### **Міністерство освіти і науки України**

### **Національний технічний університет України «Київський**

### **політехнічний інститут імені Ігоря Сікорського"**

### **Факультет інформатики та обчислювальної техніки**

### **Кафедра інформатики та програмної інженерії**

### **Звіт**

### з лабораторної роботи № 4

### з дисципліни

### «Компоненти програмної інженерії частина 3. Архітектура програмного

### забезпечення»

### **«**Event-driven architecture/Serverlessarchitecture**»**

### **Виконав** ІП-21 Скрипець О., Мунтяну А.

### **Перевірив** Зубик Л. В.

### **Київ 2024**

**Лабораторна робота №4**

**Тема:** Event-driven АБО serverlessархітектура.

**Мета:** Застосувати event-driven та serverless архітектуру для проектування складних систем.

**Завдання event-driven architecture:**

1. Вкажіть варіанти використання (use case) у вашій системі, які потребують event-driven архітектури.
2. Вкажіть, які саме патерни event-driven архітектури використовуються.
3. Вкажіть, які інструменти та технології використовуються та чому.
4. Оновіть діаграми.
5. Event-driven system PoC чи Stream processing PoC.
   1. Якщо до вашої теми не підходить ідея stream processing, то необхідно зробити PoC для event-driven частини - для одного сценарію.
   2. Якщо до вашої теми підходить ідея stream processing (коли є постійний потік вхідних даних), то зробити PoC (proof of concept) для stream processing. Можна використати інструменти на ваш розсуд. Щоб полегшити вибір - почніть розгляд із apache flink або ksqldb.

PoC означає написати небагато коду, щоб показати що підхід (технологія) на прикладі певного сценарію працює. Не треба реалізовувати всю систему і показувати, що вона працює.

**1. Варіанти використання у системі для менторства, які потребують event-driven архітектури:**

1. **Бронювання консультації:** подія бронювання клієнтом консультації може призводити до оновлення графіка ментора, відмітки часу консультації як зарезервованого, а також відправлення сповіщення ментору та клієнту з підтвердженням.

2. **Перенесення або відміна консультації:** події перенесення або відміни консультації можуть приздводити до оновлення графіка ментора, звільнення зарезервованого часу та повідомлення клієнту та ментору про зміни.

3. **Оцінювання та надання відгуків**: події оцінки ментора та надання відгуку користувачем можуть призводити до оновлення рейтингу ментора, відображення нового відгуку в його профілі та надіслання сповіщення ментору про новий відгук.

4. **Редагування профілю ментором:** подія редагування ментором свого профілю може призводити до оновлення інформації на його сторінці та зміни результатів на сторінці пошуку ментора.

5. **Перегляд профілю ментора:** подія обрання користувачем сторінки ментора може бути використана для аналізу популярності та ефективності профілю ментора, а також для видачі рекомендацій клієнтам на основі їхніх попередніх виборів.

5. **Пошук ментора:** подія пошуку користувачем ментора за певними критеріями (наприклад, за навичками чи рейтингом) може призводити до виконання запиту до бази даних для отримання відповідних результатів з фільтрацією чи сортуванням, до відображення цих результатів у веб-інтерфейсі користувача, а також до збереження критеріїв пошуку для майбутніх рекомендацій клієнтам.

6. **Оплата консультації**: подія успішної оплати консультації клієнтом може провокувати завершення бронювання, оновлення стану платежу, а також відправлення сповіщення клієнту про успішну оплату.

