для складних, великомасштабних проєктів, де важлива масштабованість і часті оновлення.

Інструменти для управління системою: використання контейнеризації (Docker) та оркестрації (Kubernetes) для спрощення управління мікросервісами.

Забезпечення безпеки та моніторингу: важливо приділяти увагу захисту API та впровадженню систем моніторингу для кожного сервісу.

Список використаних джерел

1. Taibi D., Lenarduzzi V., Pahl C. Architectural Patterns for Microservices: A Systematic Mapping Study. *Journal of Systems and Software.* DOI: 10.1016/j.jss.2017.03.065.

2. Alshuqayran N., Ali N., Evans R. A Systematic Mapping Study in Microservices Architecture. *Journal of Systems and Software.* DOI: 10.1016/j.jss.2017.03.065.

3. Taibi D., Lenarduzzi V., Pahl C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Software. – IEEE Xplore.* DOI: 10.1109/MS.2017.41212.