NICOLAUS COPERNICUS UNIVERSITY IN TORUŃ

Pedagogy and Psychology of Sport. eISSN 2450-6605.

Journal Home Page
https://apcz.umk.pl/PPS/index

DOBROVOLSKYI, Yurii, POPOVYCH, Igor, TANASHCHYSHENA, Inessa and HANZHELO, Mariia. Decomposition of monoliths of a project into microservices on the example of a web application for a medical center. Pedagogy and Psychology of Sport. 2026;31:71428. eISSN 2450-6605. https://doi.org/10.12775/PPS.2026.31.71428

The authors declare that there is no conflict of interests regarding the publication of this paper.
Received: 17.03.2026. Revised: 14.03.2026. Accepted: 14.04.2026. Published: 01.05.2026.

DECOMPOSITION OF MONOLITHS OF A PROJECT INTO MICROSERVICES ON THE EXAMPLE OF A WEB APPLICATION FOR A MEDICAL CENTER

Yurii Dobrovolskyi[1], Igor Popovych[2], Inessa Tanashchyshena[1], Mariia Hanzhelo[1]

1. Yuriy Fedkovych Chernivtsi National University, Kotsyubynsky 2, Chernivtsi, 58012, Ukraine
2. Kozyavkin International Rehabilitation Clinic, Truskavets', Ukraine

y.dobrovolsky@chnu.edu.ua (Yurii Dobrovolskyi),
i.tanashchyshena.y@chnu.edu.ua (Inessa Tanashchyshena)
m.hanzhelo@chnu.edu.ua (Mariia Hanzhelo)
i.l.popovych@gmail.com (Igor Popovych)

Abstract

The technology of splitting a monolith into independent microservices, which resulted in the development of a web application for a medical institution. It is shown that for “light” services it is worth using frameworks such as Micronaut or Express.js, and for “heavy” services it is worth using SOA architecture. Also, microservice architecture allows you to more clearly and visually localize/monitor “hot nodes” and thus develop recommendations. Kubernetes orchestration allows you to quickly manage the main processes that block secondary ones. A successful solution was the use of Security as a separate microservice. The created web application allows you to perform parallel processes by different authorized users without losing data processing speed. For example, for 100 simultaneous users, the response time of any operation in 99% of cases should not exceed 4 seconds when using MSA with standard server settings. The probability of an error does not exceed 0.1% of all cases.

Keywords. MSA, SpringCloud, monolith, decomposition, performance, optimization, medical institution, web application, generation.

ДЕКОМПОЗИЦІЯ МОНОЛІТ ПРОЕКТУ НА МІКРОСЕРВІСИ НА ПРИТКЛАДІ ВЕБ-ДОДАТКУ ДЛЯ МЕДИЧНОГО ЦЕНТРУ

Юрій Добровольський[1], Ігор Попович[2], Інеса Танащтшина[1], Марина Ганжело[1]

1. Чернівецький Національний Університет ім. Юрія Федьковича, Коцюбінського 2, Чернівці, 58012, Україна
2. Міжнародна реабілітаційна клініка Козявкіна, Трускавець, Україна

y.dobrovolsky@chnu.edu.ua (Yurii Dobrovolskyi),
i.tanashchyshena.y@chnu.edu.ua (Inessa Tanashchyshena)
m.hanzhelo@chnu.edu.ua (Mariia Hanzhelo)
prokhorov.mykola@chnu.edu.ua (Igor Popovych)

Реферат

Проведено дослідження технології розщеплення моноліту на незалежні мікросервіси, результатом якого стало розробка веб-додатку для медичного закладу. Показано, що для “легких“ сервісів варто використовувати фреймворки типу Micronaut або Express.js, а для “тяжких” сервісів варто використовувати SOA архітектуру. Також мікросервісна архітектура дозволяє більш чітко і наглядно локалізувати/моніторити “гарячі вузли” і таким чином виробити рекомендації. Оркестрування Kubernetes

дозволяє оперативно керувати головними процесами, блокуючі другорядні. Вдалим рішенням стало застосування Security як окремий мікросервіс. Створений веб-додаток дозволяє здійснювати паралельні процеси різними авторизованими користувачами без втрати швидкості обробки даних. Наприклад, для 100 одночасних користувачів час відповіді будь-якої операції у 99% випадках не повинен перевищувати 4-х секунд при використанні MSA з стандартними серверними налаштуваннями. Ймовірність помилки не перевищує 0.1 % від усіх випадків.

Ключові слова. MSA, SpringCloud, моноліт, декомпозиція, продуктивність, оптимізація, медзаклад, веб-додаток, генерація.

Вступ.

