УДК 004:656.614.2

Олещенко Л.М., к.т.н.,
Глінський В.В.

МІКРОСЕРВІСНА АРХІТЕКТУРА СИСТЕМИ ВІДЕО-ПОШУКУ ТРАНСПОРТНИХ ЗАСОБІВ, ЩО ПЕРЕБУВАЮТЬ У РОЗШУКУ У ЗВ`ЯЗКУ З ЇХ НЕЗАКОННИМ ЗАВОЛОДІННЯМ

Національний технічний університет України «КПІ ім.І.Сікорського»

oleshchenkoliubov@gmail.com,
glinskiy56@gmail.com

Проаналізовано існуючі програмні архітектурні рішення, їх переваги та недоліки для вирішення задачі побудови системи відео-пошуку транспортних засобів, що перебувають у розшуку. Розглядається розроблюваний програмний продукт з використанням мікросервісної архітектури. При розробці програмного продукту було враховано сучасність та ефективність, простоту використання, низький поріг входження, розгорнуту документацію та велику активну спільноту. Розглянуто переваги та недоліки архітектури програмного рішення

Ключові слова: JavaEE, Spring framework, мікросервісна архітектура, message broker, Enterprise Service Bus, API, API gateway, WebSocket, service discovery, polyglot persistence architecture, Docker, remote procedure call.

Вступ та постановка проблеми

У зв’язку із збільшенням кількості транспортних засобів (ТЗ), що знаходяться у розшуку, виникає необхідність у створенні ефективного програмного рішення, що дозволить в режимі реального часу виконувати збір та аналіз інформації про ТЗ, які знаходяться на дорогах загального користування. На основі цієї інформації має проводитись відеоаналіз і пошук збігів у відкритому наборі даних, який містить актуальні відомості про ТЗ, що перебувають у розшуку у зв’язку з їх незаконним заволодінням. Система повинна надавати публічний *API* (*application programming interface*), за допомогою якого користувачі зможуть отримувати доступ до статистичних даних та інформації про виявлені збіги. Наявність публічного *API* та доступ до відкритого вихідного коду дозволять зацікавленим особам розширювати та вдосконалювати програмне рішення. Публічне *API* може використовуватись правоохоронними органами чи приватними організаціями, які надають охоронні послуги чи займаються розшуком ТЗ. При використанні клієнт-серверної архітектури виникають труднощі у зв'язку з великим навантаженням на систему та відповідним зниженням її продуктивності, тому для вирішення даної проблеми необхідно визначити оптимальну технологію для передавання та обробки даних користувачів ТЗ.

Огляд існуючих програмних архітектурних рішень

При реалізації програмного рішення архітектура системи є ключовим фактором, що впливає на якість програмного продукту, можливість його масштабування, швидкість розробки. Розглянемо можливі архітектурні стилі реалізації серверної частини системи відео-пошуку ТЗ, що перебувають у розшуку у зв’язку з їх незаконним заволодінням [1-3].

Монолітна архітектура (об’єднання в одне ціле) є уніфікованою моделю розробки програмного забезпечення (ПЗ). Монолітне ПЗ проектується для досягнення повної автономності та самодостатності. Компоненти програми взаємопов'язані і взаємозалежні, на відміну від модульної архітектури, де компоненти слабо пов'язані. У монолітній

архітектурі (рис.1) кожен компонент повинен бути присутнім для компіляції або виконання коду. Дя зміни будь-якого компоненту необхідно вносити зміни до вихідного коду всього додатку та проводити його компіляцію. У модульній системі окремі модулі можуть бути змінені без впливу на інші частини ПЗ. До переваг монолітної архітектури відносять простоту розробки, тестування та розгортання. Монолітні системи, як правило, мають більшу пропускну здатність, ніж модульні альтернативи.

