Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Брокер обміну та обробки повідомлень у реальному часі»**

Виконала: студентка 4-го року навчання,

Спеціальності
122 «Комп'ютерні Науки»

Чурілова Анна Сергіївна

Керівник Бабич Т.А.

магістр комп'ютерних наук, асистент

«20» травня 2022 р.

Київ – 2022

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 122 «Комп'ютерні науки»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

"10" жовтня 2022 року

ЗАВДАННЯ
ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Чуріловій Анні

1. Тема роботи **«Брокер обміну та обробки повідомленнями у реальному часі»**, керівник роботи Бабич Трохим Анатолійович, магістр комп'ютерних наук, асистент

2. Строк подання студентом роботи 20 травня 2022

3. План роботи

Анотація

Вступ

Розділ 1. Дослідження та аналіз предметної області

1.1 HTTP протокол

1.1.1 Обмін повідомленнями із використанням HTTP вебхуків

1.2 Специфіка технічних систем реального часу
1.3 Розробка систем реального часу із застосуванням Apache Kafka
1.3.1 Архітектура Apache Kafka
1.3.2 Використання Apache Kafka
1.3.3 Переваги над аналогами
1.3.4 Недоліки технології

Розділ 2. Проектування та розробка системи

2.1 Опис складових системи та використаних технологій для розробки

2.2 Налагодження взаємодії між учасниками

2.3 Робота з фоновими потоками, синхронізація потоків за допомогою семафорів

2.4 Організація черг даних та переформатовування даних

2.5 Використання баз даних

Висновки

Список використаних джерел

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	26 жовтня 2021			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	26 жовтня 2021 – 2 листопада 2021			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	2 листопада 2021			
4.	Написання розділів роботи	2 листопада 2021 – 20 квітня 2022			
5.	Проміжний контроль виконання роботи	02 лютого 2022			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	21 листопада 20221– 01 травня 2022			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	25 січня 2022			
	Розділ 2 (аналітично-дослідницька частина)	01 березня 2022			
	Розділ 3 (проектно-рекомендаційна частина)	29 березня 2022			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	04 травня 2022 – 10 травня 2022			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	20 травня 2021			

Графік узгоджено 10 жовтня 2022 р.

Науковий керівник Бабич Трохим Анатолійович

Виконавиця курсової роботи Чурілова Анн

ЗМІСТ

Анотація ... 1

Перелік термінів та умовних позначень ... 2

Вступ ... 3

Розділ 1. Дослідження та аналіз предметної області ... 4

1.1 HTTP протокол ... 4

1.1.1 Обмін повідомленнями із використанням HTTP вебхуків ... 6

1.2 Специфіка технічних систем реального часу ... 7

1.3 Розробка систем реального часу із застосуванням Apache Kafka ... 8

1.3.1 Архітектура Apache Kafka ... 8

1.3.2 Використання Apache Kafka ... 11

1.3.3 Переваги над аналогами ... 12

1.3.4 Недоліки технології ... 14

Розділ 2. Проектування та розробка системи ... 15

2.1 Опис складових системи та використаних технологій для розробки ... 15

2.2 Налагодження взаємодії між учасниками ... 21

2.3 Робота з фоновими потоками, синхронізація потоків за допомогою семафорів ... 23

2.4 Організація черг даних та переформатовування даних ... 25

2.5 Використання баз даних ... 28

Висновки ... 30

Список використаних джерел ... 32

Анотація

У даній роботі розглядаються розробка програмних систем в режимі реального часу, архітектура та принципи роботи популярного брокера повідомлень Apache Kafka та безпечна передача повідомлень між компонентами програм.

Метою роботи є дослідження розробки інструментів із використанням найбільш популярних брокерів повідомлень та створення власного аналогу такої системи.

Результатом проведеної роботи є завершений програмний продукт, що, на прикладі обробки показників здоров'я пацієнтів лікарні, демонструє роботу брокера обміну та обробки повідомленнями у режимі реального часу. Така галузь, як медицина є однією із найбільш вибагливих до швидкості передачі даних, оскільки від цього може залежати людське життя. Саме тому актуальність таких систем очевидна у наш час. Розроблена програма має мікросервісну архітектуру, компоненти якої спілкуються один з одним за допомогою безпечних HTTP-запитів.

Платформа може бути використана у будь-якій галузі, що потребує швидкої, коректної та безпечної комунікації зі сторонніми компонентами.

Перелік термінів та умовних позначень

API - Application Programming Interface. Інструмент для взаємодії кількох програм, що дає змогу розширяти функціональність певного продукту, пов'язуючи його як зі своїми, так і з чужими розробками.

URL - Uniform Resource Locator показник місцезнаходження будь-яких веб-ресурсів у Всесвітній павутині.

Реплікація – процес копіювання даних одного з ресурсу в інший.

Журналювання – процес зберігання інформації про поведінку програмного забезпечення (виникнення та причини помилок, хронологія виконання завдань тощо), що може зберігатися на сервері чи комп'ютері у різноманітних форматах.

Масштабованість (систему можна збільшувати за рахунок простого додавання нових вузлів (брокерів повідомлень)).

NoSQL[1] **–** тип нереляційних баз даних, які зберігають інформацію у форматі, відмінному від реляційних таблиць.

XML - eXtensible Markup Language. Мова розмітки даних, що забезпечує зберігання інформації у форматі, який може підтримуватися будь-якою системою.

JSON – JavaScript Object Notation. Один із найбільш популярних форматів зберігання даних, що підтримується великим різноманіттям систем.

Вступ

Людство завжди потребувало спілкування один з одним. Від покоління до покоління наші пращури передавали певну інформацію про традиції, наукові наробки та багато іншого. Обмін повідомленнями завжди був одним із найбільш необхідних аспектів життя. В наші дні особливо важливо знати про те, що відбувається у світі, читати новини та передавати правдиву інформацію іншим.

Беручи до уваги стрімкий розвиток не тільки інформаційних технологій, а й будь-яких інших сфер життя, постійно отримувати актуальні дані є запорукою успіху для кожного підприємства. Саме тому в наш час такими поширеними стають системи обробки повідомлень від різноманітних джерел в режимі реального часу.

Проаналізувавши актуальність таких систем, було обрано у якості теми курсової роботи проведення дослідження та розробка власного продукту, який виконував би такі функції.

Метою даної роботи є створення брокера передачі та обміну повідомленнями в режимі реального часу на прикладі показників здоров'я пацієнта лікарні. Основним завданням розробленого програмного застосунку є оповіщення різноманітних споживачів (наприклад медсестри, лікаря) про життєво-загрозливі заміри таких показників, як пульс, рівень глюкози у крові та інші.

Завданням курсової роботи є розробка повного циклу передачі інформації від виробника – апаратів вимірювання даних про здоров'я людини до будь якого споживача, кому ця інформація необхідна у безпечний та швидкий спосіб на основі дослідження роботи відомого брокера повідомлень Apache Kafka.