Актуальним завданням, яке постійно стоїть перед людством, це збереження особистих даних користувачів клієнтів, які користуються різноманітними інформаційними ресурсами та веб-додатками. Це стосується і медичних закладів, які використовують різноманітні інформаційні системи для обслуговування пацієнтів, починаючи від первинної реєстрації і, аж до отримання результатів лікування і повідомлень про черговий професійний огляд. Архітектура переважної більшості таких інформаційних систем побудована на основі моноліту, або, інакше, монолітної архітектури, яка передбачає, що усі складові програмного завбезпечення, що створюють інформаційну систему, містяться в одному програмному блоці. Логічно нероздільному. З другого боку, сучасні компанії IT індустрії підтримують стійкість і масштабованість своїх програмних продуктів, спираючись на мікросервісні додатки (MSA). Через це популярність мікросервісів зростає, а моноліт стає чимось архаїчним, прикладом хибної архітектури, яка не піддається масштабуванню у відповідності до сучасних потреб бізнесу.

Раніше ми розглядали теоретичні засади процесу декомпозиції веб-додатка монолітної архітектури на окремі мікросервісні додатки, які об’єднані у аналогічну систему на основі
технології SpringCloud.

Тепер ми розглядаємо питання практичного застосування наших рекомендації на прикладі веб-додатку для медичного закладу, а саме - для роботи реєстратури.

Метою дослідження є розробка технології розщеплення моноліту на незалежні мікросервіси на прикладі веб-додатку для медичного закладу, а саме - для роботи медичного реєстратури.

Досягнення поставленої мети виконується вирішенням таких завдань:

- розробка архітектури веб-додатку для медичного центру на основі незалежних мікросервісів;

- тестування та порівняння параметрів обробки даних і ресурсів, що використовувалися;

- розробка висновків щодо доцільності використання різних технологій.

Методи дослідження.

Системний аналіз для оцінки існуючих технологій декомпозиції монолітної моделі; інструментальні методи розробки, такі як використання технологій Java версія correto 17.06, фреймворк SpringBoot 3.12; середовище розробки IntelliJ IDEA 2023.3.1 (Ultimate Edition), операційна система Ubuntu 20.04.6 LTS; методи інтеграції даних для об'єднання інформації з різних API; методи тестування для перевірки функціональності та надійності системи.

Огляд аналогів.

Існують різна методи декомпозиції монолітної моделі. Наприклад, один із відомих шаблонів – MVC, який розбиває програму на рівноправні та логічні компоненти [1]. У роботі [2] огляд теоретичних засад, основних властивостей та переваг мікросервісної технології до будови архітектур. Розглянуто також питання перетворення монолітів у мікросервіси [3], а у [4] приділяється увага патернам проектування, керування даними, цілісності в децентралізованому просторі. Організація розподілених систем збереження інформації досліджено у роботі [5], де розглянуто технології реплікації, шардування і архітектурні особливості федеративних баз даних. У [1, 6] наголошується на забезпеченні високих показників відмовостійкості, та стабільного доступу до ресурсів із забезпечення гнучкого горизонтального масштабування. У [7] продемонстровано алгоритм трансформації монолітної структури в мікросервісний формат, який містить децентралізовану базу даних. Також існують інструкції щодо розділення моноліту на мікросервіси, зокрема у [8]. Але усі вище перелічені дослідження та рекомендації не передбачають особливості реалізації конкретного проекту.

Результати дослідження.

Дослідницька задача полягала у встановленні доцільності розбиття моноліту на мікросервіси або вдосконалення існуючого моноліту. Для чистоти експерименту вирішено розбити моноліт на атомарні сервіси і дослідити характеристики. Які переваги дає такий підхід, заздалегідь відомо. А от недоліки можуть бути несподіваними.

Для реалізації була обрана платформа Spring Cloud, яка поєднує риси безсерверної архітектури та пропонує службові фреймворки як мікросервісні модулі. Мова програмування Java версія correto 17.06. Фреймворк SpringBoot 3.12. Середовище розробки IntelliJ IDEA 2023.3.1 (Ultimate Edition). Операційна система Ubuntu 20.04.6 LTS.

1. Монолітний веб-додаток — медична клініка

1.1. Структура і функціонал клініки.

Медичний центр – це профільна клініка, що займається лікуванням суглобів, захворювань хребта, реабілітацією опорно-рухового апарату, корекцією фігури, спортивною реабілітацією, та лікуванням військовослужбовців після поранень.

Центр лікування хребта та суглобів називався спочатку як “Центр народної медицини”, а пізніше перейменований у медичний центр. Спершу в ньому надавались послуги мануальної терапії та масажу.

Наразі з’явились додаткові методи лікування та збільшився штат працівників.

Організаційна структура:
- реєстратура;
- надання медичних послуг;
- бухгалтерія;
- госп. відділ;
- готель;
- ідальня;

Чисельні показники:
- 32 процедури;
- 45 процедуро-місць;
- 16 кабінетів;
- 25 лікарів;
- до 150 пацієнтів щодня;
- 10 тис послуг на місяць;
- 1.5 млн дол\місяць.