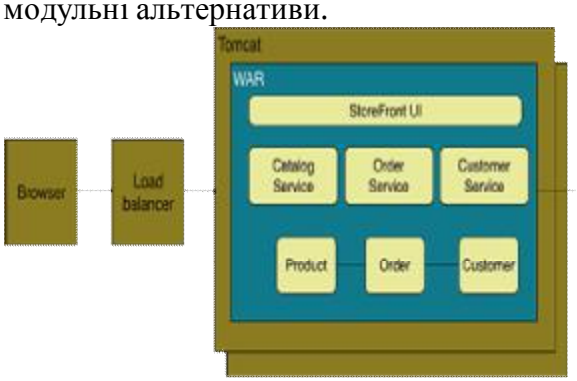

Рис. 1. Традиційна монолітна архітектура вебдодатків

Багаторівнева архітектура являє собою клієнт-серверну архітектуру, в якій функції презентації, обробки і управління даними фізично розділені. Багаторівнева архітектура застосовується з використанням трьох рівнів: представлення, бізнес рівень та рівень доступу до даних (рис.2).

Рівень представлення (фронт-енд рівень, або рівень інтерфейсу користувача) відповідає за створення та відображення інтерфейсу, обробляє дії користувача. Відображає дані, отримані з бізнес рівня.

Рівень сервісів або рівень вебсервісів вдповідає за надання *API* вебсервісів. Дані, отримані з бізнес рівня, зазвичай представляються у форматах *XML* та *JSON*.

Бізнес рівень (доменний рівень) відповідає за виконання усієї бізнес логіки додатку. Складається з доменної моделі та доменних сервісів. *Доменна модель* є системою абстракцій, що описує окремі аспекти предметної області та може бути використана для вирішення проблем, пов'язаних з цією предметною областю.

Доменний сервіс безпосередньо виконує необхідну бізнес логіку.

Рівень доступу до даних відповідає за доступ до БД, виклику веб-сервісів.

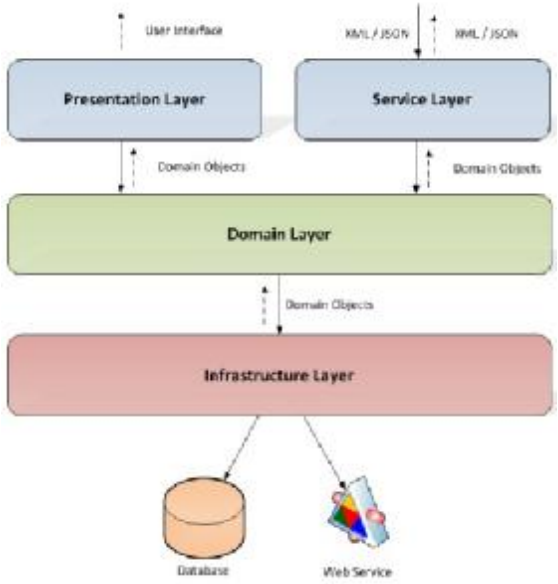

Рис. 2. Трьохрівнева архітектура

Сервіс-орієнтована архітектура (*service-oriented architecture*, *SOA*) являє собою стиль розробки ПЗ, де за допомогою певних компонентів сервіси надаються до інших компонентів програми, використовуючи протоколи передачі даних по мережі (рис.3). Сервіс є дискретним блоком функціональних можливостей, реалізація яких може бути змінена незалежно від інших сервісів. Сервіс представляє собою бізнес логіку з певним результатом, є самодостатнім, являє собою чорну скриньку для його користувачів та може складатися з інших сервісів. Розглянемо особливості сервісорієнтованої архітектури.

Сервіси повинні дотримуватись стандарту комунікації з іншими сервісами. Ці стандарти визначені в одному або декількох документах опису сервісів.

Автономність (аспект слабкого зв'язку) - відносини між сервісами мінімізуються, їм відомо лише про взаємне існування.

Прозорість розташування сервісу – сервіси можуть бути викликані з будь-якого місця в мережі.

