

Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж
Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікаційних та електронних систем
(назва відділення)

Циклова комісія комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра
(освітньо-професійного ступеня)

на тему: “Розробка клієнтської частини інтеграційного додатка
для платформи Atlassian Compass”

Виконав: студентка IV курсу, групи КН-423

Спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

Артем КРАВЧЕНКО
(ім'я та прізвище)

Керівник Володимир ЛІСОВИЙ
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем

Циклова комісія комп'ютерних наук

Освітньо-професійний ступінь «фаховий молодший бакалавр»

Спеціальність 122 Комп'ютерні науки

Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ

Голова циклової комісії
комп'ютерних наук

_____ Галина МАРЦІЯШ

« 02 » березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Кравченку Артему Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка клієнтської частини інтеграційного додатка для платформи Atlassian Compass

керівник роботи _____ Лісовий Володимир Миколайович

(прізвище, ім'я, по батькові)

затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.

2. Строк подання студентом роботи: 19.06.2026 р.

3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: JavaScript; фреймворки: Forge, бібліотека React, Vite, dagre, Forge-GraphQL, ReactFlow, Atlaskit, стандарти IEEE 29148-2018, IEEE 29119, ДСТУ 8302:2015.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1 Загальний розділ

1.1 Аналітичний огляд існуючих рішень

1.2 Аналіз платформи Atlassian та інфраструктури Forge

1.3 Технічне завдання

1.3.1 Найменування та область застосування

1.3.2 Призначення розробки

1.3.3 Вимоги до функціоналу додатку

1.3.4 Вимоги до програмної документації

1.3.5 Техніко-економічні показники

1.3.6 Стадії та етапи розробки

1.3.7 Порядок тестування та прийому

2 Розробка технічного та робочого проекту

2.1 Розробка графічного інтерфейсу додатку

2.2 Проектування та створення дерева залежностей додатка

2.3 Проектування та створення матриці зв'язків компонентів

2.4 Реалізація механізмів отримання інформації від віддаленого сервера

2.5 Тестування додатку

3 Спеціальний розділ

3.1 Інструкція з інсталяції додатку в системі Atlassian

3.2 Інструкція з використання додатку

3.3 Інструкція з популяризації та підтримки додатку

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості

проведення розробки додатку

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості додатку

4.7 Розрахунок ціни додатку

4.8 Визначення економічної ефективності і терміну окупності капітальних

вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Поняття праці та охорони праці

5.2 Робоча загальнообмінна штучна вентиляція

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – додатку «Compass-Mar».

5. Перелік графічного матеріалу:

1. UML-діаграма архітектури додатку

2. Структурна схема додатку

3. ER-діаграма формату даних

4. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	.06.2026	
12	Захист кваліфікаційної роботи	.06.2026	

7. Дата видачі завдання: 05 березня 2026 р.

Студент

(підпис)

Артем КРАВЧЕНКО

Керівник роботи

(підпис)

Володимир ЛІСОВИЙ

ЗМІСТ

АНОТАЦІЯ	7
ABSTRACT	8
ВСТУП	9
1 ЗАГАЛЬНИЙ РОЗДІЛ	11
1.1 Аналітичний огляд існуючих рішень.....	11
1.2 Аналіз платформи Atlassian та інфраструктури Forge	14
1.3 Технічне завдання.....	15
1.3.1 Найменування та область застосування	16
1.3.2 Призначення розробки.....	17
1.3.3 Вимоги до функціоналу додатку	19
1.3.4 Вимоги до програмної документації	20
1.3.5 Техніко-економічні показники	21
1.3.6 Стадії та етапи розробки.....	23
1.3.7 Порядок контролю та прийому.....	26
2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ	28
2.1 Розробка графічного інтерфейсу додатку	28
2.2 Проєктування та створення дерева залежностей додатка.....	34
2.3 Проєктування та створення матриці зв'язків компонентів.....	38
2.4 Реалізація механізмів отримання інформації від віддаленого сервера .	41
2.5 Тестування додатку.....	43
3 СПЕЦІАЛЬНИЙ РОЗДІЛ.....	47
3.1 Інструкція з інсталяції додатку в системі Atlassian.....	47
3.2 Інструкція з використання додатку.....	51
3.3 Інструкція з популяризації та підтримки додатку	55

					2026.КВР.122.423.08.00.00 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Кравченко А.А.			Розробка клієнтської частини інтеграційного додатка для платформи Atlassian Compass Пояснювальна записка	Лім.	Арк.
Перевір.		Лісовий В.М.					5
Реценз.						ВСП ТФК ТНТУ КН-423 м. Тернопіль	
Н. Контр.		Приймак В.А.					
Затверд.							