1.2. Задачі функціоналу веб-додатку

З самого початку функціонал планувався як регулятор електронної черги. Кожен пацієнт, як правило, мав пройти 5 - 6 процедур за день. Для врегулювання та оптимізації проходження всіх черг і процедур був спроектований, спланований та імплементований веб-додаток на такому стеку технологій:

- Java 8 версії 1.8.0 252
- MongoDb версії 3.6
- Angular 6.0
- SpringBoot 2.12.

На рисунку 1 представлено інтерактивне робоче місце регістратури. Електронна черга.

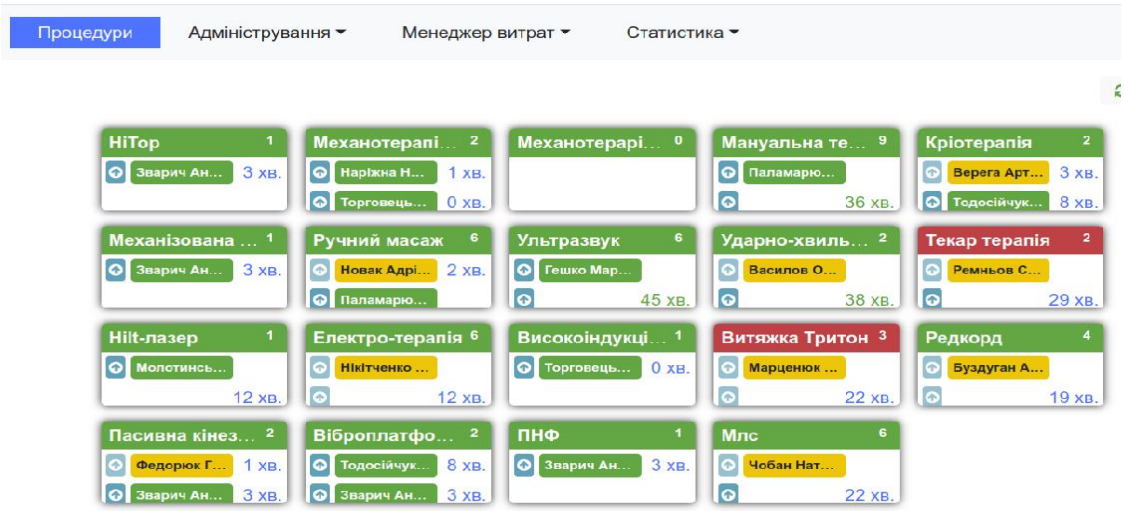

Рис. 1. – Інтерактивне робоче місце регістратури. Електронна черга.

На рисунку 1 чітко видно чергу в кожний кабінет. Видно, хто зараз проходить процедуру, чи є вільні місця. Черга рухається сама. Оновлення екрану кожні 15 сек.

Після успішного тестування та піврічної роботи функціонал було доповнено. Тепер на нього покладені наступні задачі:

- формування електронної черги:
- діагностика;
- формування проходження маршруту;
- облік процедур пацієнта;
- розрахунок пацієнта;
- розрахунок пацієнта;
- облік лікарів;
- система бонусів та надбавок;
- облік пацієнтів;
- облік процедур;
- зарплата;
- готель:
- господарські витрати;
- статистика;
- робоче місця лікаря;
- персональний кабінет пацієнта;
- безпека додатку;
- створення бекапів;
- звіти;
- виписки пацієнтам;
- мобільна версія

Як видно, за кілька років коректно продумана монолітна архітектура здобула ще десяток допоміжних функціоналів.

На рисунку 2. наведено інтерактивне робоче місце регістратури згідного оновленого програмного забезпечення.