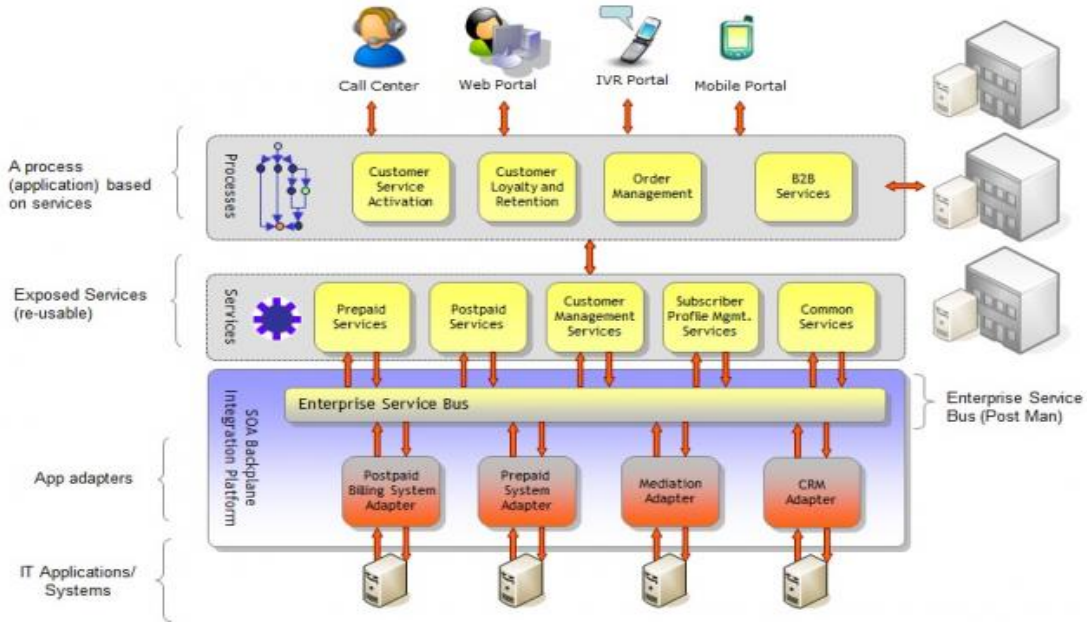

Рис. 3. Сервіс-орієнтована архітектура

Довговічність - сервіси мають бути розробленні з метою довготривалого існування. Слід уникати таких змін, що призводять до необхідності змінювати реалізацію споживача даного сервіса.

Абстракція - сервіси працюють як чорні скриньки, тобто їх внутрішня логіка прихована від користувачів.

Відмова від збереження стану - сервіси не зберігають свого стану, тобто вони повертають необхідне значення, створюють виключну ситуацію, зводячи використання ресурсів до мінімуму.

Нормалізація - сервіси розкладаються, або нормалізуються з метою мінімізації їх надмірності та оптимізації продуктивності.

Здатність до компонування - сервіси можуть бути використані для створення нових сервісів.

Повторне використання - логіка розділена на декілька сервісів з метою повторного використання коду.

SOA дозволяє підприємствам швидко і ефективно реагувати на зміни ринку.

До недоліків сервіс-орієнтованої архітектури слід віднести відсутність чіткої специфікації.

Мікросервісна архітектура має значні переваги, особливо, коли мова йде про гнучку розробку і реалізацію комплексних корпоративних додатків. Головним завданням мікросервісної архітектури є усунення надмірної складності великих корпоративних систем. *Amazon*, *eBay* і *Netflix* вирішили цю проблему шляхом впровадження мікросервісної архітектури . Замість побудови громіздкого монолітного додатку додаток розбивається на невеликі, з'єднані між собою сервіси. Сервіс реалізує набір різних функцій або функціональних можливостей, таких як управління замовленнями, клієнтами тощо. Кожен мікросервіс являє собою мініодаток, який має свою архітектуру, що складається з бізнес-логіки з різними адаптерами. Деякі мікросервіси надають *API*, який використовується іншими мікросервісами або клієнтами програми. Інші мікросервіси можуть реалізувати веб-інтерфейс. Після розгортання системи кожен екземпляр представляє собою окрему віртуальну машину у хмарі, або ж окремий *Docker* контейнер. Веб-додаток

розбивається на більш прості веб-додатки, що полегшує розгортання системи. Сервіси можуть взаємодіяти між собою, використовуючи як *REST API*, так і асинхронну комунікацію на основі повідомлень. Мікросервісна архітектура відповідає масштабуванню по осі *Y* шляхом декомпозиції на сервіси. Вісь *X* відповідає масштабуванню шляхом запуску кількох копій додатку з використанням балансувальника навантаження. Масштабування по *Z*-осі забезпечується шляхом поділу даних, наприклад, первинний ключ рядка або ідентифікатор сутності використовуються для маршрутизації запиту до конкретного сервера (рис.4,5).

