Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
ISSN 1994-7836 (print)
ISSN 2519-2477 (online)
https://doi.org/10.36930/40350418
Article received 09.06.2025
Article accepted 26.06.2025
UDC 004.[415.2+052]:519.688
Correspondence author: M. O. Shestakovych shestakovych.m@nltu.lviv.ua

М. О. Шестакович[1], Ю. В. Шабатура[2]

[1] Національний лісотехнічний університет України, м. Львів, Україна
[2] Національна академія сухопутних військ ім. гетьмана Петра Сагайдачного, м. Львів, Україна

ТЕХНОЛОГІЯ ДЕКОМПОЗИЦІЇ ФРОНТЕНД-МОНОЛІТНИХ ВЕБЗАСТОСУНКІВ НА МІКРОФРОНТЕНДИ МЕТОДОМ КЛАСТЕРИЗАЦІЇ ЇХ ГЕТЕРОГЕННИХ ГРАФІВ

Здійснено дослідження та аналіз закономірностей декомпозиції фронтенд-монолітних вебзастосунків на мікрофронтенди методом кластеризації їх гетерогенних графів із використанням графових нейронних мереж GNN (англ. *Graph Neural Networks*). Досліджено вплив поєднання різних типів зв'язку між компонентами (структурних, функціональних і тестових) на якість виділення функціонально цілісних груп елементів інтерфейсу. З'ясовано, що врахування не тільки статичних залежностей, а й динамічних сценаріїв взаємодії у тестах дає змогу ідентифікувати приховані контекстні зв'язки, недоступні для стандартних методів аналізу коду. Доведено, що застосування моделі GraphSAGE (англ. *Sample and AggregatE*) для побудови векторного подання вузлів гетерогенного графа поєднано з кластеризацією методом *k*-середніх (*k*-means) забезпечує обґрунтованіший поділ моноліту на групи компонентів, які відповідають основним доменам функціоналу вебзастосунку. Виконано порівняльний аналіз із результатами кластеризації базовим алгоритмом алгоритму Louvain (виявлення спільнот), що дало можливість визначити переваги та обмеження запропонованого підходу. З'ясовано, що підхід на підставі графових нейронних мереж дає змогу уникнути надмірної фрагментації та об'єднати компоненти з подібними сценаріями використання, визначаючи потенційні межі майбутніх мікрофронтендів. Досліджено вплив різних підходів до формування графа на кінцевий результат декомпозиції. Виявлено, що оптимальний вибір кількості кластерів забезпечує баланс між деталізацією та узагальненням, а врахування різноманітних типів зв'язку сприяє виділенню найбільш логічних підсистем. Результати дослідження підтверджують ефективність використання кластеризації гетерогенних графів для завдань архітектурного рефакторингу та модернізації складних інформаційних систем, а також доводять, що запропонований підхід можна рекомендувати для автоматизованої підтримки процесів міграції від фронтенд-монолітів до мікрофронтендів у практиці корпоративного розроблення промислових вебзастосунків та слугувати основою для подальших досліджень у сфері вдосконалення структурних трансформацій програмних систем.

Ключові слова: оптимізація структури; графові нейронні мережі; функціональна когезія; автоматизована міграція; інтерфейсні модулі; аналіз взаємодії.

Вступ / Introduction

Мікросервісна архітектура усуває ключові недоліки монолітів, зокрема складність підтримки, масштабування та оновлень, розподіляючи систему на незалежні сервіси, кожен з яких можна розробляти й масштабувати окремо. Питання вибору архітектурної моделі для побудови масштабованих інформаційних систем широко досліджують у сучасній літературі, зокрема порівнянням мікросервісного та сервісно-орієнтованого підходів [6, 10].

На фронтенд-боці вебзастосунків відбувається аналогічна еволюція архітектурних підходів. Зростання розміру і складності клієнтських застосунків призвело до появи концепції мікрофронтендів – логічного продовження мікросервісної парадигми у сфері розроблення інтерфейсу користувача. Архітектура мікрофронтендів передбачає поділ фронтенду на менші, частково незалежні "мікрозастосунки", що взаємодіють у межах єдиного інтерфейсу. Це дає змогу різним командам паралельно працювати над окремими фрагментами застосунку, пришвидшуючи процес їх розроблення й оновлення. Практичний досвід великих компаній (DAZN, IKEA, Starbucks тощо) підтверджує, що мікрофронтенди забезпечують ті самі переваги, що й архітектура мікросервісів на сервера – зростання масштабованості та підтримуваності системи за рахунок незалежного розвитку крос-функціональних модулів [16]. Однак, як і в разі мікросервісів, впровадження мікрофронтендів супроводжується певними викликами. Зокрема, розподіл фронтенду може збільшити загальний обсяг коду, призвести до дублювання логіки та ускладнити координацію між командами. Також виникають проблеми управління залежностями й інтеграції між мікрофронтендами, особливо коли над фронтендом працюють різні команди [4]. Отже, можна зробити висновок про актуаль-

Інформація про авторів:
Шестакович Маркіян Олегович, магістр, аспірант, кафедра інженерії програмного забезпечення.
Email: shestakovych.m@nltu.lviv.ua; https://orcid.org/0009-0005-6714-1582
Шабатура Юрій Васильович, д-р техн. наук, професор, кафедра електроніки і електромеханіки.
Email: shabaturayuriy@gmail.com; https://orcid.org/0000-0002-9961-1244
Цитування за ДСТУ: Шестакович М. О., Шабатура Ю. В. Технологія декомпозиції фронтенд-монолітних вебзастосунків на мікрофронтенди методом кластеризації їх гетерогенних графів. Науковий вісник НЛТУ України. 2025, т. 35, № 4. С. 159–165.
Citation APA: Shestakovych, M. O., & Shabatura, Yu. V. (2025). Technology of decomposition of frontend-monolithic web applications into microfrontends by clusterization of their heterogeneous graphs. *Scientific Bulletin of UNFU*, 35(4), 159–165. https://doi.org/10.36930/40350418

ність завдання пошуку методів оптимальної декомпозиції фронтенд-монолітів, які мінімізують ці проблеми й забезпечать обґрунтований поділ системи.