Розділ 1. Дослідження та аналіз предметної області

1.1 HTTP протокол[2]

Протокол передачі гіпертексту, або HyperText Transfer Protocol(HTTP) є основою всесвітньої павутини (WWW), що використовується для завантаження веб-сторінок за допомогою гіпертекстових посилань. HTTP — протокол прикладного рівня, що необхідний бля того, щоб передавати інформацію між різними мережевими пристроями і працює над усіма іншими рівнями стеку протоколів. Робота з HTTP являє собою запит клієнта до сервера, який відправляє повідомлення з відповіддю назад.

Всі HTTP-запити, мають певний набір шифрованих даних, що містять інформацію про нього. Компонентами, що входять в такий запит є URL, HTTP метод, заголовки запиту HTTP та опційне тіло запиту HTTP.

HTTP метод вказує на дію, що очікується від сервера. Одним із таких методів може бути:

- GET-метод запитує всі дані ресурсу, які є. Запит методом GET містить лише URL та заголовки;
- HEAD-метод запитує дані певного ресурсу. Проте, на відміну від GET, відповідь приходить без тіла запиту;
- POST-метод використовується для створення ресурсу. Запит POST вимагає тіло запиту, в якому визначається об'єкт в різних форматах (зазвичай JSON), який має бути створено;
- PUT-метод вносить певні зміни в існуючий ресурс або створює новий;
- DELETE-метод видаляє ресурс. Тіло запиту у цього метода відсутнє, зазвичай видалення відбувається за унікальним ідентифікатором, який передається в параметри URL;

- TRACE-метод відображає зміни, що були внесені до ресурсу;
- OPTIONS-метод відображає, які HTTP- методи можуть бути застосовані для конкретної URL-адреси. Запит не містить тіла;
- CONNECT-метод перетворює з'єднання запиту в тунель TCP/IP;
- PATCH-метод частково змінює ресурс. Запит містить тіло;

Ідемпотентність — одна із властивостей HTTP методу, яка описує повернення від серверу однакового результату після кожного виклику, при відправленні однакових запитів. Безпечність — властивість HTTP методу, яка описує здатність не змінювати ресурс.

Проаналізуємо наведені методи за цими властивостями:

Метод	Безпечність	Ідемпотентність
GET	Так	Так
HEAD	Так	Так
POST	Ні	Ні
PUT	Ні	Так
DELETE	Ні	Так
TRACE	Так	Так
OPTIONS	Так	Так
CONNECT	Ні	Ні
PATCH	Ні	Ні

Заголовки HTTP- запиту містять в собі інформацію в форматі ключ-значення. Ці заголовки передають основну інформацію, наприклад, який браузер використовує клієнт, які дані запитуються, який токен авторизації.

Тіло запиту — це частина запиту, яка містить більшість інформації, що передається. Тіло HTTP-запиту містить в собі будь-які дані, що були відправлені на сервер, наприклад із певної форми на веб-сайті.

Відповідь HTTP — це відповідь на HTTP-запит від сервера, що з'являється на стороні клієнта. Цей компонент відповідає за інформацію, яку може надати сервер на основі певного HTTP-запиту. Зазвичай відповідь HTTP містить код статусу HTTP, заголовки відповіді HTTP та опційне тіло HTTP.

Коди статусу HTTP – це 3-значні коди, які необідні для того, щоб відобразити успішність, невдачу або помилку відпрацювання HTTP-запиту. Коди стану розбиті на наступні 5 блоків ("xx" відноситься до різних чисел від 00 до 99):

- 1xx - інформаційний;
- 2xx - успішний;
- 3xx – перенаправлення;
- 4xx - помилка клієнта;
- 5xx - помилка сервера;

Успішні відповіді HTTP, наприклад на запити методом GET, у більшості випадків мають тіло, яке містить запитувані дані ресурсу. Зазвичай це дані, представлені у вигляді HTML, які веб-браузер перекладає на веб-сторінку.

1.1.1 Обмін повідомленнями із використанням HTTP вебхуків

Вебхук[3] — це зворотній виклик HTTP. HTTP POST-запит, який реагує на якусь подію та спрацьовує у відповідь на неї. Вебхуки використовуються для підтримки сповіщень у режимі реального часу, тому система, яка їх використовує може оновлятись одразу, коли станеться певна подія (сповіщення буде відправлено). Вебхуки дозволяють зареєструвати URL-

адресу, де дані події зберігатимуться в потрібних форматах. Вебхук виконує зворотний виклик HTTP до цієї, налаштованої системою, яка отримує дані, URL-адреси. Така адреса називається кінцевою точкою вебхуку. Точки мають бути загальнодоступними, і обов'язково, щоб ця URL-адреса належала отримувачу. Зворотний виклик запускається щоразу, коли виникає подія, про яку необхідно повідомити перед тим визначену систему.

1.2 Специфіка технічних систем реального часу

Система реального часу – це така система, що продовжує свою роботу постійно в режимі реального часу. Іншими словами певна дія (запит) обов'язково повинна бути виконана в межах визначеного часу або з визначеною періодичністю.

Види систем реального часу за можливістю помилки:

- Жорстка – це система, яка в жодному разі не може пропустити визначений термін спрацювання. Невчасне виконання може мати непоправні наслідки. І з кожною секундою після запізнення, інформація, що мала дістатися такою системою втрачає свою корисність та несе катастрофічні збитки. Запізнення означає, на яку кількість часу пізніше система виконує своє завдання відповідно до встановленого терміну.
- М'яка – це система, яка має право іноді пропускати термін з прийнятною ймовірністю. Пропущення терміну не несе таких сильних збитків, як у випадку із жорсткою системою. Корисність результатів поступово зменшується зі збільшенням запізнення системою.

1.3 Розробка систем реального часу із застосуванням Apache Kafka

Kafka може будувати потоки даних у режимі реального часу, які надійно відправляють інформацію із однієї системи на іншу. Apache Kafka є однією з найбільш популярних серед розробників технологій і одним з найбільш використовуваних інструментів для передачі великої кількості даних. Kafka присутня у системах понад 12 000[4] IT-компаній по всьому світу. Вона надає їм вигідний спосіб переді даних у режимі реального часу, коли інші брокерські технології зазнають невдачі. Kafka продовжує зростати в кількох галузевих сегментах. Ця технологія ідеально підходить постійно змінюваним та збільшуваним ринкам, що працюють із великими обсягами даних, які потрібно обробляти в момент їх появи.

1.3.1 Архітектура Apache Kafka

Apache Kafka пропонує чотири ключові API[5] : API Producer, API Consumer, API Streams і API Connector з такими функціями, як резервне зберігання великих обсягів даних, що передаються повідомленнями, кількість яких при цьому сягає до мільйону на секунду.