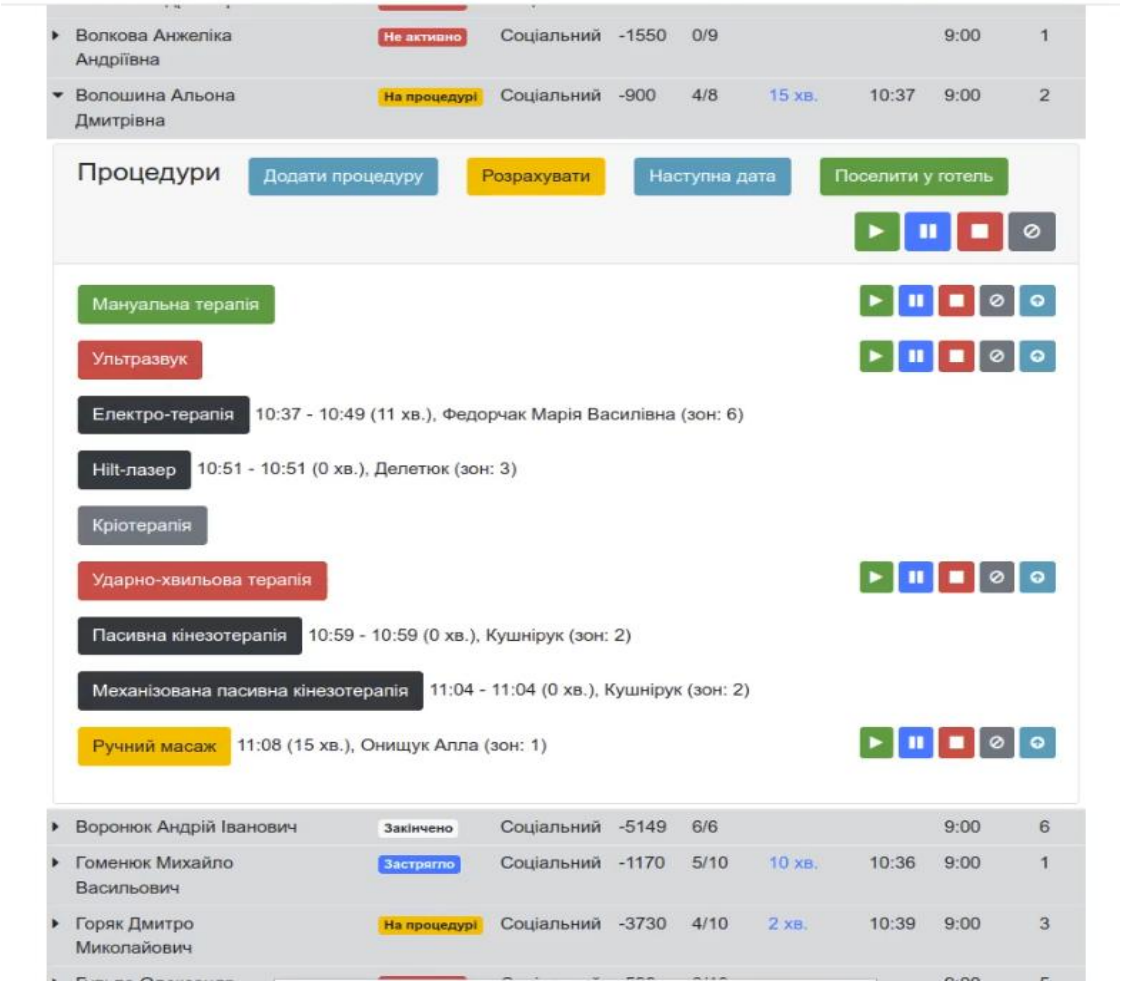

Рис. 2. Інтерактивне робоче місце регістратури.

На рисунку 2 видно облік пацієнтів на сьогодні. Відображається загальна кількість пацієнтів. Запланована кількість. Карта і маршрут проходження процедур кожним пацієнтом.

В межах короткого екскурсу не має сенсу продемонструвати інтерфейси робочого місці лікаря (мобільний додатка), діагноста (веб-додаток), бухгалтера, головного лікаря та панель системного адміністратора. Вони не менш деталізовані і споживають великий ресурс. Все це дає змогу зрозуміти , яким додатковим функціоналом поповнився первісна архітектура і необхідність зробити кардинальну переробку.

2. Серверна частина

Такий складний та ресурсоємний функціонал вимагав потужної серверної підтримки. Після кількох років міграцій командним рішенням була затверджена наступна конфігурація сервера: 4 core, 3.40 GHz; 8M Cache, LGA1200; 16 Gb RAM; 500Gb SSD; Debian 8.11 Jessie; Network 100 Mb/s.

2.1. Серверний моноліт-додаток N-tier архітектури

Додаток створений за допомогою фреймворку SpringBoot 2.12 згідно стандартної архітектури N-tier наведений на рисунку 3.

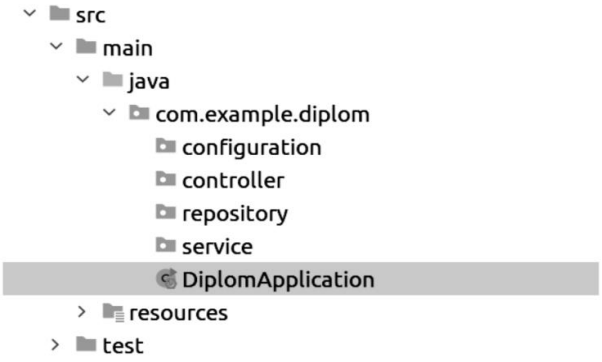

Рис. 3. Зразок стандартної архітектури N-tier

2.2. Діаграма класів (часткова)

Всього класів моделей разом з ENUMами налічується 52. Нема змоги всіх їх відобразити на одній діаграмі то ж приведемо частково, просто для уявлення розміру хаосу в архітектурі (рис. 4).