Об'єкт дослідження – декомпозиція фронтенд-монолітних вебзастосунків.

Предмет дослідження – методи і засоби побудови гетерогенних графів, які дадуть змогу сформувати їх векторні подання та виділити межі потенційних мікрофронтендів через кластеризацію у просторі ембедінгів.

Мета роботи – розробити технологію декомпозиції фронтенд-монолітної архітектури на мікрофронтенди методом кластеризації з використанням графових нейронних мереж, що підвищить ефективність та обґрунтованість поділу архітектури застосунку для підтримки його масштабованості, модульності та розвитку системи.

Для досягнення зазначеної мети визначено такі основні *завдання дослідження*:

- проаналізувати сучасні підходи до організації мікросервісної та мікрофронтендної архітектур, визначити їх переваги й обмеження порівняно з монолітними підходами у фронтенд-розробленні, що дасть змогу обґрунтувати актуальність і потребу структурної декомпозиції фронтенд-застосунків;
- розробити метод побудови гетерогенного графа фронтенд-застосунку, який враховує різні типи компонентів і зв'язків (структурних, функціональних, тестових), що дасть змогу інтегрувати інформацію про різнотипні залежності у єдину модель для аналізу структури застосунку;
- реалізувати автоматизований процес навчання ембедінгів вузлів графа із застосуванням моделі GraphSAGE та сформувати відповідні ознаки для якісної кластеризації вузлів, що забезпечить урахування як статичних, так і динамічних зв'язків між компонентами в разі подальшого їх групування;
- провести кластеризацію вузлів графа методом *k*-середніх у просторі ембедінгів, отримати оптимальний розподіл компонентів на групи (кандидати у мікрофронтенди), порівняти результати з базовою кластеризацією алгоритму Louvain, що дасть змогу оцінити переваги запропонованого підходу над традиційними методами;
- оцінити якість і доцільність отриманого поділу компонентів, проаналізувати його відповідність функціональній структурі застосунку, сформулювати рекомендації для практичного використання методу за міграції від монолітної архітектури до мікрофронтендів, що дасть змогу підвищити ефективність підготовки архітектурних рішень у реальних програмних проєктах.

Аналіз останніх досліджень та публікацій. Питання поділу монолітних застосунків на дрібніші модулі привертає значну увагу дослідників і практиків у галузі програмної інженерії. Концепцію мікросервісної архітектури було вперше сформульовано у 2012-2014 рр. і вона набула популярності завдяки можливості підвищити гнучкість розвитку програмних систем. Згідно з визначенням Dragoni [5, с. 206], мікросервіс – це самодостатній незалежний процес, що взаємодіє з іншими винятково через повідомлення, не розділяючи з ними спільного стану. Мікросервісну архітектуру, відповідно, розглядають як розподілений застосунок, усі компоненти якого є мікросервісами, тобто можуть розроблятися, розгортатися та підтримуватися незалежно і взаємодіють між собою через чітко визначені інтерфейси. Такий підхід значно полегшує підтримку і масштабування великих систем, вирішуючи проблеми монолітів, зокрема, складність внесення змін у взаємозалежному коді, тривалий простій під час оновлень та обмеження паралельної роботи над різними частинами проєкту.

У роботі [16] проаналізовано мотиви, переваги і проблеми впровадження мікрофронтендів. Згідно з їхніми результатами, основною мотивацією компаній є масштабування розроблення великого фронтенду, коли декілька команд одночасно працюють над одним застосунком. Традиційні підходи (моноліт або єдиний SPA з розподілом тільки на компоненти) стають громіздкими зі зростанням команди розробників і кількості функцій застосунку. Застосування мікрофронтендів підтвердило очікувані вигоди – вони забезпечують аналогічні до мікросервісів переваги у вебзастосунку: підвищують модульність та масштабованість вебзастосунків. Зокрема, кожна команда може повністю відповідати за свій фрагмент інтерфейсу (повний вертикальний зріз функціоналу), обираючи власний стек технологій та випускаючи оновлення незалежно від інших, що пришвидшує розвиток продукту.

Негативні наслідки та обмеження архітектури мікрофронтендів також досліджують у сучасній літературі. Зокрема, автори роботи [4] відзначають, що поділ фронтенду на багато дрібних застосунків може збільшити загальний розмір пакету, призвести до дублювання логіки та ускладнити відстеження стану системи. Координація роботи кількох мікрофронтенд-команд потребує налагоджених процесів; також з'являються додаткові витрати на інтеграцію та тестування сумісності різних частин інтерфейсу. Для пом'якшення цих проблем пропонують шаблони архітектурних рішень, такі як Backend-for-Frontend, серверна композиція інтерфейсу тощо [9], однак повністю уникнути додаткової складності не вдається. Додатково, у праці [7] запропоновано мову архітектурних рішень, яка дає змогу формалізувати вибір підходу до декомпозиції під час проєктування мікросервісних систем.