- API Producer дозволяє програмі відправляти (публікувати) потік даних в одну або декілька тем Kafka;
- API Consumer дозволяє програмі підписатися на одну або кілька тем, щоб отримувати з неї дані, а також працювати з потоками записів, доданих до цих тем;
- API Streams дозволяє програмі використовувати вхідні потоки з однієї або кількох тем, обробляти їх як звичайні потоки і в результаті

створювати вихідні потоки з подальшим відправленням їх до однієї або кількох тем;

- API Connector дозволяє з'єднувати програми або інші компоненти з темами Kafka. Це надає можливості взаємодії та керуванням роботою виробників і споживачів, а також досягнення зв'язків між цими рішеннями;

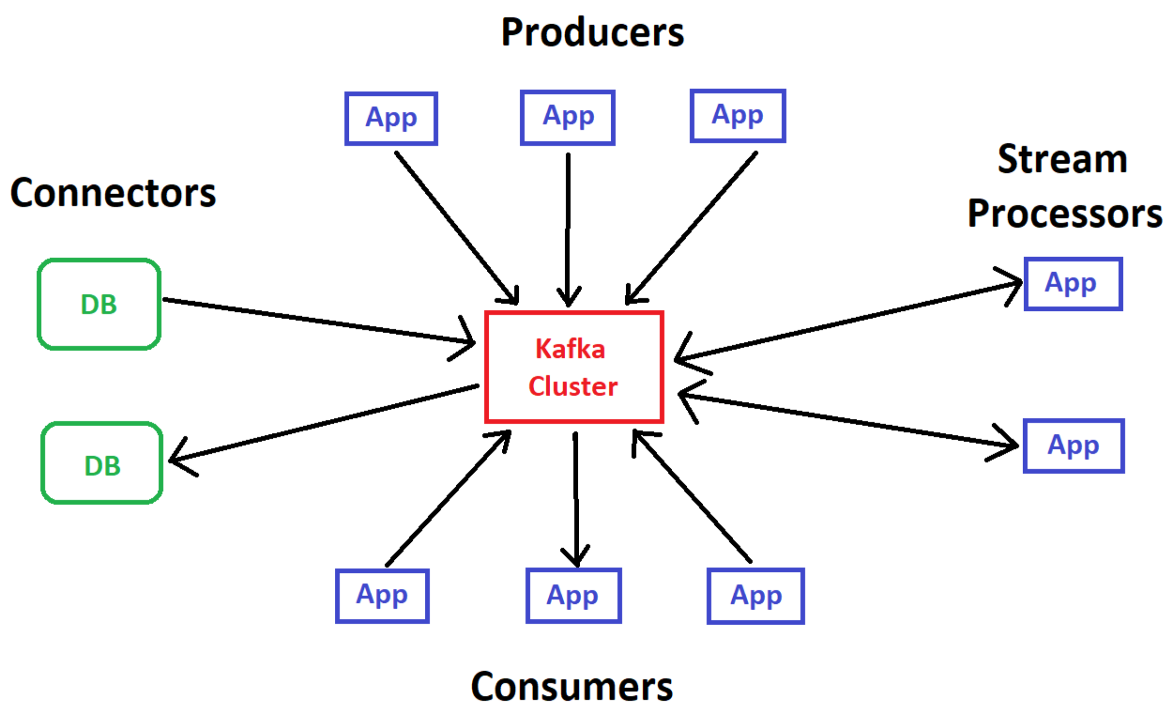

Рис. 1.1 Схематичне зображення Kafka API

Kafka являє собою кластер, що складається із одного або кількох серверів, що можуть бути розширені у великих кількостях.

Компоненти кластеру:

- Kafka брокер

Задля забезпечення балансу навантаження, кластер Kafka у більшості випадків містить у собі певну кількість брокерів. Брокер відповідає за зберігання даних. Всі вони мають бінарний вигляд, і він мало знає про те, які вони та яку структуру мають. Один Kafka брокер може обробляти сотні тисяч читань-записів на секунду.

- Kafka ZooKeeper

Для управління своїми брокерами Kafka використовує так званий ZooKeeper. Також ZooKeeper потрібен для того, щоб повідомляти виробника та споживача про створення нового брокера або збій в роботі вже існуючого. Як тільки Zookeeper відправляє звіт про роботу брокерів, виробники і споживачі, в залежності від ситуації, вирішують свої завдання з іншим брокером тощо.

- Kafka продюсери

Найчастіше являють собою сервіс, що безпосередньо здійснює додавання даних до Apache Kafka. Продюсер обирає та призначає Kafka тему, у якій зберігатимуться його повідомлення, відсортовані за певними темами, і починає додавати у нього інформацію.

- Kafka споживачі

Споживачі – це той компонент кластеру, який отримує дані із Apache Kafka. Цей сервіс буде підписаний на такі теми, які йому потрібні і, як тільки з’являться нові повідомлення в цих темах, буде отримувати їх та поводитися з ними як йому потрібно. Kafka запам'ятовує, які останні події отримав певний споживач, що допомагає гарантувати отримання одного повідомлення лише одного разу, без дублікатів.

- Kafka теми

Теми є такими собі сховищами, в які продюсери публікують свої повідомлення, а споживачі їх отримують. Певний тип повідомлень публікується на відповідну тему, таким чином додаючи певний порядок та логічну організацію даних. Унікальним ідентифікатором теми є її назва, про яку відомо всім учасникам обміну повідомленнями. Кількість тем в кластері не обмежена. Після додавання даних в тему, можливість змінювати їх стає неможливою.

- Kafka партиції

Теми Кафки поділені на кілька розділів (партицій). Партиція — це найменша одиниця зберігання даних в Kafka, яка містить підмножину записів, що належать певній темі. Кожен розділ являє собою один файл журналу, в який додаються дані. Кожному запису в розділах призначається унікальний ідентифікатор, для розрізнення даних в межах однієї партиції.