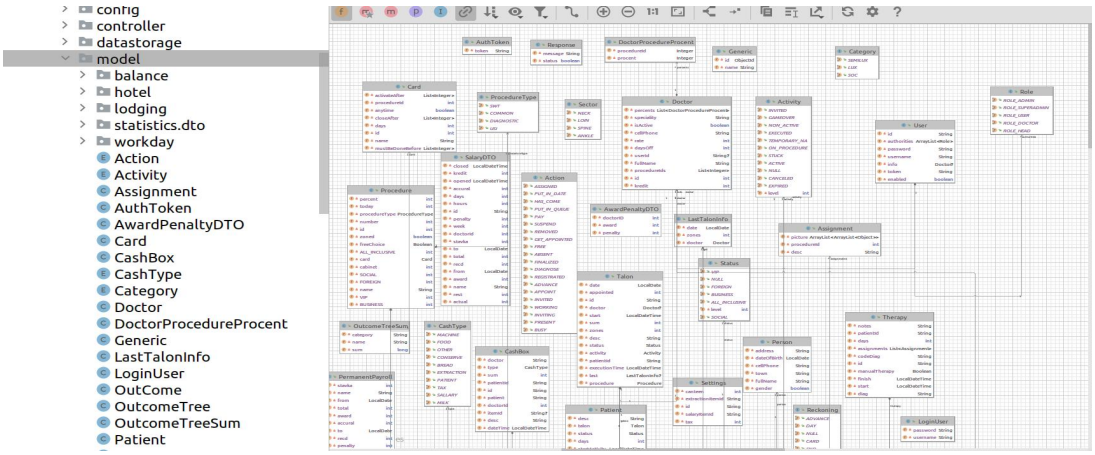

Рис. 4. Часткова діаграма класів додатка.

Особливий внесок у великий хаос вносили класи, які робились на майбутнє, але так і не були застосовані. Наприклад, клас по УЗД-діагностиці. Апарат зламався – функціонал втратив актуальність. Проте загальний клас Diagnose все ще тримає на нього посилання, що займає зайву оперативну пам’ять та має вплив на швидкодію.

3. База даних

З самого початку було прийняте коректне рішення використовувати noSQL базу даних mongoDb. Передбачалось , що будуть кардинальні зміни у алгоритмах і архітектурі, то ж SQL база даних з її жорсткими зв’язками не дозволить робити зміни швидким чином. База даних mongoDb не має зовсім ніяких зовнішніх ключів і , що дуже важливо для нашого випадку, не має жорсткої архітектури самої таблиці (колекції).

На рисунку 5 наведений перелік таблиць (колекцій) у базі даних.

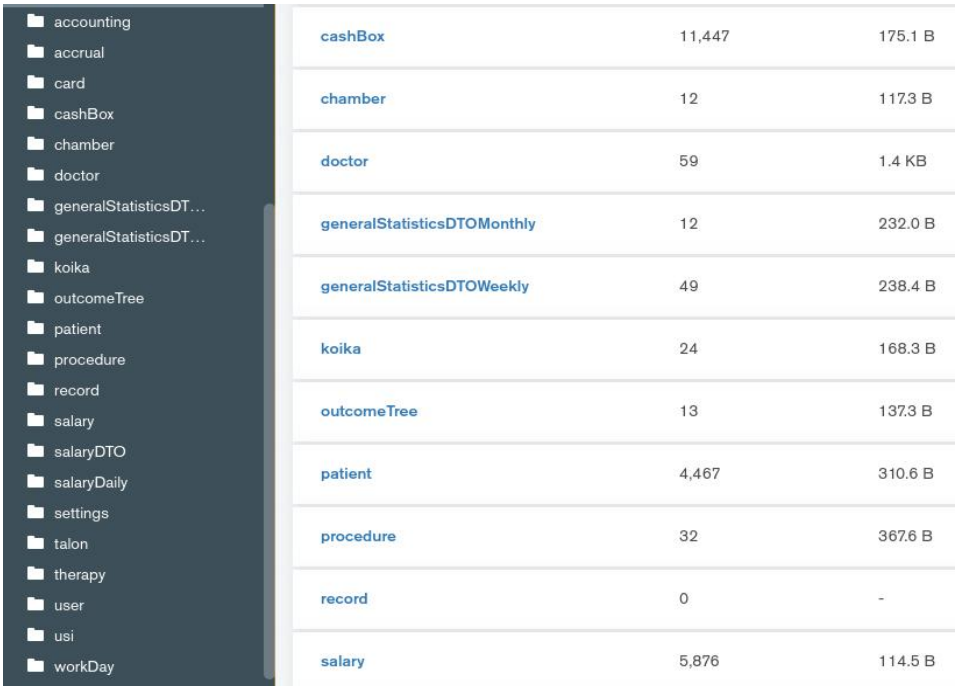

Рис. 5. Перелік таблиць (колекцій) у базі даних

Проблематика функціоналу веб-додатку.