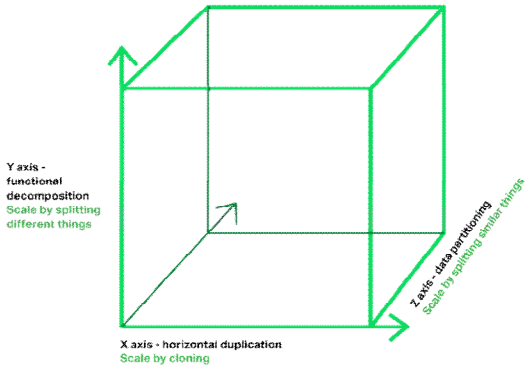

Рис. 4. “Куб масштабування”

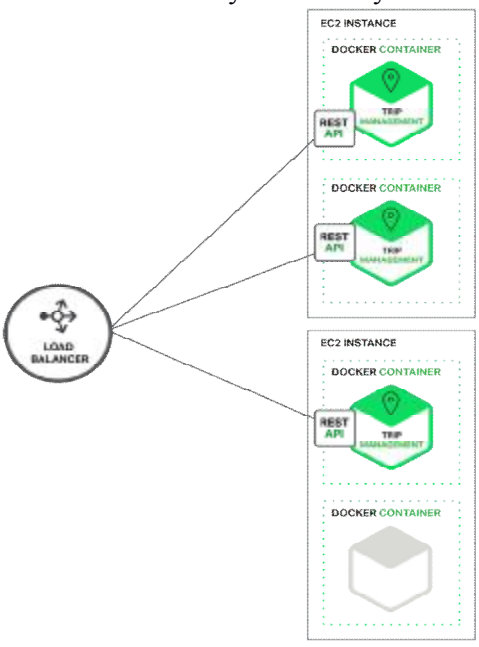

Рис. 5. Масштабування по осі *X* шляхом запуску декількох копій додатку з використанням балансувальника навантаження

Мікросервісна архітектура впливає на комунікацію між додатком і базою даних (БД). Замість того, щоб ділити одну схему БД з іншими сервісами, кожен сервіс має свою власну схему БД (рис.6).

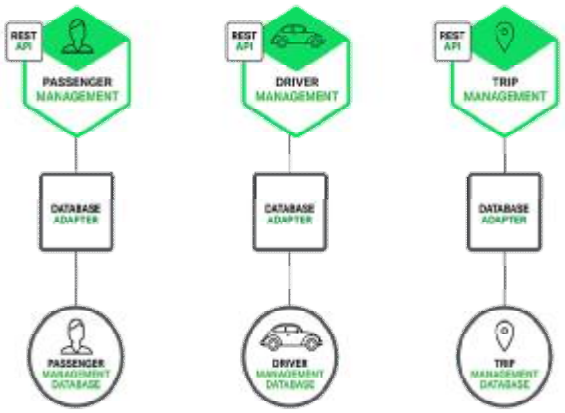

Рис. 6. Архітектура БД при використанні мікросервісної архітектури

Кожен сервіс використовує тип БД, який найкраще підходить для його потреб. Наприклад, можливе використання одними сервісами реляційних БД, а іншими – нереляційних. Мікросервісну архітектуру можна вважати її копією сервіс-орієнтованої архітектури без комерціалізації, складних протоколів комунікації і сервісної шини підприємства (*Enterprise Service Bus*). Мікросервіси використовують більш прості протоколи, такі як *REST* [4].