Літературний аналіз останніх досліджень та публікацій показує, що питання здійснення "розумної" декомпозиції фронтенд застосунку є актуальним і відкритим. Існують рекомендації з поділу фронтенд-моноліту за доменними контекстами (аналогічно до підходів Domain-driven Design, що застосовують для мікросервісів), проте формальних методів або алгоритмів, які автоматично визначають оптимальні межі між мікрофронтендами, поки що мало. Здебільшого поділ здійснюють експертно – на підставі знання предметної області та коду. Це мотивує здійснити дослідження методів автоматизованої кластеризації компонентів фронтенду з метою виявлення груп, які слабо пов'язані з іншими і можуть бути виділені в окремі підсистеми (мікрофронтенди).

Матеріали та методи дослідження. Для досягнення мети дослідження використано демонстраційний вебзастосунок CRW-App як типовий фронтенд-моноліт. Структуру застосунку змодельовано у вигляді гетерогенного графа з різними типами вузлів і зв'язків. Для отримання векторного подання компонентів застосовано GraphSAGE – графову нейронну мережу. Подальшу кластеризацію здійснено методом *k*-середніх; для порівняння ефективності використано алгоритм Louvain. Аналіз результатів ґрунтувався на порівнянні отриманих кластерів з функціональною структурою застосунку.

Результати дослідження та їх обговорення / Research results and their discussion

Фронтенд-моноліт вебзастосунку моделюють зазвичай гетерогенним графом із трьома типами вузлів: ком-

поненти інтерфейсу, маршрути (сторінки) та інтеграційні тести. Таке його подання дає змогу об'єднати в єдину модель різнорідні зв'язки: композиційні (між сторінками й компонентами), залежності імпорту між компонентами та зв'язки "тест – компонент" (якщо тест перевіряє роботу компонента). Формально отриманий граф задають так [11]:

$$G = f(V, E), \quad (1)$$

де $V=V_{comp} \cup V_{route} \cup V_{test}$ – множина вузлів (компоненти інтерфейсу, сторінки, тести), а $E=E_{pc} \cup E_{tc} \cup E_{cc}$ – множина ребер.

У зразку CRW-App граф містить 103 вузли (58 компонентів, 11 сторінок, 34 тести) і 212 ребер. На рис. 1 показано фрагмент структури такого графа: вузли UI-компонент (позначено оранжевим) згруповані навколо вузлів-сторінок (синім) і пов'язані через спільні вузли тестів (зеленим), що виступають "ланками зв'язку" між компонентами.

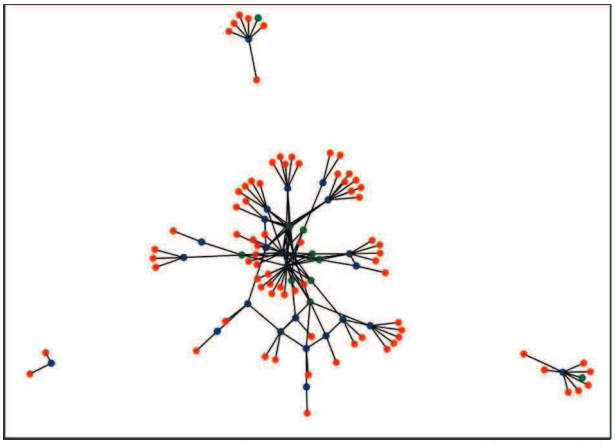

Рис. 1. Гетерогенний граф структури застосунку (фрагмент) / Heterogeneous graph of the application structure (fragment)

Це означає, що компоненти, які розміщені на одній сторінці або перевіряються одним тестом, утворюють щільно пов'язані підграфи, слабо зв'язані з іншими частинами системи. Зокрема, внесення вузлів тестів формує гіперзв'язки між компонентами, що трапляються разом у тестових сценаріях: наприклад, сторінки "баланс рахунку" і "список транзакцій", хоч і не мають прямого імпорту одна в одну, опосередковано з'єднані спільним тестом (перевірка оновлення балансу після створення транзакції). Отже, гетерогенний граф враховує як статичні (імпортні та композиційні), так і динамічні (спільна участь у тестах) залежності між компонентами, закладаючи основу для подальшого аналізу.

Для виділення спільнот на графі застосовано алгоритм Louvain, що максимізує модульність Ньюмана-Гірвана. Його цільову функцію задають формулою [3]:

$$Q = \frac{1}{2m} \sum_{i,j}^{m} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \quad (2)$$

де: A_{ij} – матриця суміжності; k_i – ступінь i-го вузла (кількість ребер, приєднаних до вузла); m – кількість ребер графа; $\delta(c_i, c_j) = 1$ для вузлів у межах одного кластера (= 0 – інакше).

Алгоритм Louvain виявив 13 спільнот компонентів. Проте аналіз показав надмірну деталізацію такого поділу: компоненти одного Louvain-кластера на PCA-діаграмі (рис. 2) часто розсіяні або утворюють тільки дрібні групи, що свідчить про низьку внутрішню когезію таких кластерів. Іншими словами, кластеризація за алгоритмом Louvain дала багато малих, надто фрагментарних спільнот.