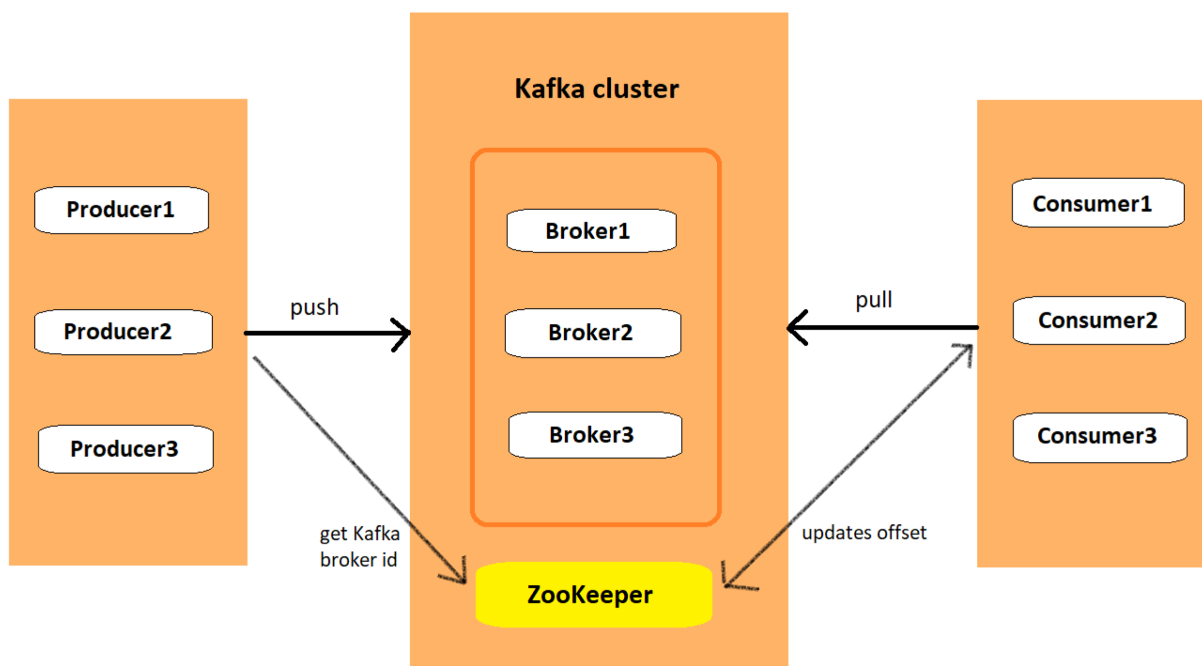

Рис. 1.2 Схематичне зображення компонентів кластеру Kafka

1.3.2 Використання Apache Kafka

Основні приклади використання Kafka[6] :

- Обмін повідомленнями

Kafka створена для заміни традиційного брокера повідомлень між користувачами. Вона має кращу між аналогами пропускну здатність, можливість реплікацій та стійкість до відмов, що робить її гарним рішенням для програм обробки повідомлень великих масштабів.

- Відстеження активності на веб-сайті

З самого початку Kafka використовувалась для відстеження активності користувачів у вигляді набору каналів публікації та підписки в режимі реального часу. Це означає, що дії користувача на сайті (натискання на посилання, перехід між сторінками, завантаження файлів тощо) публікується в Kafka темах відповідно до типу діяльності. На такі канали можна підписуватися і аналізувати, обробляти та моніторити поведінку користувачів в режимі реального часу.

- Агрегація логів

Агрегація логів передбачає збір журналювання із серверів і розміщення їх у сховищі задля подальшої обробки. Кафка проводить певну фільтрацію файлів журналювання та абстрагує дані у вигляді потоку повідомлень. Це забезпечує швидшу обробку та легшу підтримку кількох джерел даних.

1.3.3 Переваги над аналогами

Основою архітектур, що працюють з BigData, є потокова обробка даних. Стандартом роботи із передавання повідомлень є Apache Kafka.

Головним конкурентом Kafka серед розробників є технологія RabbitMQ[7]. RabbitMQ – це відомий брокер повідомлень, що може працювати із багатьма мережевими протоколами. Apache Kafka сьогодні набуває все більшої популярності, а RabbitMQ починає відходити на задній план. Однією із причин, чому Kafka привертає більше уваги є те, що Kafka здатна до масштабованості краще. Кількість повідомлень, що можуть надсилатися в системі кожну секунду для Kafka досягає до 1 млн, коли RabbitMQ – біля 50 тис.

Щодо архітектурної відмінності між даними технологіями - вона у тому, що RabbitMQ управляє повідомленнями майже на рівні пам'яті і тому використовує для цього великий кластер, що нараховує більше, ніж 30 вузлів, тоді як Kafka використовує можливості послідовних операцій дискового вводу-виведення та потребує менше апаратного забезпечення.

Ще одним аналогом Kafka є Redis - база даних у форматі ключ-значення NoSQL, яка також є дуже популярною у використанні. Не зважаючи на те, що це система зберігання даних в парах ключ-значення, вона сама підтримує функції MQ, тому може бути використана як спрощений варіант черги. Продуктивність Redis вища, ніж у RabbitMQ на малих даних, проте якщо взяти дані, що перевищуватимуть 10КБ, робота Redis стає надто повільною. Найчастіше Redis стає в нагоді при кешуванні відповідей на запити. З Redis можна зменшити кількість викликів на різні ресурси, в тому числі і на базу даних. Таким чином, завдяки близькому розташуванню до кінцевого користувача, прискорюється отримання відповіді на запит.

1.3.4 Недоліки технології

Важливо розглянути і недоліки Kafka, аби враховуючи їх, обирати потрібну технологію[7].

- Відсутній набір інструментів моніторингу

Деякі компанії відмовляються використовувати Kafka в довгостроковій перспективі через нестаток повного набору інструментів управління та моніторингу.

- Проблеми з налаштуванням повідомлень

Продуктивність Kafka значно знижується у випадку потреби для будь-яких налаштувань даних (повідомлень). Тому з цим брокером краще працювати передаючи повідомлення, які не потребують змін.

- Знижує продуктивність

Брокери та споживачі знижують продуктивність Kafka, стискаючи та розпаковуючи потік даних, коли збільшується їх кількість. Це впливає не лише на його продуктивність, але й на пропускну здатність.

- Зниження швидкості роботи

При збільшенні кількості черг у кластері, Kafka починає поводитися трохи повільно.

Розділ 2. Проектування та розробка системи

2.1 Опис складових системи та використаних технологій для розробки

Програмний продукт являє собою систему передачі повідомлень про стан здоров'я пацієнта в режимі реального часу. Заміри тих чи інших критерій оцінки відправляються із відповідних апаратів перевірки до лікарів, медсестер та інших апаратів в момент їх появи.

Структура проекту включає в себе 3 основні компоненти: продюсери, споживачі та сам конвеєр «Kafka».

- Продюсери

У проекті створено 4 продюсери – BloodPressureProducer, BreathingRateProducer, GlucoseLevelProducer та PulseProducer. Вони виступають у ролі «апаратів» вимірювання артеріального тиску, частоти дихання, рівня глюкози у крові та пульсу пацієта відповідно. На рівні програмного коду являють собою нескінченні цикли, які із певною періодичністю відправляють HTTP POST-запити (вебхуки), що містять випадкові дані про відповідні заміри, на Receiver – одну із складових частин «Kafka».

Кожен із продюсерів містить в заголовку запиту так званий «Receiver-Secret» - кодове слово, що є обов'язковим для можливості доступу до Receiver'а. Таким чином реалізована авторизація продюсерів, що забезпечує перевірку компонентів, які звертаються до «Kafka». У разі пропущення даного заголовку, або вказання невірного коду, продюсер отримає помилку клієнта "Unauthorized" зі статусом 401.