4	ЕКОНОМІЧНИЙ РОЗДІЛ.....	59
4.1	Визначення стадій технологічного процесу та загальної тривалості проведення розробки додатку.....	59
4.2	Визначення витрат на оплату праці та відрахувань на соціальні заходи.....	61
4.3	Розрахунок витрат на електроенергію	62
4.4	Розрахунок суми амортизаційних відрахувань додатку	63
4.5	Обчислення накладних витрат.....	64
4.6	Складання кошторису витрат та визначення собівартості додатку	64
4.7	Розрахунок ціни додатку	65
4.8	Визначення економічної ефективності і терміну окупності капітальних вкладень	65
5	ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ.....	67
5.1	Поняття праці та охорони праці.....	67
5.2	Робоча загальнообмінна штучна вентиляція	68
	ВИСНОВКИ.....	70
	ПЕРЕЛІК ПОСИЛАНЬ.....	72
	ДОДАТКИ.....	73
	Додаток А. Лістинг файлу «App.jsx».....	73
	Додаток Б. Лістинг файлу «treeHelpers.js»	76
	Додаток В. Лістинг файлу «ReactFlowTree.jsx»	80
	Додаток Г. Лістинг файлу «dependencyHelpers.jsx»	85
	Додаток Д. Лістинг файлу «DependencyTable.jsx»	90
	Додаток Е. Лістинг файлу «component-page.js».....	93

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка клієнтської частини інтеграційного додатка для платформи Atlassian Compass.

Для досягнення мети роботи виконано проектування веб-додатка, призначеного для візуального представлення та аналізу зв'язків між компонентами всередині хмарного середовища Atlassian. За структурою пояснювальна записка складається з п'яти розділів.

У загальній частині описуються аналітичний огляд існуючих рішень та аналіз технічного завдання. Також на основі порівняння визначаються недоліки аналогів та обґрунтовується необхідність розробки власного рішення.

У другому розділі представлено процес створення програмного продукту, опис та обґрунтування вибору структури та методу організації даних на базі бібліотеки React. Детально описано побудову ієрархічного дерева залежностей за допомогою ReactFlow, матриці зв'язків, асинхронний обмін даними через інструмент Forge Bridge, а також тестування та налагодження програми.

В спеціальній частині описані процес інсталяції програмного продукту, порядок тестування та інструкція з користування додатком. Загальний обсяг пояснювальної записки становить 93 сторінок.

В економічній частині проекту приведено розрахунок вартості розробки та оцінку економічної ефективності. Розраховані фінансові показники підтверджують практичну доцільність створення власного продукту.

У п'ятому розділі детально розглянуто основні питання охорони праці та техніки безпеки. Зокрема, розкрито загальні поняття праці та її охорони, а також виконано розрахунки параметрів робочої загальнообмінної штучної вентиляції.

До складу кваліфікаційної роботи входить графічна частина, яка представлена на окремих аркушах формату A1. Вона включає структурну схему графічного інтерфейсу, діаграму архітектури додатку, UML-діаграму формату даних та таблицю техніко-економічних показників.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

ABSTRACT

Thesis Topic: Development of the client-side component of an integration application for the Atlassian Compass platform.

To achieve the objective of this thesis, a web application was designed to visually represent and analyze the relationships between components within the Atlassian cloud environment. The thesis consists of five chapters.

The general section provides an analytical review of existing solutions and an analysis of the technical specifications. It also identifies the shortcomings of comparable solutions through comparison and justifies the need to develop a proprietary solution.

The second section presents the process of creating the software product, along with a description and justification of the choice of structure and data organization method based on the React library. It describes in detail the construction of a hierarchical dependency tree using ReactFlow, the relationship matrix, asynchronous data exchange via the Forge Bridge tool, as well as testing and debugging the application.

A special section describes the software installation process, the testing procedure, and instructions for using the application. The total length of the explanatory note is 93 pages.

The economic section of the project presents a cost estimate for development and an assessment of economic efficiency. The calculated financial indicators confirm the practical feasibility of creating an in-house product.