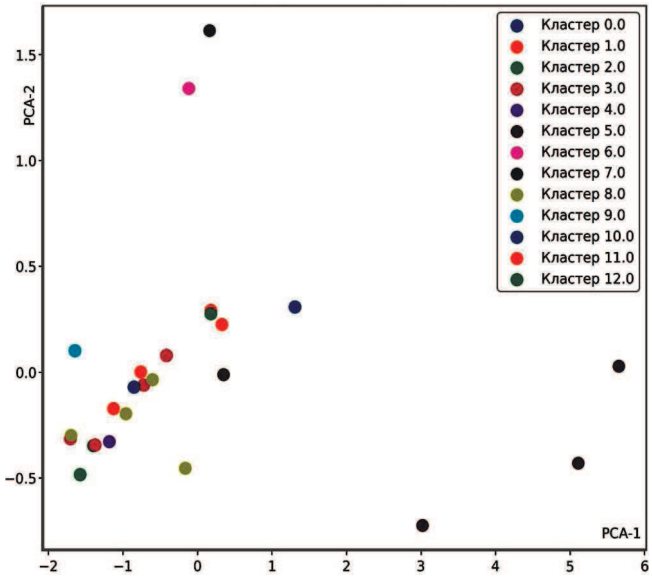

Рис. 2. Розподіл вузлів графа у просторі ембедінгів (PCA), кольором відзначено кластери за методом Louvain (13 кластерів) / Distribution of graph nodes in the embedding space (PCA), clusters marked by colour according to the Louvain method (13 clusters)

Щоб подолати фрагментарність Louvain-кластерів, застосовано комбінований підхід GraphSAGE+k-means. Спершу за допомогою GraphSAGE на графі навчено вузлові ембедінги з mean-агрегатором. На ітераціях оновлення ознаки вузла v описуються таким рівнянням [8]:

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot \left[\mathbf{h}_v^{(k-1)} \,\|\, \text{mean}_{u \in \mathcal{N}(v)} \left(\mathbf{h}_u^{(k-1)} \right) \right] \right), \quad (3)$$

де: $\mathbf{h}_v^{(k)}$ – вектор ознак v-го вузла на k-му шарі (для $k = 1$ це початковий вектор ознак x_v); $N(v)$ – множина сусідів v-го вузла; mean – оператор усереднення; $\mathbf{W}^{(k)}$ – матриця вагових коефіцієнтів k-го шару, σ – нелінійна активація (ReLU).

Це дає змогу врахувати контекстні зв'язки вузла разом із локальними сусідами. Після навчання кожен вузол має векторне подання, і всі подання кластеризують методом k-середніх. Функцію k-середніх визначають за формулою [12]:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2, \quad (4)$$

де: C_i – множина точок (ембедингів) у i-му кластері; μ_i – центр цього кластера. На підставі графіка "ліквідації" інерції і доменних знань вибрано $k = 6$ і отримано 6 функціонально осмислених кластерів компонент. Найбільший із них містить 18 компонент (враховуючи 2 сторінки), що значно більше за максимальний Louvain-кластер (10 компонент). Ембедінги GraphSAGE об'єднали вузли з подібним контекстом (спільними сусідами чи участю в тестах), які раніше були в різних Louvain-кластерах. Унаслідок отримана декомпозиція відображає основні функціональні підсистеми: модулі автентифікації, платіжних операцій, соціальних функцій, налаштувань та дашборду. На рис. 3 наведено PCA-простір вузлів із фарбуванням за кластерами GraphSAGE+k-means: видно, що вузли одного кластера утворюють набагато щільніші групи, ніж на рис. 2, що свідчить про більшу когезію й осмисленість кластерів за використання GNN-ембедінгів.

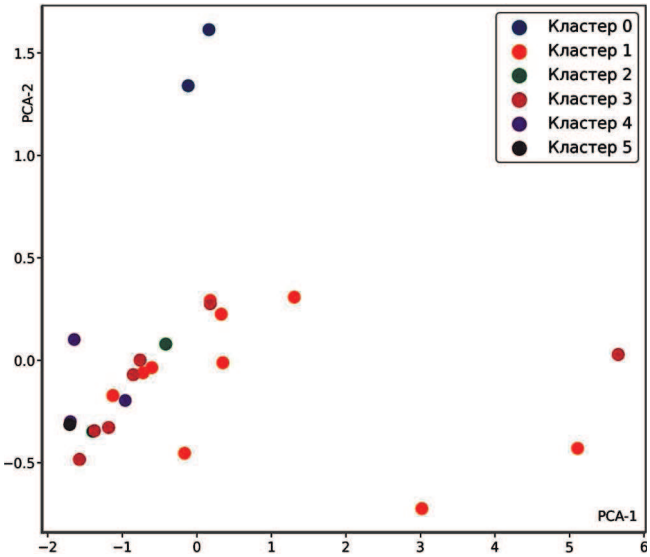

Рис. 3. Розподіл вузлів графа у просторі ембедінгів (PCA), кольором відзначено кластери за методом GraphSAGE + k-means / Distribution of graph nodes in the embedding space (PCA), clusters marked by colour according to the GraphSAGE+k-means method

Для порівняння розподілів побудовано діаграму Sankey (рис. 4), яка ілюструє перерозподіл компонентів між Louvain-кластерами (ліворуч) та GraphSAGE+*k*-means (праворуч). Криві потоку показують, скільки компонент із кожного Louvain-кластера перейшли до конкретного GNN-кластера. Як видно, декілька початкових спільнот алгоритму Louvain фактично об'єдналися під час застосування GraphSAGE: наприклад, спільно пов'язані з платежами та сповіщеннями елементи увійшли в один кластер G3 (що логічно, оскільки ці функції тісно переплетені). Водночас, деякі Louvain-кластери залишилися майже без змін, що свідчить про осмислене об'єднання їх там, де були приховані зв'язки, і про стабільність уже чітких модулів.