Результати тестування створеного веб-додатку показали його недоліки, які полягали у наступному. Незадовільна швидкодія – відповідь на запит може тривати до 15 сек; громіздка та заплутана архітектура; потовщали об’єкти (рентген фотографії); узгодження з вимогами E-HEALTH; безпека (двічі зламували); інтеграція з платіжною системою; цілісність даних (бекапи); безпека особистих даних. У зв’язку з отриманими результатами було прийнято рішення про декомпозицію монолітної архітектури проекту для усунення недоліків.

4. Об’єм роботи

У проекті 28 сервісів,які складені у монолітну архитектуру. Деякі «гарячі» викликаються 4 рази на хвилин (наприклад, перегенерація черги), деякі раз на місяць (наприклад, зарплата). Кожен сервіс має свою таблицю (колекцію) у базі даних, яка згодом при декомпозиції стане окремою базою. Немає змісту всі недоробки попереднього додатка перетягувати у нову архітектуру.

То ж має сенс оптимізувати на відрефакторити кожен сервіс перед вилученням, як це показано на рисунку 6.

Рис. 6. Об’єм роботи: 28 потенційних мікросервісів

4.1. Базова конфігурація Spring Cloud

Блок схема типової мікросервісної архітектури наведена на рисунку 7.

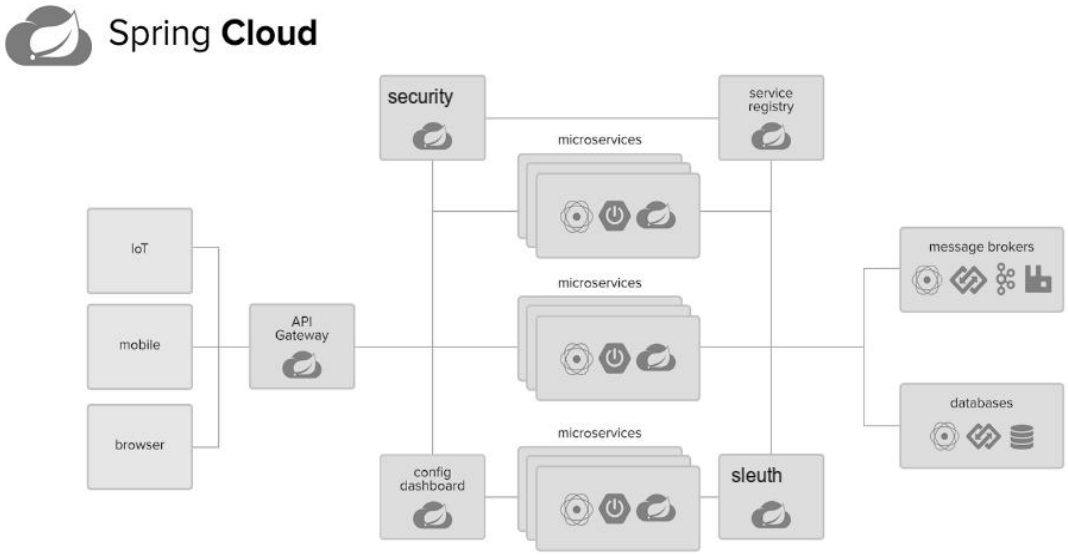

Рис. 7. Блок схема типової мікросервісної архітектури

API Gateway – ефективний мікросервіс маршрутизації до API всіх складових мікросервісів і надати їм наскрізні послуги , такі як безпека, моніторинг/метрики та стійкість.

Config DashBoard – надає можливість керувати конфігурацією між продакшн та тестовим середовищами . Стандартна реалізація серверної системи зберігання даних використовує git, тому вона легко підтримує позначені версії середовищ конфігурації, а також має доступ до широкого спектру інструментів для керування контентом.

Registry (Eureka) – це спеціальна нода в хмарі, яка реєструє всі мікросервіси всередині клауда. Мікросервіс надсилає запит до Registry Server, і той заносить його до реєстру. Коли до мікросервісу захоче звернутися інший мікросервіс, все станеться автоматично. З реєстром у Registry Server ми можемо легко зматчити ім'я та шлях.

Sleuth Service – інструментує загальні точки входу та виходу з додатків Spring (фільтр сервлетів, шаблон відпочинку, заплановані дії, канали повідомлень, імітація клієнта).

4.2. Декомпозиція моноліту на атомарні мікросервіси

Існує два підходи до такого процесу. Верхній та нижній. Верхній підхід означає, що ми поступово відокремлюємо від моноліту складові сервіси, лишаючи при цьому взаємодію з ним через HTTP-запити.

Нижній підхід означає, що починаємо розробку з нуля, поступово доставляючи нові сервіси. Кожен з цих підходів має свої переваги і недоліки.

Нами обрано другий спосіб, бо це більше відповідає дослідницьким цілям.