The fifth chapter examines key issues of occupational health and safety in detail. In particular, it explains general concepts of labor and occupational health, and provides calculations of the parameters for general-exchange mechanical ventilation.

The thesis includes a graphical section, presented on separate A1 -size sheets. It comprises a structural diagram of the graphical user interface, an application architecture diagram, a UML diagram of the data format, and a table of technical and economic indicators.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		8

ВСТУП

Під час розробки сучасних масштабних продуктів більшість великих технологічних компаній повністю переходять на використання розподіленої мікросервісної архітектури замість звичних застарілих монолітних рішень. Завдяки впровадженню такого технічного підходу гарантується збереження дуже високої гнучкості всієї системи. Проте внаслідок стрімкого збільшення кількості незалежних мікросервісів процес повсякденного керування інфраструктурою стає надзвичайно складним завданням навіть для найдосвідченіших інженерних команд компанії [1].

Сучасні дослідження підтверджують, що без належної візуалізації підтримка розгалужених мереж мікросервісів призводить до критичного накопичення технічного боргу [2]. Ця архітектура швидко стає вкрай заплутаною. Зокрема, прихована мережа логічних зв'язків між програмними об'єктами дуже часто залишається зовсім неочевидною без постійного оновлення належного технічного документування процесів. Технічні помилки виникають досить часто та регулярно. Через відсутність чіткої та свіжої візуалізації фахівці витрачають невинновдано багато дорогоцінного часу на швидкий пошук кореневих причин виникнення критичних апаратних збоїв.

Для успішного вирішення подібних проблем хмарні середовища на кшталт Atlassian Compass пропонують зберігати всі розрізнені дані в єдиному зручному централізованому каталозі. На практиці стандартні вбудовані інструменти не завжди відображають повну архітектурну картину платформи. Використовуючи базові методи звичайного перегляду табличної інформації, інженери зовсім не мають змоги миттєво проаналізувати глибоку багаторівневу ієрархію великого проєкту зі складною структурою.

З огляду на ці недоліки, головною метою роботи визначено розробку додатка для глибокої візуалізації архітектурних зв'язків. Щоб досягти вказаного результату, передбачено виконання чіткої послідовності практичних завдань під час написання коду майбутнього функціонального додатку.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		9

Спочатку проводиться детальне дослідження існуючих алгоритмів автоматичної побудови інтерактивних графів. Далі здійснюється проектування складного графічного інтерфейсу з суворим дотриманням усіх корпоративних стандартів системи Atlassian Design System.

Після підготовки візуальних макетів розробляються алгоритмічні механізми асинхронного обміну даними через захищений міст Forge Bridge для безпечного отримання масивів формату JSON.

Завершальним етапом стає ретельне детальне тестування всіх створених інтерфейсних компонентів під час тривалої роботи з великими обсягами складної технічної інформації. Цей процес вимагає надзвичайної уваги від розробника.

Через постійне зростання суворих вимог до загальної надійності корпоративних серверів необхідність проведення поточної технічної розробки є повністю обґрунтованою та дуже доречною.

У результаті створений програмний продукт Compass-map дозволить усім командам програмістів миттєво оцінювати масштаб впливу будь-яких несподіваних змін у робочому коді на мікросервіси.

Насправді додаток створює зручний робочий простір. Щодо галузі можливого використання, то запропоновані програмні рішення охоплюють відділи контролю якості та системного адміністрування у великих технологічних корпораціях по всьому світу.

Надалі сформовані технічні висновки та детально описані рекомендації можуть бути успішно застосовані для суттєвого полегшення процесів щоденного навчання нових інженерів компанії.

Завдяки зручній наочній візуальній схемі нові працівники зможуть набагато швидше орієнтуватися у заплутаній програмній інфраструктурі власного великого корпоративного проекту розробки.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Вибір правильного інструменту для побудови візуальних моделей часто викликає труднощі. Інженери зазвичай використовують готові корпоративні платформи для нагляду за станом компонентів та їхніми зв'язками. Вони допомагають швидко знаходити архітектурні проблеми. Розглянемо декілька популярних продуктів, які компанії використовують для вирішення таких завдань.