Мікросервісна архітектура має ряд важливих переваг. Система розкладається на кілька простіших додатків, у той час як загальний обсяг функціональності не змінюється. При цьому окремі сервіси набагато легше розробляти та підтримувати, ніж моноліт. Мікросервісна архітектура дозволяє кожному з сервісів розроблюватись незалежно один від одного, різними командами на окремому сервісі. Розробники можуть вільно вибирати необхідні технології. Мікросервісна архітектура дозволяє сервісам розгортатися незалежно один від одного, що дозволяє змінювати окрему функціональність без необхідності зупинення роботи усієї системи. Мікросервісна архітектура дозволяє масштабувати сервіси незалежно один від

одного. Мікросервісний додаток має розподілену архітектуру БД. Розробники повинні вибрати механізм зв'язку між сервісами, наприклад, на основі *RPC(remote procedure call*). Нові бізнес вимоги часто змушують проводити зміни одразу в декількох схемах БД.

Побудова системи відеопошуку ТЗ з використанням мікросервісної архітектури

Сервіс профілів користувачів (Account service) відповідає за збереження інформації про профілі користувачів та їх налаштування. Використовується при реєстрації, аутентифікації та авторизації користувачів. Має власну БД, а для взаємодії з іншими сервісами надає *REST API*.

Статистичний сервіс (Statistic service) проводить розрахунок основних статистичних параметрів облікового запису користувача, що базуються на кількості унікальних розпізнаних номерів. Аналізує дані про ТЗ, що знаходяться у розшуку. Надає *REST API* для використання іншими сервісами.

Сервіс розпізнавання зображень (Recognition service) відповідає за розпізнавання номерних знаків на зображеннях. Зображення завантажуються за допомогою *API* з використанням протоколу *WebSocket*, який надає повнодуплексний канал зв'язку через одне *TCP*-з'єднання. Використання *WebSocket* дозволяє клієнту відправляти нове зображення на обробку, навіть якщо результати розпізнавання попередніх зображень ще не доступні.

Розпізнавання номерів відбувається за допомогою бібліотеки автоматичного розпізнавання номерних знаків *OpenALPR*. Розпізнавання номерних знаків на зображенні потребує в середньому 20-70 мс на обробку одного зображення. Для вирішення проблеми великої кількості клієнтів та запитів можливе використання посередників повідомлень (*message broker*), таких як *Kafka, ActiveMQ* або ж *RabbitMQ*, для організації повідомлень клієнтів у чергу. Кожне повідомлення містить зображення, що має бути оброблене сервісом розпізнавання зображень. Використання черги дозволяє запускати декілька копій сервісу, що зможуть працювати незалежно один від одного [5,6].

Головним завданням *сервіса номерів ТЗ* (*Number plate service*) є отримання актуальних даних про ТЗ, що знаходяться у розшуку. Сервіс надає зручний та простий *REST API* для перевірки номерних знаків на наявність у БД розшукуваних ТЗ. Приклад запиту до сервісу та його відповіді:

```
POST /number-plate-service HTTP/1.1
Host: localhost
Content-Type: application/json
[     {"number_plate": "AA 2569 BB"},
      {"number_plate": "BI 7153 BE"}     ]
HTTP/1.1 200 OK
Content-Type:
application/json;charset=UTF-8
[  {
  "department": "ВІДДІЛЕННЯ ПОЛІЦІЇ
№1 КРЕМЕНЧУЦЬКОГО ВІДДІЛУ ГУНП
В   ПОЛТАВСЬКІЙ ОБЛАСТІ",
  "chassis_number": "",
  "body_number": "XTA21074031814325",
  "number_plate": "BI7153BE",
  "model": "ВАЗ - 2107",
  "color": "ЧЕРВОНИЙ"
 } ]
```

Отримання актуальних даних про ТЗ, що знаходяться розшуку, відбувається за допомогою порталу *data.gov.ua*. Так як портал не підтримує методів надання повідомлень про оновлення даних, таких як *Webhook*, сервіс номерів ТЗ перевіряє версію набору даних автоматично через фіксований проміжок часу.