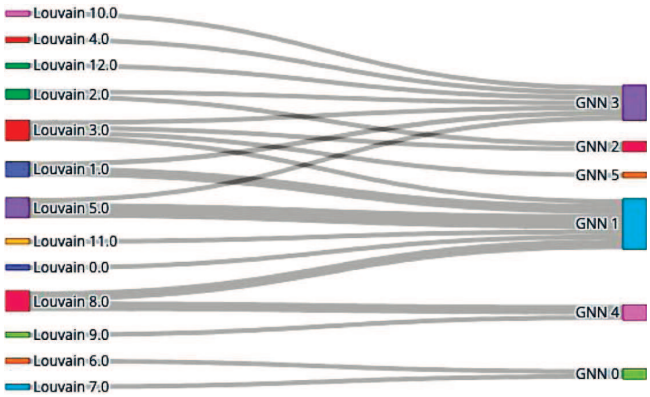

Рис. 4. Діаграма Sankey для порівняння кластеризації компонентів / Sankey diagram for comparison of component clustering

Для оцінювання якості кластеризації у дослідженні було використано коефіцієнт силуету та Adjusted Rand Index (ARI). Коефіцієнт силуету відображає ступінь компактності та розділеності кластерів, порівнюючи середню відстань між елементом і всіма іншими елементами свого кластера з мінімальною середньою відстанню до елементів іншого кластера [18]:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \forall i \in m \,. \qquad (5)$$

Для поділу, отриманого методом GraphSAGE+*k*-середніх, середнє значення коефіцієнта силуету становило 0,68, що істотно перевищує аналогічний показник для Louvain-кластеризації (0,19), і свідчить про вищу згуртованість та чіткість меж між кластерами за використання GNN-ембедінгів.

Додатково, для кількісного порівняння розподілів використано Adjusted Rand Index (ARI), що характеризує ступінь збігу між двома різними кластеризаціями. Для нашого експерименту ARI дорівнював 0,53, що вказує на помірний рівень відповідності – приблизно половина пар елементів залишилась у тих самих кластерах, а інша частина була перегрупована внаслідок урахування складніших контекстних зв'язків у структурі даних. Це підтверджує здатність запропонованого підходу виявляти нові, логічно осмислені групи компонентів, які не визначаються стандартними структурними ознаками. Узагальнені кількісні та якісні результати дослідження наведено у табл. 1.

Табл. 1. Основні результати дослідження декомпозиції фронтенд-моноліту / Key results of the decomposition study of the frontend monolith

Етап / Метод	Опис результату	Кількісні параметри	Якісна оцінка
Гетерогенний граф	3 типи вузлів (58 компонентів, 11 сторінок, 34 тести)	103 вузли, 212 ребер	Урахування статичних і динамічних зв'язків
Louvain	Виявлено спільноти алгоритмом Louvain	13 кластерів, макс. 10 компонентів	Сильна фрагментація, низька когезія (0,19)
GraphSAGE+*k*-means	Кластеризація у просторі ембедінгів	6 кластерів, макс. 18 компонентів	Висока когезія (0,68), відповідність доменам

Важливу роль у групуванні компонент має внесення вузлів інтеграційних тестів у граф. Хоча тести не входять до виконуваного коду, вони виступають "спостерігачами" взаємодії компонент: одним тестовим сценарієм об'єднуються декілька компонент, перевірка роботи яких проходить у рамках одного тесту. Це формує гіперзв'язки між компонентами, що часто використовуються разом (навіть, якщо між ними немає прямого імпортного зв'язку). Наприклад, сторінка створення платежу і сторінка списку транзакцій виявилися пов'язаними через спільний тест (створення нового платежу та перевірка його відображення у списку); GraphSAGE врахував це за навчання ембедінгів і відніс ці вузли до одного кластера (платежі "*Transactions*"). Розподіл компонентів, що пов'язані із *Transaction*, у Louvain- та GNN-кластерах представлено у табл. 2.

Компонент навігаційного меню та індикатор сповіщень практично не мали прямих зв'язків (окрім того, що обидва трапляються на головній сторінці), тому алгоритм Louvain відніс їх до різних кластерів (NavBar окремо, Badge – до модуля сповіщень). Проте GraphSAGE зблизив їхні ембедінги (через спільного сусіда – головну сторінку і схожих сусідів), що призвело до їхнього об'єднання в один кластер Dashboard. Ці приклади демонструють здатність підходу виявляти приховані контекстні зв'язки між компонентами і формувати логічно осмислені підсистеми, навіть якщо такі зв'язки не відображені явно в коді.

Зазначимо декілька важливих особливостей щодо налаштування та обмежень методу. По-перше, параметр *k* (кількість кластерів) відіграє роль: було обрано $k = 6$ на підставі аналізу кривої інерції та доменних знань, праг-

нучи отримати осмислені середньорозмірні кластери. Занадто велике *k* спричинило б занадто дрібні групи (як у алгоритмі Louvain), а занадто мале – надто грубе об'єднання різних функціональностей. По-друге, спостерігалася нерівномірність розмірів кластерів: один кластер (Dashboard) вийшов доволі великим, що відображає його центральну роль, але такий "перевантажений" кластер може вимагати додаткової рекурсивної декомпозиції (наприклад, ієрархічною кластеризацією). По-третє, перед кластеризацією векторних подань вузлів виконана їх нормалізація: без цього ознаки з широким діапазоном домінували б у метриці відстані, спотворюючи результати. Нормалізація ембедингів дала змогу підвищити когезію кластерів, забезпечивши, щоб жоден тип ознак (наприклад, частоти викликів чи центральностей) не мав непропорційної ваги.