Серед систем управління каталогами сервісів з відкритим кодом виділяється платформа Backstage. Її створили розробники з компанії Spotify. Для показу зв'язків між елементами в екосистемі програмісти переважно обирають базовий плагін графіка залежностей. Цей модуль читає інформацію з текстових файлів конфігурації. Потім він будує наочну мережу, вигляд якої показано на рисунку 1.1. Завдяки простій логіці інструмент працює стабільно.

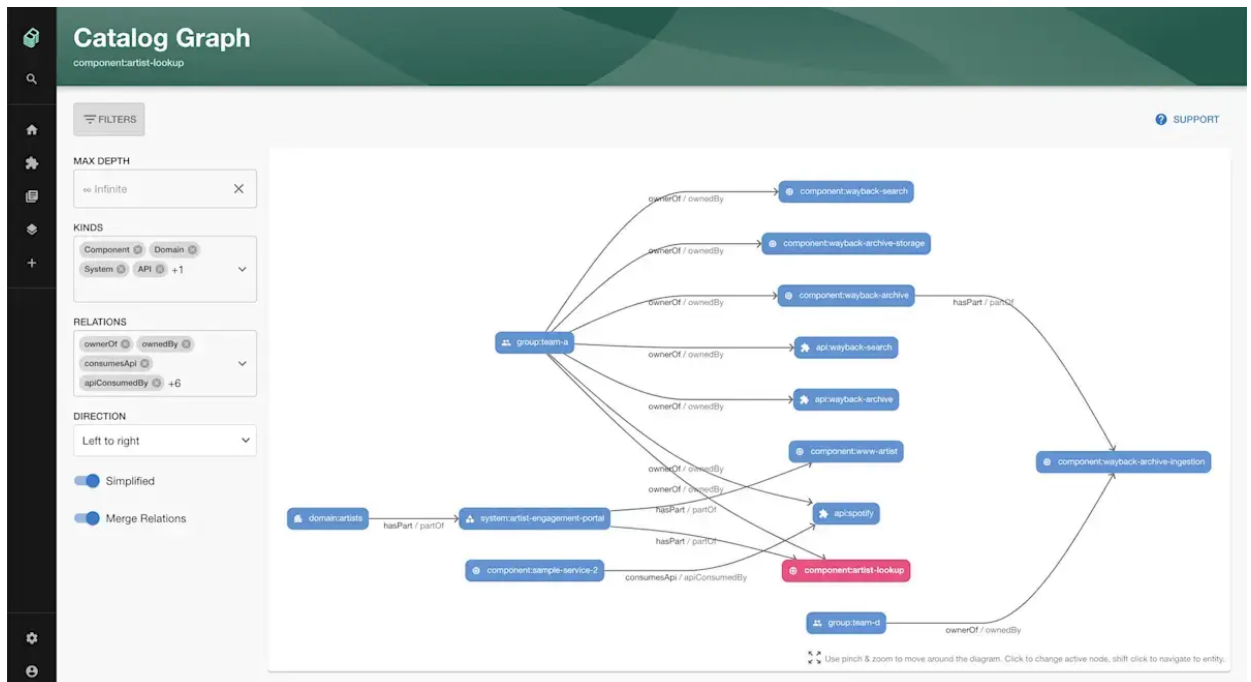


Рисунок 1.1 - Інтерфейс графіка залежностей у системі Backstage

Практичне використання цього плагіна виявило певні недоліки у його

архітектурі. До плюсів платформи відносять відкритий код і наявність активної спільноти незалежних розробників. Вони часто створюють нові корисні розширення. Проте первинне налаштування займає багато робочого часу. Для розгортання повноцінного середовища адміністраторам доводиться витратити ресурси на глибоке налаштування бази даних та ручну інтеграцію багатьох сторонніх сервісів.

Для безперервного спостереження за станом хмарних інфраструктур у реальному часі фахівці часто обирають комерційний модуль Service Map від сервісу Datadog. Завдяки внутрішнім механізмам збору телеметрії платформа сама формує детальну карту взаємодій між мікросервісами без втручання людини. Готову карту та її складові елементи наведено нижче на рисунку 1.2. Це значно полегшує подальший візуальний аналіз системи.