При публікації нової версії набору даних, сервіс отримує оновлені дані та проводить їх аналіз, заносячи необхідні зміни до власної БД (табл.1).

Сервіс номерів ТЗ використовує СУБД *MongoDB*, яка забезпечує строгу узгодженість даних, високу продуктивність та здатність до масштабування. У даній БД використовуються такі поля сервісу номерів ТЗ:

"*id*" – ідентифікатор запису;

"*class*" – стандартний атрибут при використанні *Sprind Data MongoDB*, що вказує на відповідність запису екземпляру *Java* класу;

"*chassis number*" – номер шасі ТЗ;

"*body_number*" – номер кузова ТЗ;

"*number_plate*" – номерний знак ТЗ;

"*model*" – модель ТЗ;

"*color*" – колір ТЗ

"*registered_as_ wanted*" – дата реєстрації ТЗ як такого, що знаходиться у розшуку;

"*mvs_id*" – Ідентифікатор запису у наборі даних МВС;

"*department*" – відділ поліції, який зареєстрував ТЗ як такий, що знаходиться у розшуку;

"*data_set_version_id*" – ідентифікатор версії набору даних.

Дані зберігаються у форматі *JSON*.

Сервіс надання повідомлень (*Notification service*) зберігає інформацію користувачів та налаштування повідомлень. Спеціальний компонент сервісу збирає дані з інших сервісів і відправляє повідомлення клієнтам системи.

Сервіс веб-інтерфейсу (*Web UI service*) забезпечує взаємодію з кожним сервісом та об'єднання їх у систему відео-пошуку ТЗ.

Для уникнення навантаженя на рівні браузера клієнта, сервіс веб-інтерфейсу взаємодіє з кожним компонентом, проводить ресурсоємні обчислення та виступає у ролі фасада для усіх інших сервісів, надаючи веб-інтерфейс для кінцевого користувача.

Розглянемо зв'язки між сервісами системи відео-пошуку ТЗ, що знаходяться у розшуку (рис.7).

1. Користувач проходить авторизацію та аутентифікацію, після цього сервіс веб-інтерфейсу звертається до сервісу профілів користувачів.
2. Користувач завантажує відео-файл на обробку, використовуючи додаток. Клієнт відповідає за розбиття відео-файлу на кадри, які потім потрапляють на обробку до сервісу розпізнавання зображень.