- Споживачі

Дані із продюсерів можуть бути передані на один із існуючих в системі споживачів: ArtificialRespiratorConsumer, HeadPhysicianConsumer, InsulinPumpConsumer і NurseConsumer. Вони є кінцевими точками передачі замірів про стан пацієнта, які мають реагувати на екстремально низькі або високі показники. Апарата штучного дихання цікавить частота дихання, головного лікаря – пульс та частота дихання, інсулінову помпу – рівень глюкози у крові, а медсестру – усі можливі показники.

Кожен споживач під'єднаний до своєї бази даних (імітація справжньої ситуації), де зберігає важливі йому показники. На рівні програмного коду являють собою контролери, які приймають пакети з даними із компоненту «Kafka» Sender. Кожен споживач підтримує різний формат даних – NurseConsumer та ArtificialRespiratorConsumer чекають дані у форматі JSON, решта - XML. Після отримання інформації, вона десеріалізується із отриманих форматів у зрозумілий програмі в залежності від типу пакету. Заміри додаються до бази даних та відображаються на екрані.

Breathing Rate Measurements

Patient Surname	Patient Name	RoomNumber	Measurement	Max Measurement	Min Measurement	Measurement Time
Dykhner	Ivan	210	21	19	14	5/11/2022 12:17:05 AM
Dykhner	Milena	204	22	19	13	5/11/2022 12:17:05 AM
Dykhner	Milena	206	23	19	14	5/11/2022 12:17:05 AM
Krupko	Milena	213	23	20	14	5/11/2022 12:17:06 AM
Ivanov	Milena	213	12	21	14	5/11/2022 12:17:05 AM
Zhurbylo	Taras	207	23	21	14	5/11/2022 12:17:05 AM
Ivanov	Olga	214	22	21	13	5/11/2022 12:17:06 AM
Krupko	Olga	205	25	20	14	5/11/2022 12:17:06 AM
Krupko	Taras	208	23	19	12	5/11/2022 12:17:06 AM
Zhurbylo	Ivan	212	25	21	13	5/11/2022 12:17:06 AM
Ivanov	Olga	200	11	19	14	5/11/2022 12:17:06 AM
Zhurbylo	Milena	211	25	20	12	5/11/2022 12:17:06 AM
Dykhner	Yaroslav	214	11	21	13	5/11/2022 12:17:06 AM
Dykhner	Ivan	205	22	20	12	5/11/2022 12:17:07 AM
Ivanov	Ivan	203	25	21	13	5/11/2022 12:17:07 AM
Zhurbylo	Taras	201	20	19	14	5/11/2022 12:17:07 AM
Dykhner	Olga	202	23	21	12	5/11/2022 12:17:07 AM
Ivanov	Olga	203	22	19	12	5/11/2022 12:17:07 AM
Krupko	Olga	201	24	19	13	5/11/2022 12:17:07 AM
Krupko	Yaroslav	213	12	19	13	5/11/2022 12:17:07 AM
Dykhner	Milena	208	11	20	12	5/11/2022 12:17:07 AM
Zhurbylo	Yaroslav	202	20	19	12	5/11/2022 12:17:07 AM
Krupko	Yaroslav	205	12	21	14	5/11/2022 12:17:07 AM
Dykhner	Ivan	206	23	19	12	5/11/2022 12:17:08 AM
Tovstiuk	Taras	202	22	20	14	5/11/2022 12:17:08 AM
Ivanov	Olga	212	11	19	12	5/11/2022 12:17:08 AM
Zhurbylo	Taras	202	24	21	13	5/11/2022 12:17:08 AM

Рис 1.3 Зображення інтерфейсу розробленої програми – список показників частоти дихання

- «Kafka»

Серцем передачі даних в режимі реального часу у розробленому проекті являється конвеєр «Kafka», який скаладється із шести проектів: Common, Db, Db.Migrations, Receiver, Dispatcher та Sender.

Common – це проект-бібліотека, який містить у собі компоненти, які використовуються у декількох інших проектах системи. Для зручності та зменшенні кількості програмного коду було вирішено винести певні частини в одне місце. Інші проекти мають доступ до сервісів, інтерфейсів, моделей тощо даного проекту за рахунок встановлення так званих ProjectReference до файлів *.csproj.

Db – проект-бібліотека, який відповідає за роботу з базою даних лікарні – HospitalDbContext. Містить класи, які працюють із такими таблицями, як Ticket та TicketToQueue.

Db.Migrations – проект-бібліотека, який зберігає весь список міграцій бази даних проекту Db. Файли міграції містять інформацію про дату та час її створення і які зміни були внесені до бази даних та її таблиць. На основі кожного такого файлу можна відтворити стан сховища на всіх етапах розробки застосунку, повернути минулі версії та згенерувати нові. Потрібно лише додати у будь-який проект, що працює із базою даних, залежність на Microsoft.EntityFrameworkCore.Design і виконати потрібну команду в терміналі.

Receiver – один із трьох найголовніших компонентів «Kafka». Відповідає за отримування різних показників стану здоров'я пацієнтів від апаратів контролю, а також створення та зберігання відповідних тікетів. Перевірка авторизації продюсерів відбувається за рахунок створеного атрибуту VerifyRequiredHeadersAttribute. Такий атрибут перевіряє валідність кодового слова із заголовку запиту і, в залежності від його правильності, вирішує, допускати даний запит до себе чи ні. Застосування атрибуту відбувається у контролері проекту для кожного типу отримуваних даних. Після виконання усієї вищевказаної роботи, Receiver відправляє HTTP POST-запит із унікальним ідентифікатором створеного тікету до Dispatcher'а.

Dispatcher - наступний, після Receiver'а, крок у системі «Kafka». Отримавши id тікета, він достає його повністю із бази даних, та починає складати в різні пакети для подальшого додавання у черги. По-перше, відбувається розпізнавання виду показників та створення відповідних моделей даних

(пакетів) – для кожного заміру окремо, спільного для частоти дихання та пульсу і для всіх показників разом. Це зроблено задля формування різних черг для відправки на подальші компонети системи. По-друге, відбувається перевірка критичності показників – якщо вони виходять за межі норми – відправка до кінцевого отримувача відбувається негайно.

Варто зазначити, що Dispatcher використовує так звані фонові потоки, які дозволяють проекту працювати у фоновому режимі та обробляти дані у той момент, коли вони додаються до черг. Для синхронізації потоків в межах черг тут було використано семафор (див. Про організацію черг та семафорів у п.2.3 та 2.4).

Вкінці своєї роботи Dispatcher відправляє створені пакети даних на останній крок конвеєру – Sender.

Sender – на відміну від Dispatcher має лише одну чергу, куди додає отримані від нього пакети. Після потрапляння до черги, пакет проходить переформатовування своїх даних у той формат, який підтримує споживач, що його чекає. Так само, як і Dispatcher, у фоновому режимі, за допомогою семафору, відправляє дані з черги до споживачів.