Табл. 2. Розподіл компонентів, пов'язаних із Transaction, за кластерами методів Louvain та GraphSAGE+k-means / Distribution of components related to Transaction across clusters obtained by the Louvain and GraphSAGE+k-means methods

Назва компонента	Кластер Louvain	Кластер GraphSAGE+*k*-means
src/components/TransactionCreateStepThree.tsx	9	4
src/components/TransactionCreateStepTwo.tsx	10	3
src/components/TransactionDateRangeFilter.tsx	11	1
src/components/TransactionDetail.tsx	2	3
src/components/TransactionInfiniteList.tsx	8	1
src/components/TransactionItem.tsx	8	1
src/components/TransactionList.tsx	8	4
src/components/TransactionListAmountRangeFilter.tsx	0	1
src/components/TransactionNavTabs.tsx	5	1

Обмеження запропонованого підходу пов'язані з повнотою даних про залежності компонент. Якщо в системі відсутні інтеграційні тести або чітко визначені точки входу (роути), то граф міститиме здебільшого статичні імпорт-зв'язки, і метод гетерогенної кластеризації зводиться до вдосконаленого аналізу цих залежностей – у таких випадках переваги гетерогенності будуть менш виразними. Водночас, підхід GraphSAGE+*k*-means добре масштабується: GraphSAGE, як індуктивний GNN-підхід, може працювати з великими графами (тисячі чи мільйони вузлів), тому його застосування можливе і в промислових фронтенд-монолітах із сотнями компонентів (за умови наявності обчислювальних ресурсів). Проте для успішного практичного застосування потрібно мати доступ до різнорідних даних (логи сценаріїв, тестові кейси, конфігурація роутів тощо); без них запропонований підхід звузиться до статичної кластеризації і не розкриє повною мірою свого потенціалу.

Обговорення результатів дослідження. Результати, отримані в нашій роботі, узгоджуються із сучасними тенденціями в області декомпозиції монолітних застосунків та застосування графових моделей і нейронних мереж для автоматизації цього процесу. Зокрема, у роботі [1] автори вказують на те, що перехід від моноліту до мікросервісної архітектури найчастіше ґрунтується на графовому поданні початкової системи, де вузли відповідають компонентам, а ребра – залежностям між ними. Автори пропонують фреймворк M2MDF і виділяють ключові етапи та методи декомпозиції, що підтверджує доцільність використання подібної моделі, як і у нашому дослідженні.

У роботі [14] розглянуто застосування графових нейронних мереж у мікросервісних архітектурах хмарних систем. Аналіз показує, що GNN, зокрема згорткові архітектури, поступово стають основою для вирішення завдань удосконалення структури, автоматизації розподілу ресурсів та підвищення гнучкості системи. Визначені авторами перспективні напрями – просторово-часові та динамічні GNN – підтверджують, що подальший розвиток графових підходів є актуальним і для завдань, пов'язаних із декомпозицією frontend-монолітів.

Автори роботи [13] представили гетерогенну графову нейронну мережу (CHGNN) для завдання трансформації монолітного програмного забезпечення у мікросервіси. У їхній роботі, аналогічно до нашого підходу, застосунок моделюють як гетерогенний граф, у якому враховано різні типи сутностей. Експериментальні результати CHGNN свідчать про високу якість кластеризації компонентів за ознаками згуртованості та слабкого зв'язку між кластерами, що є важливим для формування осмислених мікросервісів – ці висновки узгоджуються з результатами, які ми отримали для frontend-компонентів.

Автори у роботі [17] запропонували метод вилучення мікросервісів GDC-DVF на підставі глибокої кластеризації графів із об'єднанням структурної і семантичної інформації. Їх підхід передбачає використання двох графових подань (структурного і бізнес-орієнтованого) з подальшою їхньою інтеграцією в GNN. Це дає змогу досягати кращих результатів кластеризації порівняно з традиційними методами. Власне, ідея поєднання різних типів зв'язків та аналізу контекстуальних залежностей, використана в цій праці, є співзвучною з нашим підходом побудови гетерогенного графа для фронтенд-моноліту.

Автори роботи [21] розробили підхід MAGNET, який автоматизує виділення мікросервісів на підставі графових нейронних мереж, оперуючи графом методів із багатими семантичними і статичними ознаками. Результати експериментів свідчать про високу точність і повноту кластеризації, а також про здатність забезпечити модульність сервісів. Це узгоджується з нашими висновками щодо ефективності використання GNN для виявлення функціонально цілісних груп у гетерогенній структурі frontend-застосунку.

Питання поділу фронтенд-моноліту на мікрофронтенди активно обговорюють у сучасній літературі. Так, у роботі [19] зазначено, що мікрофронтенди сприяють підвищенню масштабованості й гнучкості, однак, водночас ускладнюють підтримку консистентного інтерфейсу та потребують ретельного архітектурного планування. Це співвідноситься з нашими спостереженнями щодо необхідності формування осмислених модулів, чітко відповідних функціональним областям.

Автори роботи [2] наголошують, що впровадження мікрофронтендів у великі фронтенд-проєкти є доцільним тільки за умови відповідного масштабу й наявності технічного контролю. Мікрофронтенди дають змогу пришвидшити процес розроблення і підвищити незалежність команд, проте, як і в нашому дослідженні, автори вказують на потребу у балансі між поділом і підтримкою цілісності системи.

У роботі [15] автори вказують на те, що ідентифікація мікросервісів у процесі декомпозиції моноліту залишається відкритою дослідницькою проблемою. Автори відзначають недостатню автоматизацію процесу й наголошують на потребі інтеграції статичних і динамічних методів, а також створенні стандартних підходів до валідації результатів. Зазначені у цій роботі виклики повною мірою актуальні і для завдань декомпозиції frontend-монолітів, підтверджуючи перспективність застосування комплексних графових підходів, які розкривають нові взаємозв'язки.

У роботі [20] автори розглядають комбінування статичного і динамічного аналізу за поділу монолітних застосунків. Їх гібридний підхід дає змогу врахувати як структуру коду, так і реальні сценарії використання компонентів, що забезпечує кращу якість кластеризації та підвищує відповідність отриманих сервісів реальним вимогам бізнесу. Аналогічно у нашому дослідженні було показано, що інтеграція різних типів залежностей (зокрема, з тестів і роутів) сприяє формуванню логічно згуртованих мікрофронтендів.