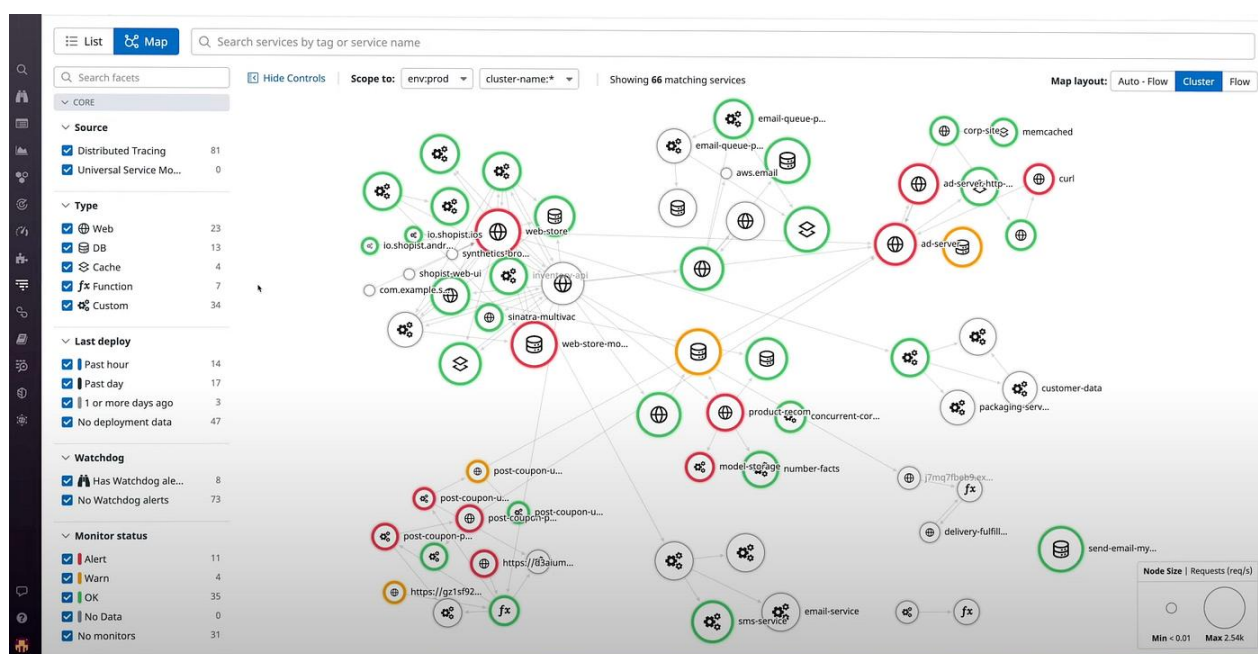


Рисунок 1.2 - Карта сервісів у платформі Datadog

Сильною стороною цього рішення є висока точність побудови топології. Програма розраховує її на основі аналізу обсягів мережевого трафіку між вузлами. Завдяки цьому оновлення графа відбувається динамічно. Незважаючи на потужний функціонал, інструмент орієнтується переважно на збір та показ метрик

продуктивності. З огляду на високу вартість підписки та складність інтеграції, для невеликих проєктів використання цієї системи часто виявляється економічно не вигідним.

У сфері управління корпоративною архітектурою широкого поширення набув інформаційний продукт під назвою LeanIX. Для побудови візуальних звітів про поточний стан інфраструктури компанії платформа використовує гнучкий конструктор діаграм зі значною кількістю фільтрів. Детальний вигляд цього інтерфейсу продемонстровано на рисунку 1.3. Загалом інтерфейс програми виглядає досить продуманим.