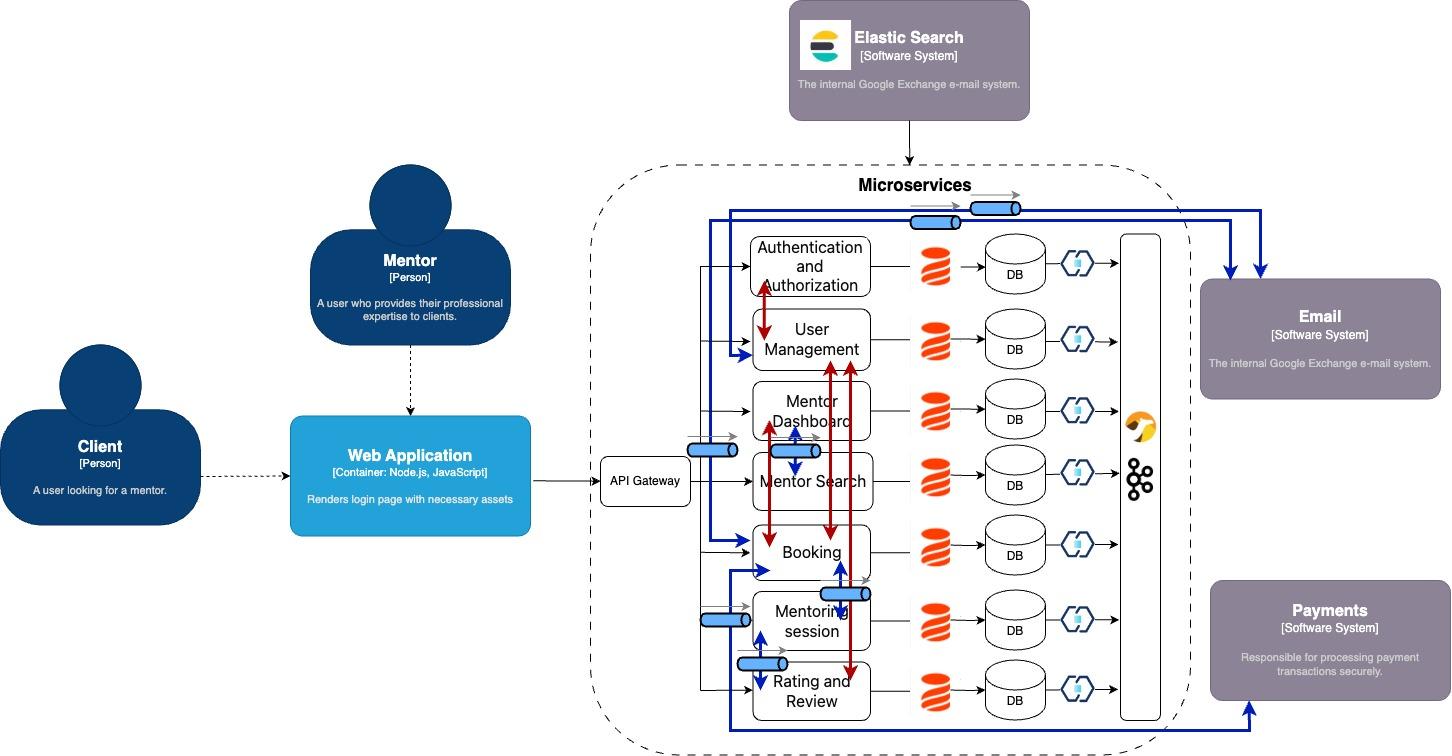
**2. Вкажіть, які саме патерни event-driven архітектури використовуються.**

1. **Event Sourcing:** Цей паттерн дозволяє зберігати стан системи, заснований на послідовності подій. Кожна дія у системі, така як бронювання консультації або оцінювання ментора, генерує подію, яка зберігається у журналі подій. Це дозволяє відтворювати стан системи в будь-який момент часу і створювати аудит-логи для аналізу.
2. **Сommand Query Responsibility Segregation (CQRS)**: Цей паттерн розділяє операції на команди (запис) і запити (читання), дозволяючи використовувати різні моделі для запису та читання даних. Використання CQRS може полегшити масштабування системи та покращити її продуктивність. Так як було обрано Database per Service pattern, використання CQRS є необхідним.
3. **Saga Pattern**: Цей паттерн може бути використаний для керування довготривалими транзакціями, що включають кілька мікросервісів. Він допоможе забезпечити атомічність та консистентність виконання дій, які складаються з кількох кроків або операцій. Наприклад, при процесі бронювання консультації клієнтом можуть виникнути кілька кроків, таких як перевірка доступності часу у ментора, резервування часу, оплата та підтвердження. Якщо один із кроків не вдасться, наприклад, через помилку в оплаті, Saga Pattern дозволить системі виконати відповідні компенсуючі дії, такі як скасування бронювання та повернення коштів користувачу, щоб зберегти консистентність даних і уникнути стану неповної операції.
4. **Domain Events**: Цей паттерн використовується для сповіщення інших частин системи про зміни в доменній моделі. Кожна важлива подія у системі (наприклад, бронювання консультації або оновлення профілю ментора) може бути представлена як доменна подія, яка відправляється усім зацікавленим сервісам.

**3. Вкажіть, які інструменти та технології використовуються та чому.**

1. **Apache Kafka:** Кафка буде використовуватися як потужний та масштабований брокер подій для зберігання та передачі потоку подій між мікросервісами системи. Вона дозволить реалізувати Event Sourcing, зберігаючи всі події та стан системи у потоках, а також забезпечить надійну та асинхронну комунікацію між компонентами.
2. **Apache Camel**: Apache Camel буде використовуватися для реалізації Saga Pattern, дозволяючи керувати послідовністю транзакцій та компенсуючих дій у ї системі. Він надасть інструменти для розробки та виконання різноманітних компонентів саг, які взаємодіють з Apache Kafka та іншими сервісами.
3. **ElasticSearch**: ElasticSearch буде використовуватися для забезпечення потужного та швидкого пошуку, індексації та аналізу даних у системі. Він дозволить зберігати та опрацьовувати великі обсяги даних, наприклад, відгуки клієнтів та іншу неструктуровану інформацію.