Результати дослідження, які ми отримали, підтверджують доцільність розроблення подібної технології, дають можливість досягати вищої якості декомпозиції та формувати функціонально цілісні мікрофронтенди у сучасних фронтенд-архітектурах.

Отже, внаслідок виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – вперше розроблено технологію декомпозиції фронтенд-монолітних застосунків, що полягає у поєднанні гетерогенного графового моделювання різнотипних зв'язків між компонентами інтерфейсу з подальшим навчанням подань вузлів засобами графових нейронних мереж та їх кластеризацією у просторі ембедінгів, що дає змогу виявляти приховані контекстні взаємозв'язки й формувати осмислені функціональні мікрофронтенди.

Практична значущість результатів дослідження – запропоновану технологію декомпозиції можна застосувати для автоматизованої підтримки процесу міграції від фронтенд-моноліту до мікрофронтендів у корпоративному розробленні, що особливо актуально для рефакторингу великих або застарілих кодових баз, і можуть бути інтегровані у практику архітектурного аналізу з метою підвищення модульності та підтримуваності сучасних вебзастосунків.

Висновки / Conclusions

Розроблено технологію декомпозиції фронтенд-монолітної архітектури на мікрофронтенди методом кластеризації з використанням графових нейронних мереж, що підвищило ефективність та обґрунтованість поділу архітектури застосунку для підтримки його масштабованості, модульності та розвитку системи. За результатами проведеного дослідження можна зробити такі основні висновки.

1. Запропоновано методологію декомпозиції фронтенд-моноліту на мікрофронтенди, що ґрунтується на поданні структури застосунку у вигляді гетерогенного графа та кластеризації вузлів цього графа засобами графових нейронних мереж. Метод враховує структурні, функціональні й тестові типи зв'язків між елементами інтерфейсу, що дає змогу виявити приховані контекстні залежності між компонентами.
2. Розроблено та апробовано технологічний процес декомпозиції, який охоплює побудову гетерогенного графа компонентів, навчання моделі GraphSAGE для отримання векторного подання вузлів та подальшу кластеризацію методом k-середніх у просторі ембедінгів. Цей процес успішно апробовано на реальному застосунку CRW-App, що підтвердило його працездатність і ефективність.
3. Отримано декомпозицію фронтенд-моноліту на п'ять мікрофронтендів, кожен із яких відповідає окремому логічному модулю системи (авторизація, платежі, соціальна взаємодія, налаштування, дашборд). Такий поділ узгоджується з експертним розумінням доменних підсистем досліджуваного застосунку, що підтверджує валідність результатів кластеризації.
4. Проведено порівняння з базовим алгоритмом Louvain та встановлено переваги GNN-методу. Застосування GraphSAGE з кластеризацією k-середніх дало змогу уникнути надмірної фрагментації, властивої Louvain, забезпечило більшу осмисленість кластерів і дозволило об'єднати компоненти, які взаємодіють у спільних сценаріях, навіть за відсутності прямих кодових зв'язків. Це підтверджує доцільність застосування графових нейронних мереж для архітектурного рефакторингу фронтенд-застосунків.
5. Обґрунтовано практичне значення запропонованої технології для розробників, які здійснюють міграцію від моноліту до мікрофронтендів. Запропонована технологія надає інструмент для автоматизованого аналізу наявної кодової бази та формування оптимального поділу на модулі. Особливо це актуально для великих та застарілих фронтенд-систем, де ручна реорганізація є утрудненою. Метод дає змогу виявити групи компонентів, які доцільно винести в окремі застосунки, що мінімізує зв'язність між ними, підвищує модульність і покращує масштабованість системи.
6. Визначено перспективи подальших досліджень, що містять перевірку ефективності підходу на інших проєктах, розширення моделі за рахунок додаткових типів вузлів (глобальний стан, виклики API) та застосування інших моделей графових нейронних мереж, адаптованих до роботи з гетерогенними графами. Перспективним також є інтегрування метрик якості декомпозиції (когезія кластерів, міжкластерна зв'язаність) безпосередньо до цільової функції під час навчання моделі, що дасть змогу здійснювати цільовий поділ графа з використанням навчання з підкріпленням або диференційованої кластеризації.

References

1. Abgaz, Y., Montalvão, J. M., de Pádua, C. I. P. S., Santos, E. C. F., Sampaio, L. S., & Barreto, A. A. (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2023.3287297
2. Amorim, T. S., & Canedo, E. D. (2025). Micro-Frontend Architecture in Software Development: A Systematic Mapping Study. *Proceedings of the 21st International Conference on Web Information Systems and Technologies – WEBIST*. https://doi.org/10.5220/0013195800003929
3. Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10), article P10008. https://doi.org/10.1088/1742-5468/2008/10/p10008
4. Cheripurathu, K. G., & Kulkarni, S. (2024). Integrating Microservices and Microfrontends: A Comprehensive Literature Review on Architecture, Design Patterns, and Implementation Challenges. *Journal of Scientific Research and Technology*, *2*(7), 1–12. https://doi.org/10.61808/jsrt115
5. Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and To-