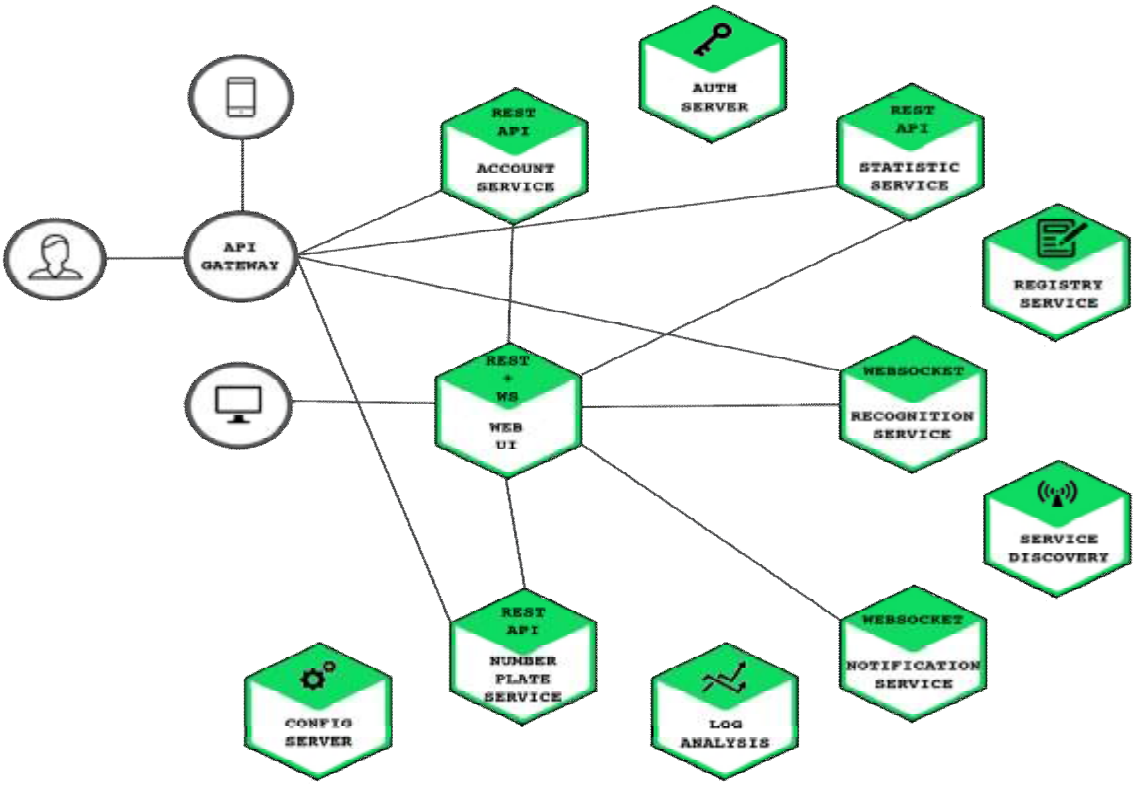

Рис. 7. Функціональні та інфраструктурні сервіси системи відео-пошуку ТЗ

3. Після отримання результатів розпізнавання зображень відбувається перевірка знайдених номерів на наявність у БД розшукуваних ТЗ.

4. У випадку знаходження номерного знаку у БД розшукуваних ТЗ користувач отримує миттєве повідомлення з детальною інформацією про ТЗ.

5. Отримані результати розпізнавання, а також результати пошуку у БД розшукуваних ТЗ, потрапляють до статистичного сервісу, де проводиться їх аналіз, обчислюється кількість унікальних розпізнаних номерів, яка впливає на оцінку активності користувача.

6. Зв'язок між сервісом веб-інтерфейсу та сервісом надання повідомлень забезпечує можливість користувача встановити налаштування отримання повідомлень, використовуючи веб-інтерфейс системи.

Висновки

Здійснивши аналіз сучасних архітектурних рішень для задачі побудови системи відео-пошуку ТЗ, що знаходяться у розшуку у зв'язку з їх незаконним заволодінням, було обрано мікросервісну архітектуру, оскільки при використанні клієнт-серверної архітектури виникли труднощі у зв'язку з великим навантаженням на систему та відповідним зниженням її продуктивності.

Мікросервісна архітектура допускає масштабування окремих частин системи, що дозволяє справлятися з навантаженням на систему та обслуговувати велику кількість клієнтів одночасно, що є актуальним при завантаженні багатьма клієнтами відео-файлів руху ТЗ. Тестування мікросервісного додатку виявилося складнішим, аніж тестування моноліту.

Публічне *API* розробленого додатку дозволяє використовувати його пересічним користувачам ТЗ та організаціям, які займаються розшуком ТЗ, для оптимізації пошуку ТЗ.

Список літератури

1. Microservice Architecture Aligning Principles, Practices, and Culture, by Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen, O'Reilly Media, July 2016.

2. Building Microservices Designing Fine-Grained Systems, by Sam Newman, O'Reilly Media, February 2015.

3. Production-Ready Microservices Building Standardized Systems Across an Engineering Organization, by Susan J. Fowler, O'Reilly Media, November 2016.

4. Create RESTful Web services with Java technology // [Електронний ресурс] – Режим доступу: http://www.ibm.com/developerworks/ webservices/library/wa-jaxrs.

5. Christian Wolf , Jean-Michel Jolion and Francoise Chassaing. Text Localization, Enhancement and Binarization in Multimedia Documents // [Електронний ресурс] – Режим доступу: http://liris.cnrs.fr/christian.wolf/papers/icpr2002v.pdf

6. OpenALPR, open source Automatic License Plate Recognition library, documentation // [Електронний ресурс] – Режим доступу: http://doc.openalpr.com.

Статтю подано до редакції 29.04.2017