**4. Оновіть діаграму.**

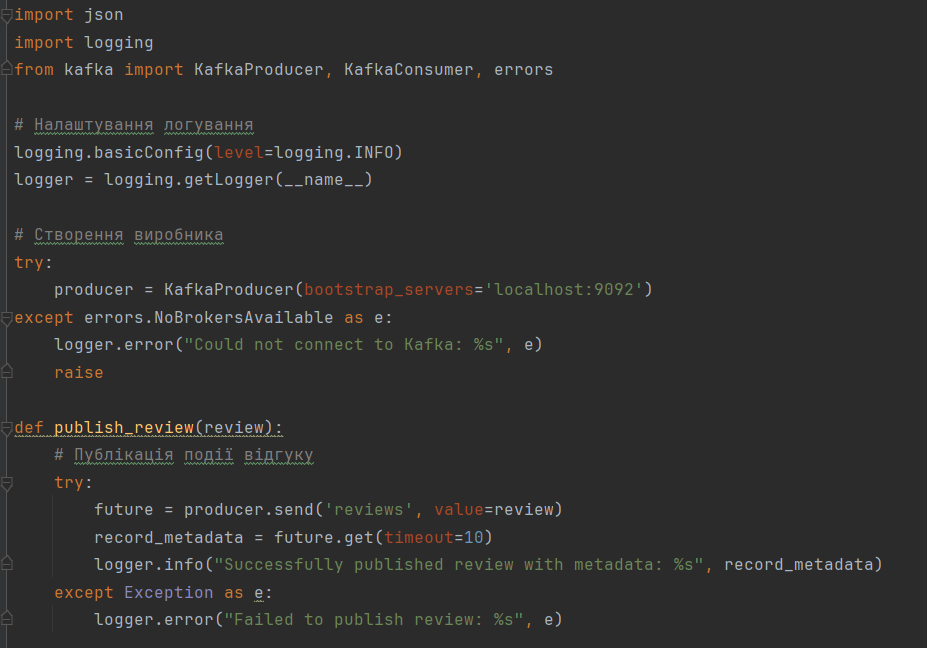
****

**5. Event-driven system PoC**

Для сценарію публікації відгуку користувачем на платформі для менторства ми використали event-driven систему.

1. Патерни:
   * **Domain Events**: Коли користувач публікує відгук, відбувається подія, яка представляє зміну стану в домені. Ця подія відгуку публікується в Kafka за допомогою виробника в функції publish\_review. Тут відгук виступає як доменна подія. Споживач слухає ці події відгуку в Kafka і реагує на них. В даному випадку, він використовує функцію notify\_mentor для відправлення сповіщення ментору, коли отримує нову подію відгуку. Тут споживач реагує на доменну подію.
2. Технології:
   * **Apache Kafka**: Kafka використовується в цьому коді як центральна система для обробки подій, що дозволяє публікувати та споживати події відгуку в реальному часі. Вона використовується для створення виробника, який публікує події відгуку на сервер Kafka. Виробник, створений за допомогою Kafka, використовується для публікації подій відгуку на тему ‘reviews’ в Kafka.

Ось приклад коду для Apache Kafka, який може бути використаний для реалізації цього сценарію. Цей код включає в себе виробника(користувач), який публікує подію відгуку, і споживача, який слухає цю подію та відправляє сповіщення.





Цей код включає в себе обробку помилок при підключенні до Kafka, логування успішних та невдаліх спроб публікації відгуків, а також логування отриманих відгуків.

Логування отриманих відгуків - це процес запису інформації про отримані відгуки в систему логування. Це допомагає відстежувати, які відгуки були отримані, коли вони були отримані, а також може допомогти виявити будь-які проблеми або помилки, які можуть виникнути під час обробки відгуків.

У наведеному вище коді, логування отриманих відгуків відбувається в функції notify\_mentor. Кожен раз, коли споживач Kafka отримує новий відгук, він записує інформацію про цей відгук в систему логування за допомогою команди logger.info("New review published: %s", review). Це дозволяє відстежувати, які відгуки були опубліковані і коли.

Це особливо корисно для відлагодження та моніторингу системи, ми зможемо швидко переглянути логи, щоб дізнатися, які відгуки були опубліковані, і виявити будь-які можливі проблеми.