morrow. In: Present and Ulterior Software Engineering, 195–216. Springer. https://doi.org/10.1007/978-3-319-67425-4_12
6. Fritzsch, J., Bogner, J., Zimmermann, A., & Wagner, S. (2019). From monolith to microservices: A classification of refactoring approaches. *Software Evolution and Process*, *31*(11), article ID e2068. https://doi.org/10.1007/978-3-030-06019-0_10
7. Gysel, M., Giersche, W., Müller, O., & Hölzle, K. (2016). Service Cutter: A Systematic Approach to Service Decomposition. *Service-Oriented Computing*. ICSOC 2016. *Lecture Notes in Computer Science*, 9936, 185–200. Springer. https://doi.org/10.1007/978-3-319-44482-6_12
8. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* (NeurIPS), 30, 1024–1034. https://doi.org/10.48550/arXiv.1706.02216
9. Jackson, C. (2019). Micro Frontends. MartinFowler.com. URL: https://martinfowler.com/articles/micro-frontends.html (Accessed: 12 May 2025)
10. Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The Journey So Far and Challenges Ahead. *IEEE Software*, *35*(3), 24–35. https://doi.org/10.1109/MS.2018.2141039
11. Leighton, T., Meyer, A. R., & Lehman, E. (2010). Chapter 5: Graph theory. In Mathematics for computer science (MIT OpenCourseWare). *Massachusetts Institute of Technology*. URL: https://ocw.mit.edu/courses/6-042 j-mathematics-for-computer-science-fall-2010/resources/mit6_042jf10_chap05/
12. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. V Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1, pp. 281–297. *University of California Press*. URL: https://projecteuclid.org/euclid.bsmsp/1200512992
13. Mathai, A., R G, N., Krishna, S., M, B., & Vinod, P. (2022). Monolith to Microservices: Representing Application Software through Heterogeneous Graph Neural Network. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence* (*IJCAI*), 123 p. https://doi.org/10.48550/arXiv.2112.01317
14. Nguyen, Q.-V., Nguyen, T.-H., Le, V.-K., Le, D.-K., & Vo, D.-N. (2022). A Survey on Graph Neural Networks for Microservice-Based Cloud Applications. Sensors (Basel, Switzerland), 22(23), article ID 9492. https://doi.org/10.3390/s22239492
15. Oumoussa, M., & Saidi, R. (2024). Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review. IEEE Access, 12, 26868–26892. https://doi.org/10.1109/ACCESS.2024.3365079
16. Peltonen, S., Mezzalira, L., & Taibi, D. (2021). Motivations, Benefits, and Issues for Adopting Micro-Frontends: A Multivocal Literature Review. *Information and Software Technology*, 136, article ID 106571. https://doi.org/10.48550/arXiv.2007.00293
17. Qian, Y., Yang, M., Jiang, W., & Zhang, P. (2023). Microservice extraction using graph deep clustering based on dual view fusion. Information and Software Technology, 159, article ID 107171. https://doi.org/10.1016/j.infsof.2023.107171
18. Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53–65. https://doi.org/10.1016/0377-0427(87)90125-7
19. Santos, L., Silva, M., Santos, S., Rocha, F., & Barbosa da Silva, E. (2024). Microfront-End: Systematic Mapping. In Proceedings of the 20th International Conference on Web Information Systems and Technologies – WEBIST, SciTePress, pp. 119–130. https://doi.org/10.5220/0013015400003825
20. Sellami, M., Bouguerra, S., Ghorbel, A., & Ben-Abdallah, H. (2022). Combining Static and Dynamic Analysis to Decompose Monolithic Application into Microservices. In *Proceedings of the 19th International Conference on Service-Oriented Computing* (*ICSOC 2021*), Lecture Notes in Computer Science, Vol. 13121 (pp. 220–235). Springer. https://doi.org/10.1007/978-3-031-20984-0_14
21. Trabelsi, N. Moha, Guéhéneuc, Y. -G., & Geffard, L. (2024). Magnet: Method-based Approach using Graph Neural Network for Microservices Identification. *IEEE 21st International Conference on Software Architecture* (*ICSA*), *Hyderabad*, India, 2024, pp. 1–11. https://doi.org/10.1109/ICSA59870.2024.00009

M. O. Shestakovych[1], Yu. V. Shabatura[2]

[1] Ukrainian National Forestry University, Lviv, Ukraine
[2] Hetman Petro Sahaidachnyi National Army Academy, Lviv, Ukraine

TECHNOLOGY OF DECOMPOSITION OF FRONTEND-MONOLITHIC WEB APPLICATIONS INTO MICROFRONTENDS BY CLUSTERIZATION OF THEIR HETEROGENEOUS GRAPHS

This paper presents a comprehensive approach to decomposing frontend monolithic web applications into microfrontends using heterogeneous graph clustering powered by graph neural networks (GNNs). We integrated and analyzed various types of relationships among components, in particular, structural, functional and test-driven, to identify functionally cohesive interface groups. The results of the research show that considering both static dependencies and dynamic interaction scenarios derived from test cases reveals hidden contextual links that traditional code analysis methods often miss. By applying the GraphSAGE model for node embeddings in combination with the *k*-means clustering algorithm, we achieved a more robust and meaningful partitioning of the monolith into groups of components aligned with the main functional domains of the application. Comparative analysis against the baseline Louvain method demonstrated that the proposed GNN-based technique reduces excessive fragmentation and efficiently brings together components with similar usage patterns, defining potential boundaries for future microfrontends. Furthermore, our assessment of various graph construction strategies indicates that optimal cluster selection balances detail and generalization, while the inclusion of multiple relationship types improves the identification of logical subsystems. In summary, our findings confirm the effectiveness of heterogeneous graph clustering for architectural refactoring and modernization of complex information systems. The proposed method offers a practical tool for automating the migration from frontend monoliths to microfrontends in enterprise software development and provides a foundation for further research on optimizing structural transformations in software systems.

Keywords: structural optimization; graph neural networks; functional cohesion; automated migration; interface modules; interaction analysis.