Загалом, схема складових частин проекту виглядає так:

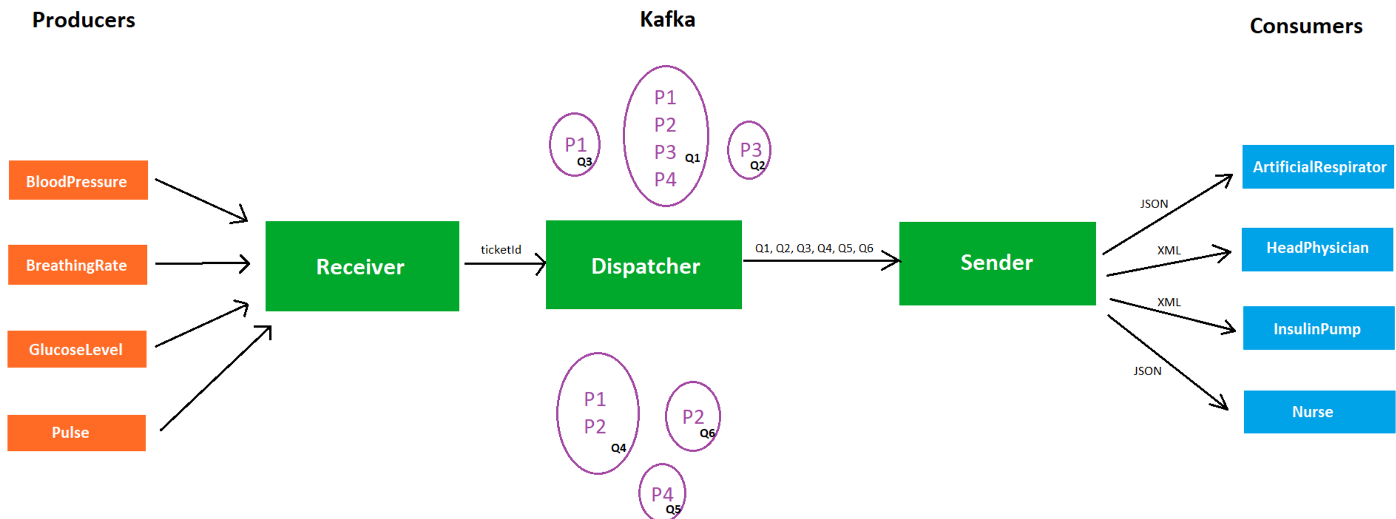

Рис 1.4 Схема складових частин розробленої програми

Проект створено з використанням .NET Framework - програмної платформи, випущеної компанією Microsoft у 2002 році. Основою платформи є багатомовне середовище виконання Common Language Runtime (CLR), яке підтримує різні мови програмування. У даній роботі використовувалась мова C# із версією .NET Framework'а 3.1. Розробка відбувалась у середовищі розробки Microsoft Visual Studio, що дає змогу створювати не тільки консольні застосунки, а й такі, що підтримують графічний інтерфейс програми.

Робота зі сховищами відбувалась за допомогою Entity Framework – це рішення для взаємодії з базами даних, яке використовується разом з мовами програмування сімейства .NET (C#, F#, VB.NET, тощо). Він дозволяє взаємодіяти із системою керування базами даних за допомогою сутностей (entity), а не таблиць. Для зберігання та відображення даних у сховищі

використовувався PostgreSQL - популярна об'єктно-реляційна система управління базами даних. PostgreSQL базується на мові SQL та підтримує велику кількість можливостей для роботи зі сховищем.

2.2 Налагодження взаємодії між учасниками

Повідомлення передаються між учасниками процесу за допомогою HTTP запитів (вебхуків), які містять інформацію про показники, унікальні ідентифікатори тікетів, назви споживачів тощо.

Як зазначалось раніше, продюсери мають право звертатися POST-запитом до Receiver'а лише тоді, коли в мають в заголовку правильний Receiver-Secret. У тілі запиту міститься інформація про стан здоров'я пацієнта та його самого – ім'я, прізвище, номер кімнати, дата та час створення заміру, сам показник, його верхня можлива границя та нижня. URL запитів із продюсера до Receiver'а виглядають так так: https://localhost:5010/RegisterBloodPressure, https://localhost:5010/RegisterBreathingRate,https://localhost:5010/RegisterGlucoseLevel, https://localhost:5010/RegisterPulse, де localhost:5010 - це хост та порт, на яких працює Receiver, а останній компонент URL – показник відповідної точки API, на яку мають бути відправлені потрібні заміри.

Receiver спілкується із Dispatcher'ом схожим чином – відбувається надсилання GET-запиту на такий порт: https://localhost:5020. Цього разу важлива інформація (унікальний ідентифікатор тікету) передається не через тіло запиту, а через параметри URL. Таким чином тікет, id якого 1, буде отримано на стороні Dispatcher'а за таким запитом: https://localhost:5020/send/dispatcher/1.

В Dispatcher'і пакети формуються в черги, з яких надсилаються у вигляді тіла POST-запитів на різні точки входу проекту Sender. URL ж запиту наприклад на показник пульсу виглядатиме так: https://localhost:5030/sendPulse/sender, рівень глюкози - https://localhost:5030/sendGlucoseLevel/sender, частоту дихання - https://localhost:5030/sendBreathingRate/sender і артеріальний тиск - https://localhost:5030/sendBloodPressure/sender.

Останнім етапом роботи є передача набору даних у форматах JSON та XML до споживачів. POST-запит містить у своєму тілі переформатовані дані пакетів, а URL має вигляд https://localhost:2222/send/nurse для споживача медсестра, для споживача інсулінова помпа - https://localhost:1111/send/insulin_pump, https://localhost:4444/send/head_physician - головний лікар та https://localhost:3333/send/artificialRespirator - апарат штучного дихання.

Варто зазначити, що кожен компонент, який проводить відправку запиту, чекає у результаті на відповідь, яку аналізує та виконує потрібні дії у разі невдачі. При використанні такого способу взаємодії між компонентами програми, як запит/відповідь, можна бути впевненим, що відповідь надійде за короткий час, зазвичай менше ніж за секунду. До того ж, це дає змогу створювати власні вебхуки, які нестимуть інформацію у такому вигляді, яка потрібна розробнику. Ще однією перевагою використання вебхуків є той факт, що передача даних відбувається безпечно, за рахунок використання https, який створює захищений канал, куди зловмисникам не можливо потрапити. Https підтримує технологію шифрування TLS/SSL — це такий цифровий підпис сайту[9]. Перш ніж встановити захищене з'єднання, браузер запитує цей підпис і звертається до центру сертифікації, щоб підтвердити його валідність(легальність). Якщо він дійсний, то браузер вважає цей сайт безпечним та починає з ним обмін даними.