Кожен мікросервіс має свою власну базу даних у докер-контейнері і його необхідно наповнити актуальною інформацією для коректного тестування. Такий підхід дозволить точніше моніторити споживання ресурсів, особливо часових. Бізнес-логіка такого ресурсу буде менше за батьківський ресурс за рахунок того, що багато наскрізних сервісів (логування, безпека ...) буде загальним для всіх сервісів.

Сервіси можна поділити на три групи:

1. **Сервіси-словники.** Відрізняються тим, що дані, які вони видають, на протязі довгого часу сталі. Наприклад, сервіс DoctorService. Видає список лікарів, та їх супроводжуючі параметри. Ця інформація оновлюється дуже рідко. Для такого сервісу має сенс пересилати свої данні у кеш-сервіс на основі технології Redis.

2. **Гарячі сервіси.** Це такі сервіси, дані яких весь час змінюються. Наприклад, QueueService, який формує електронну чергу в кожний кабінет. Кожна подія з любим пацієнтом змушує цей сервіс перераховувати всі черги в кабінети. В середньому перерахунок відбувається 4 рази на хвилину.

Такі сервіси не має потреби кешувати, проте необхідно зробити серйозну заявку на горизонтальне масштабування (клонування).

3. **Сервіси по розкладу**. Це група сервісів, які запускаються у певний час (як правило, вночі). Наприклад, зведення бухгалтерського балансу або заробітна плата. Характеризуються великим навантаженням на базу і всі обчислювальні ресурси. Від них не вимагається швидкодія. Нехай працюють хоч цілу ніч. Але вимагається запускати чітко по черзі, щоб вони не перетинались у часі для запобігання DataRace.

Такий алгоритм запуску можна оркеструвати за допомогою Kubernetes, або спеціального ScheduleService.

4.3. Тестування

Моніторинг і тестування проводились на кожному кроці (мікросервісі). Перевірялась не тільки коректність роботи, але й споживання ресурсів та працездатність у режимі великих навантажень.

Коректність роботи перевіряться відповідними інтеграційними тестами, які були обов’язковими при написанні коду. Споживання ресурсів перевірялась мікросервісами Eureka та Actuator з коробки SpringCloud.

Мікросервіс ConfigDashBoard (SpringCloud) дає можливість зручно налаштовувати параметри роботи і тестування.

Тестування при великих навантаженнях проводилось за технологією Apache Benchmark. Інструментарій **Apache Bench** застосовується для комплексного навантажувального тестування веб-сервісів, що дає змогу визначити кореляцію між кількістю одночасних запитів та показниками латентності системи. Такий підхід

дозволяє ідентифікувати ймовірність виникнення відмов, встановити межі нормального та пікового навантаження, а також розробити стратегії превентивного реагування на критичні збої.

Відмовою може бути зависання програми, повідомлення про помилку або високий час обробки запиту. Зазвичай, відмови трапляються у будь-яких системах, проте все залежить після якого наробітку відмова станеться (інтенсивність відмов). Зазвичай, відмови допускаються в системах, в яких немає загрози втрати даних або системи, які не є критичними у своєму використанні.

Вимоги до часу відповіді можуть бути збільшені, якщо відбувається якась затратна операція по часу. Наприклад, архівація даних на віддаленому диску (хмарному сховищі) або складна обробка даних.

Вимоги до часу відповіді можуть бути зменшені, якщо потрібно отримати доступ до якоїсь інформації. Для зменшення часу відповіді потрібно застосовувати оптимізовані алгоритми під конкретні задачі. Також, можливе використання кластерів надійності та кешу. Результати тестування наведено у таблиці 1.

Таблиця 1.

Параметри навантаження / Система під навантаженням	Monolith (сервер станд.)	Monolith (mem 8 MB, CPU 2)	MSA (сервер станд.)	MSA (mem 8 MB, CPU 2)
-n 100, -c 25, -s 180	99%	99%	99%	95%
-n 100, -c 50, -s 180	99%	97%	95%	90%
-n 100, -c 100, -s 180	99%	96%	94%	70%

Результати тестування

Як видно з результатів тестування, для 100 одночасних користувачів час відповіді будь-якої операції у 99% випадках не повинен перевищувати 4-х секунд при використанні MSA з стандартними серверними налаштуваннями, що майже у чотири рази краще, ніж у веб-додатку з монолітною архітектурою. Ймовірність помилки (код відповіді сервера інший від 200) не повинна перевищувати 0.1% від усіх випадків. Архітектура нового додатку стала більш прозорою і менш «важкою». Враховано вимоги E-HEALTH. Підвищений рівень безпеки додатку, в тому числі особистих даних, шляхом застосування нових алгоритмів хешування на основі клітинних автоматів та послідовностей випадкових чисел. Додана інтеграція з існуючою платіжною системою. Забезпечена цілісність даних (бекапи).