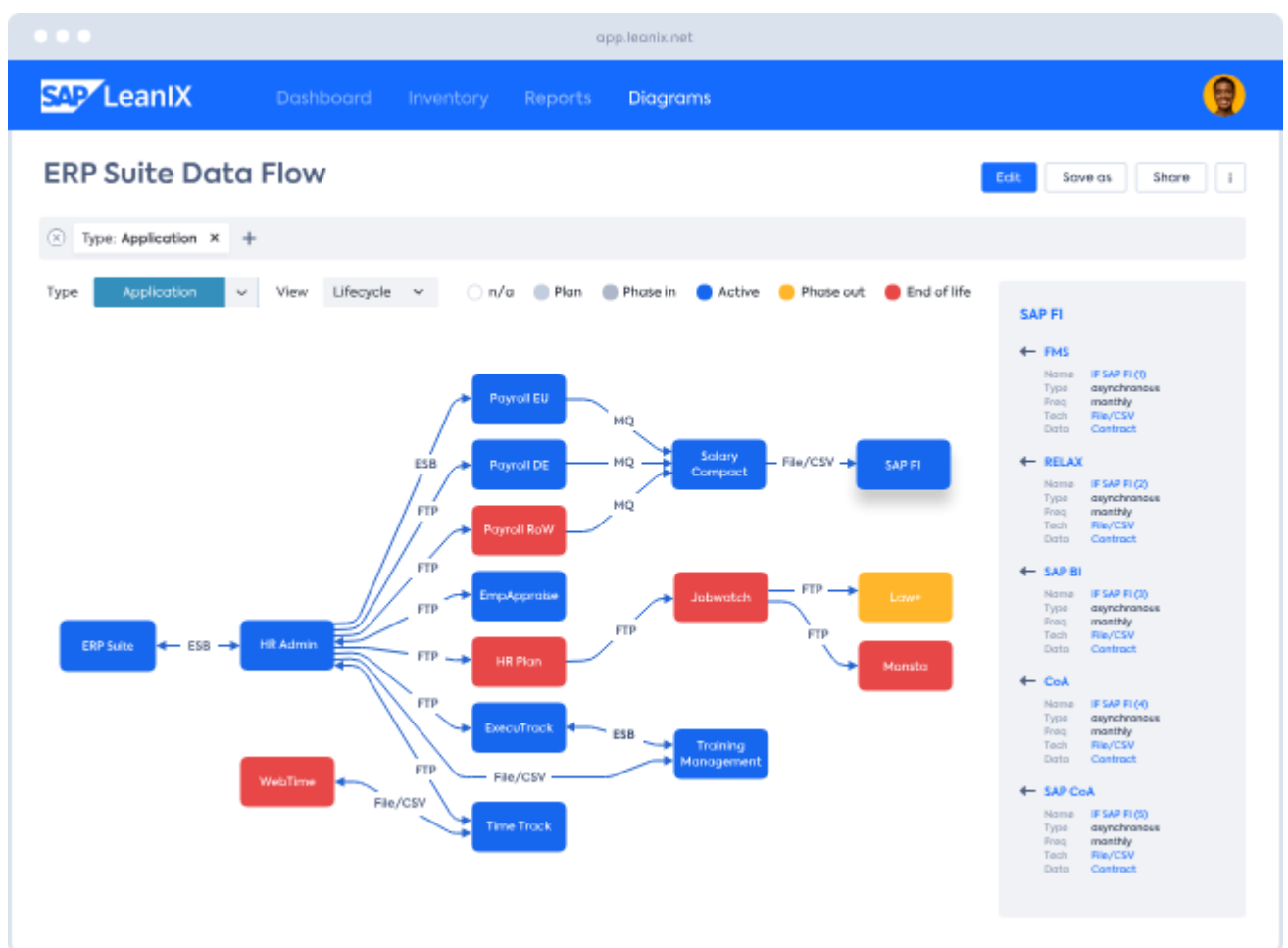


Рисунок 1.3 - Модуль візуалізації архітектури у системі

Через веб-інтерфейс фахівець переглядає наявні взаємозв'язки на різних рівнях логічної деталізації. До плюсів системи належить проведення глибокої

аналітики та наявність інструментів для планування структурних змін у ландшафті програмного забезпечення. На практиці для освоєння базового меню фахівцям доводиться проходити тривале навчання. Зважаючи на орієнтацію платформи на вирішення потреб бізнес-аналітиків, технічним фахівцям часто бракує функцій для показу інженерних деталей мікросервісів.

Проведений аналіз виявив потребу у створенні спеціалізованого додатка для хмарної екосистеми Atlassian Compass. Для усунення недоліків існуючих рішень наразі розробляється новий програмний модуль з фокусом на інтуїтивну зрозумілість інтерфейсу та високу швидкість роботи. Він працюватиме набагато ефективніше. На відміну від перевантажених функціями корпоративних платформ, запропонований інструмент забезпечить швидку візуалізацію архітектури через побудову дерева та формування матриці залежностей. Завдяки використанню безсерверної технології Atlassian Forge відпадає потреба у налаштуванні виділених серверів чи розгортанні складних реляційних баз. У процесі роботи додаток отримуватиме інформацію про компоненти безпосередньо через захищений GraphQL інтерфейс платформи без застосування посередників. Такий підхід гарантує повну сумісність з каталогом програмних компонентів користувача та зменшує навантаження на систему.