2.3 Робота з фоновими потоками, синхронізація потоків за допомогою семафорів

З основного потоку програми, можна створювати додаткові потоки для виконання потрібних дій. Процес розробки програм з використанням декількох потоків має назву багатопотокове програмування, а сам процес – багатопотоковість. У даній роботі ця функція додана задля швидшої обробки даних, адже заміри здоров'я мають бути передані споживачу якомога швидше – від цього може залежати чиєсь життя.

У мові програмування C# існують 2 види потоків: потоки переднього плану, та заднього плану (foreground та background threads). Потік переднього плану продовжує працювати до завершення своєї роботи, навіть якщо основний потік перестає працювати. Усі потоки, які ми створюємо є потоками переднього плану за замовчуванням. Однак, у розробці своєї програми, я застосувала потоки заднього плану (фонові), які є запущеними на фоні, обробляють дані по мірі їх надходження та працюватимуть допоки не буде команди зупинитись.

Максимальна кількість потоків, які може використовувати програма визначено як 3. Такий функціонал додано до проектів Dispatcher та Sender, оскільки лише вони працюють із чергами даних та вимагають швидкої їх обробки.

Робота з фоновими потоками здійснена через перевизначення методу ExecuteAsync() абстрактного класу BackgroundService, що міститься в просторі імен Microsoft.Extensions.Hosting. Це дає змогу працювати із довготривалими потоками, оскільки у звичайному використанні фонових потоків веб-сервер може випадково зупинити їх роботу, бо вона не є

прив'язаною до жодного поточного запиту. Таким чином код, який буде прописаний в методі ExecuteAsync() буде виконуватись на фоні в постійно робочому режимі. Такими функціями є діставання елементів із черг та відправка на інші сервіси у конвеєрі «Kafka». У класах-розширеннях, які мають назви SenderServiceCollectionExtension та DispatcherServiceCollectionExtension відбувається реєстрація класів, які реалізують BackgroundService. У цих місцях кожен фоновий сервіс додається тричі (максимальна кількість можливих потоків). У випадку розширення функціоналу та необхідності пришвидшення його роботи можна збільшити кількість потоків.

При роботі з багатопотоковістю у розробників завжди постає проблема із одночасним доступом до ресурсу кількома потоками. У ситуації даної системи декілька потоків можуть намагатися діставати однакові дані із черги. Таким чином один і той же пакет даних може бути оброблено і відправлено споживачу, що значно сповільнюватиме роботу програмного продукту. До того ж, така поведінка потоків може призвести до багатьох проблем із пам'ятю, а також такого поняття, як deadlock. Deadlock - це ситуація, коли виконання програми блокується, оскільки два або більше потоків чекають завершення роботи один одного.

Для вирішення такої проблеми було створено різноманітні інструменти синхронізації потоків, наприклад Mutex, Monitor, lock, ManualResetEvent, AutoResetEvent, Semaphore тощо. У своїй роботі я використала Semaphore, оскільки вони дозволяють більше, ніж одному потоку отримувати доступ до синхронізованої частини. У даному випадку семафор працює разом із чергою: як тільки елемент було додано до черги (виклик методу Enqueue()), у семафорі вивільняється місце для наступного елементу (виклик методу Release()). Далі

в той момент, коли користувачу знадобиться дістати елемент із черги (виклик методу Dequeue()), програма буде чекати (виклик методу WaitAsync()) додавання нового елементу до черги. Таким чином дані додаватимуться та діставатимуться із черги один за одним та одночасно мати доступ до черги зможе лише один із них.

У обох проектах, які використовують семафор, його об’єкт створюється із початковим значенням 0. Ця цифра відповідає за кількість потоків, які мають право доступу до ресурсу з самого початку, до виклику методів Release(), який збільшує це значення на 1 та WaitAsync(), що зменшує його на 1. Таким чином виключена можливість доступу до черги до того моменту, поки туди не буде додано хоча б одного елементу.

2.4 Організація черг даних та переформатовування даних

Заміри певних критеріїв здоров’я пацієнта додаються до черг у двох проектах конвеєру «Kafka» - Dispatcher та Sender.

Для роботи із чергами у обох проектах використано ConcurrentQueue<T> - потокобезпечну структуру даних, що працює за принципом «першим прийшов – першим пішом»(FIFO). Потокобезпечність такої черги проявляється в прийнятті кількох потоків читання та запису в чергу без необхідності явного блокування структури даних. У класах черг реалізовані асинхронні методи додавання до черги (EnqueueAsync()) та діставання з неї (DequeueAsync()), які працюють в парі із семафором.

У Dispatcher’і створюється та реєструється 6 черг: PulseQueue, BloodPressureQueue, GlucoseLevelQueue, BreathingRateQueue,

BreathingRateAndPulseQueue, AllMeasurementsQueue, що відповідають за передачу кожного пакету даних окремо через свою чергу – заміри пульсу, артеріального тиску, рівню глюкози в крові, частоти дихання, спільного пакету для частоти дихання і пульсу та спільного пакету з усіма показниками(кожного по одному) відповідно. Таке перегруповування відбувається задля відображення можливості використання даних у різний, потрібний користувачу, спосіб, що має певну схожість із ориганіальною Apache Kafka. Усі вищезазначені черги є спадкоємцями узагальненого класу MeasurementPackageQueue<T>. За рахунок використання такого підходу значно зменшується кількість програмного коду, адже черги працюють за однаковим принципом, тому їх реалізація може бути винесена в один узагальнений клас.

Принцип формування черг на стороні Dispatcher’a: у випадку критично низьких або високих показників, пакет даних негайно додається до однойменної черги та відправляється споживачу.

Якщо ж заміри виявились не загрозливими для життя пацієнта, то їх не потрібно надсилати у першу чергу, вони можуть почекати на надходження інших типів даних і бути сформованими у одну з черг BreathingRateAndPulseQueue та AllMeasurementsQueue. Ці показники потрібні споживачу лише задля фіксації архівних даних про кожного пацієнта. Саме тому їх можна надсилати не окремо, а у згрупованому вигляді. Таким чином зменшується кількість запитів та навантаження на черги, що може бути критичним для скорішої передачі життєво-загрозливих даних. Такий функціонал реалізовано за допомогою заповнення тимчасової таблиці бази даних TicketToQueue. Якщо дані виявилися не катастрофічними, то даний тікет кладеться у цю таблицю, де чекатиме витягування та врешті-решт

видалення. При надходженні нового тікету до Dispatcher'а відбувається перевірка, чи є в таблиці TicketToQueue заміри усіх інших типів, якщо так, то дістаються найдавніші, формуються у спільний пакет, додаються в чергу і видаляються із таблиці. Таким чином усі дані, як архівні, так і критичні буде сформовано у спільні пакети та відправлено через свою чергу на потрібного споживача.