Висновки

На основі аналізу існуючого веб-додатку для медичного закладу, а саме - для роботи реєстратури, розроблено технологію розщеплення моноліту на незалежні мікросервіси. Розроблено архітектуру існуючого веб-додатку для медичного центру на основі незалежних мікросервісів. Тестування та порівняння параметрів обробки даних і ресурсів, що використовувалися у попередній та оновленій версії веб-додатку показали, що для 100 одночасних користувачів час відповіді будь-якої операції у 99% випадках не повинен перевищувати 4-х секунд при використанні MSA з стандартними серверними налаштуваннями, що майже у чотири рази краще, ніж у веб-додатку з

монолітною архітектурою. Ймовірність помилки (код відповіді сервера інший від 200) не повинна перевищувати 0.1% від усіх випадків. Архітектура нового додатку стала більш прозорою і менш «важкою». Враховано вимоги E-HEALTH. Підвищений рівень безпеки додатку, в тому числі особистих даних, шляхом застосування нових алгоритмів хешування на основі клітинних автоматів та послідовностей випадкових чисел. Додана інтеграція з існуючою платіжною системою. Забезпечена цілісність даних (бекапи).

Посилання.

[1] Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. 2019, p. 272.

[2] Drahoni N., Dzhiallorentso S., Lafuente L., Mazzara M., Montezi F., Mustafin R., Safina L. Mikroservisy: vchora, sohodni ta zavtra. Present and ulterior software engineering. 2017. S. 195-216. DOI:https://doi.org/10.1145/343477.343502.

[3] Fauler M., Liuis Dzh. Mikroservisy: vyznachennia novoho arkhitekturnoho terminu. 2014. URL:https://martinfowler.com/articles/microservices.html (Accessed March 25, 2026).

[4] Niumen S. Buduvannia mikroservisiv: proektuvannia dribnozernystykh system. OReilly Media, 2015. 280 p.

[5] Pramod D. Venkataramana K., Fani S. Ohliad stratehii rozpodilu ta replikatsii danykh u rozpodilenykh bazakh danykh. International Journal of Advanced Computer Research. 2018. T. 8. № 36. S. 80-90. DOI: http://dx.doi.org/10.30534/ijatcse/2019/117852019.

[6] Hrolinher K. Khihashino V., Tivari A., Kaprets M. Vyklyky velykykh danykh: tekhnolohii, modeli ta paradyhmy. Upravlinnia danymy v khmarnykh, hrid ta P2P systemakh. Springer, Berlin, Heidelberg. 2013. S. 1-15. – DOI: https://doi.org/10.1007/978-3-642-40053-7_1.

[7] Anisimov V.H., Kunanets N.E. Perekhid vid monolitnoi do mikroservisnoi arkhitektury: metodolohiia ta dosvid vprovadzhennia. Kompiuterno-intehrovani tekhnolohii: osvita, nauka, vyrobnytstvo. 2024. Vypusk № 55. S. 30-41. DOI: https://doi.org/10.36910/6775-2524-0560-2024-55-03.

[8] Kharovyuk V. Yak rozdilyty monolit na mikroservisy. URL: https://webscraft.org/blog/yak-rozdiliti-monolit-na-mikroservisi-pokrokova-instruktsiya (Accessed March 25, 2026).

About the authors

Yuriy Dobrovolsky. Graduated from the Faculty of Physics and Mathematics in 1984. Received the degree of Doctor of Technical Sciences. Currently, he is a professor at the department of software engineering at Yuri Fedkovich Chernivtsi National University. Scientific interests include research and development of mathematical models and algorithms for creating reliable software, cryptographic algorithms, as well as reliable measuring, medical, and environmental technology.

ORCID ID:0000-0003-0626-0594.

Igor Popovych. Graduated from Ternopil Medical Institute (1978), Doctor of Philosophy (1988), Senior Researcher, Senior Researcher at the Kozyavkin International Rehabilitation Clinic, Truskavets, Ukraine. Field of scientific interests - physiology, balneology.

ORCID.org/0000-0002-5664-5591

Inessa Tanashchyshena. Graduated from Yuri Fedkovich Chernivtsi National University in 2011. Major: Informatics, Qualification: University Lecturer, Researcher (Computer Systems). Assistant at the department of software engineering at Yuri Fedkovich Chernivtsi National University. Area of scientific interests - cryptographic algorithms and computer modeling.

ORCID ID:0009-0008-5119-8420.

Mariia Hanzhelo. In 2017, she graduated from the Department of Software Engineering of the Institute of Computer Information Technologies of the National Aviation University (Kyiv). She is a postgraduate student at the Department of Software Engineering of Computer Systems of the Yuriy Fedkovych Chernivtsi National University. Scientific interests include research and development of mathematical models and algorithms for creating reliable software.

ORCID.org/0000-0002-6312-740X