Враховуючи, що черги Dispatcher'а реалізовані узагальнено, то є очевидним, що методи EnqueueAsync() та DequeueAsync() є також узагальненими і працюють із різними типами пакетів, що мають бути відправлені Sender'у.

Принцип формування черг на стороні Sender'а: у цього компонента створюється лише одна черга, яка працює із таким типом даних, як PostTicketTask. Ця модель може мати в собі будь-який із шести можливих в системі тип даних – пульсу, артеріального тиску, частоти дихання та рівня глюкози. Щоразу при надходженні нових даних на Sender, створюється екземпляр класу PostTicketTask на основі отриманого типу даних і одразу додається до черги.

У обох проектах процес діставання даних із черги відбувається безпосередньо перед їх відправкою – у класі ExecuteAsync() фонового сервісу, про які зазначалося раніше.

Варто відмітити, що Sender перед відправкою даних займається ще їх зміною у формат, який підтримується відповідними споживачами. Це реалізовано із використанням серіалізації даних із C# моделей у формати JSON та XML. Серіалізація – це процес перетворення будь-якого об'єкта в такі формати, як бінарний, JSON, XML тощо. В даній роботі було використано інструменти серіалізації JsonSerializer та XmlSerializer.

Отже, за допомогою налагодженої роботи семафорів, черг та сервісів фонових потоків реалізовується безперервна обробка та передача показників здоров’я всередині конвеєру «Kafka».

2.5 Використання баз даних

Компоненти частини програмного продукту, що входять до «Kafka» використовують спільну базу даних Hospital. Дані для підключення до неї знаходяться у файлі appsettings.json проектів Receiver та Dispatcher. Спільно, вони звертаються до таблиць Ticket та TicketToQueue для отримання, додавання та видалення даних, які там містяться. Кожна таблиця містить унікальний складений індекс, до якого входять колонки EntityId та EntityType.

Така реалізація забезпечує недопускання дублікатів записів за цими полями одночасно.

Усі можливі колонки обох таблиць є обов’язковими (NOT NULL), оскільки є необхідними компонентами запису про замір стану здоров’я людини.

У роботі із технологією EntityFramework розробник зобов’язаний знати про таку деталь, як відслідковування (tracking). Трекінг визначає, чи слід відстежувати інформацію про екземпляр сутності, коли його змінено[8]. Якщо сутність відстежується, то будь-які зміни, що були внесені до неї, зберігатимуться в базі даних після виклику методу SaveChanges(). Однак, варто зауважувати, що зміни, додані в трекінг можуть бути не внесені до бази даних через збій програми або помилку при виконанні методу SaveChanges(). Таким чином «зламані» дані, що все ще міститимуться в трекінгу при наступному зверненні до бази даних, будуть застосовані до записів в таблицях.

Така поведінка не бажана, тому у своєму проекті я реалізувала метод TryDetach(entity), що викликається при додаванні нових записів до бази даних. За допомогою зміни State заданої сутності на EntityState.Detached, цей елемент від'єднується від трекінгу, що виключає можливість застосування цих змін при наступному виклику SaveChanges().

Файли mirg.sql, що знаходяться в деяких проектах системи, містять в собі згенерований останньою (актуальною) міграцією sql-скрипт. Текст файлу можна скопіювати та вставити у будь-яку систему керування базами даних, що призведе до автоматичного створення там пустих таблиць із коректним вмістом індексів, колонок тощо. Це дає змогу не створювати всі ці елементи вручну, що може призвести до марної трати часу та помилок.

Кожен проект папки Consumers використовує своє окреме сховище, куди зберігає потрібні йому дані. Так було вирішено реалізувати задля відображення того факту, що споживачем може бути будь-яка система, веб-сайт чи інший програмний продукт, який є незалежним від “Kafka”.

Висновки

У сучасному світі кількість інформації у різних сферах настільки зросла, що потреба постійного її опрацювання стає все більш гострою. Саме така необхідність і спонукала розробників на створення систем, які працюють в режимі реального часу задля постійної та безперервної обробки новий даних. Такі системи можуть бути різних типів. Найголовнішим критерієм порівняння є жосткі та м'які системи, які відрізняються можливістю пропущення визначеного терміну спрацювання.

Одним із найбільш використовуваних у сучасній IT-індустрії засобів обробки даних являються так звані брокери повідомлень, за допомогою яких різні системи можуть спілкуватися один із одним і отримувати найактуальнішу та найновішу інформацію. У даній роботі було проаналізовано популярні брокери Apache Kafka, RabbitMQ та Redis. Найбільше уваги приділено інструменту Kafka, переглянуто його архітектуру, переваги та недоліки.

У практичній частині роботи малось на меті створення програмного продукту, що дає змогу обробляти нові дані у момент їх появи на прикладі показників здоров'я пацієнтів лікарні. Структура проекту складається з 3 основних компонентів: продюсери (апарати вимірювання запропонованих показників), споживачі (медичний персонал лікарні та апарати штучної підтримки стану пацієнта) та основний конвеєр передачі і обробки даних «Kafka».

Результатом роботи над практичною частиною стало досягнення поставленої мети: довершений програмний застосунок передачі замірів стану здоров'я людей в режимі реального часу.

Проект має багато можливостей для покращення та вдосконалення. Наприклад черги з даними для передачі споживачу можуть бути доповнені можливістю їх

розділення. Таким чином, робота по зберіганню та обробці повідомлень могла б бути розділеною між багатьма компонентами конвеєру. Також, можна впровадити інструмент реплікації в систему. За допомогою цього були б наявні кілька копій розподілених між серверами даних, що допомогло б підтримувати високу доступність на випадок, якщо один з серверів конвеєру вийде з ладу і буде недоступний для обробки запитів.

Список використаних джерел

1. Що таке NoSQL? [Електронний ресурс] https://www.oracle.com/cis/database/nosql/what-is-nosql/
2. HyperText Transfer Protocol. [Електронний ресурс] https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/
3. What is webhook and why you need it? [Електронний ресурс] https://www.mailjet.com/blog/news/what-is-webhook/
4. Dive into real-time data with Apache Kafka. [Електронний ресурс] https://softwaremill.com/real-time-data-apache-kafka/
5. Kafka Architecture and Its Fundamental Concepts. [Електронний ресурс] https://data-flair.training/blogs/kafka-architecture/
6. Use cases of Kafka. [Електронний ресурс] https://kafka.apache.org/uses
7. Advantages and Disadvantages of Kafka. [Електронний ресурс] https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/
8. Tracking vs No-Tracking Queries. [Електронний ресурс] https://riptutorial.com/ef-core-advanced-topics/learn/100010/tracking-vs-no-tracking-queries
9. What is HTTPS? [Електронний ресурс] https://www.cloudflare.com/learning/ssl/what-is-https/