1.2 Аналіз платформи Atlassian та інфраструктури Forge

Компанія Atlassian пропонує набір інструментів для спільної роботи команд розробників. Продукти Jira та Confluence виступають основними елементами цієї платформи. Кожна програма виконує власну задачу, покращуючи процес написання коду. Розробники часто потребують додаткові можливості, які виходять за межі стандартного набору функцій.

Для розв'язання цих задач призначено платформу Forge. Інструментарій дозволяє створювати власні додатки без необхідності розгортання та тривалої підтримки окремих серверів. Інфраструктура для запуску додатків функціонує в середовищі Atlassian, що спрощує технічне обслуговування. Код, написаний на

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

Forge, отримує доступ до даних через захищені інтерфейси.

Окреме місце в цій екосистемі займає Compass [3]. Сервіс допомагає наводити лад в архітектурі великих програмних проєктів. Розробники часто втрачають контекст при роботі з десятками сервісів, тому Compass збирає всю розрізнену інформацію в єдиному місці для зручного доступу та швидкого аналізу.

Завдяки цьому сервісу фахівці отримують відповіді на такі питання:

- хто відповідальний за працездатність конкретного компонента?;
- де зберігається документація та де розгорнуто конкретну версію сервісу?;
- які технічні показники продуктивності демонструє вузол програми прямо зараз?

Платформа Compass використовує компоненти. Це логічні одиниці програмної архітектури, наприклад сервіс, бібліотека або інфраструктурний об'єкт. Додатки, створені на базі Forge, розширюють можливості Compass, дозволяючи відображати дані або додавати нові методи аналітики.

Команди розробників програмного забезпечення, технічні архітектори та фахівці DevOps використовують платформу Compass. Вони відстежують стан тисяч сервісів без потреби шукати розрізнену документацію. Програмні додатки для цієї системи дозволяють адаптувати робочий простір під потреби конкретної інженерної групи.

Використання Forge гарантує стабільну роботу таких доповнень, оскільки платформа бере на себе управління життєвим циклом коду [4]. Розробник пише логіку обробки даних, а система автоматично виділяє ресурси під навантаження. Процес створення нових інструментів стає швидким та не залежить від наявності власного серверного обладнання.

1.3 Технічне завдання

Підставами для проведення розробки являється індивідуальне завдання на кваліфікаційну роботу.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

Найменування теми кваліфікаційної роботи – Розробка клієнтської частини інтеграційного додатка для платформи Atlassian Compass “Compass-map”.

Виконавець - Кравченко Артем Андрійович.

1.3.1 Найменування та область застосування

Розроблюваний додаток має найменування Compass-map. Цей продукт проектується як спеціалізований інструмент для глибокого візуального аналізу компонентів усередині екосистеми Atlassian. Головною платформою для інтеграції обрано хмарне середовище Compass, оскільки саме цей продукт Atlassian активно вибудовує нові стандарти управління мікросервісною архітектурою сучасних компаній.

Область застосування додатка Compass-map охоплює команди розробників програмного забезпечення, системних архітекторів та інженерів DevOps. Основними користувачами продукту стануть технічні фахівці компаній, що використовують інструменти Atlassian для відстеження стану тисяч взаємопов'язаних сервісів.

Створення подібного візуального інструменту дозволить спростити розуміння складних потоків даних та залежностей, які виникають під час масштабування великих розподілених систем.

Додаток автоматично оброблятиме інформацію про зв'язки між окремими компонентами, надаючи наочну схему для прийняття технічних рішень.

Впровадження Compass-map у внутрішні процеси дозволить значно зменшити час на вивчення застарілих документальних описів інфраструктури, надаючи кожному розробнику актуальний вигляд поточної архітектури проекту. Продукт знайде широке застосування в середовищах, де підтримка цілісності системи вимагає постійного моніторингу статусу кожного окремого вузла та швидкого аналізу впливу змін на загальну структуру продукту.

Особливої актуальності Compass-map набуває для організацій, що переходять на модель самообслуговування розробників (Developer Self-Service) та

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

впроваджують концепції внутрішніх платформ розробки (Internal Developer Platforms - IDP). У таких середовищах швидкий доступ до інформації про власників сервісів (service ownership), їхні взаємозв'язки та поточний стан здоров'я (health status) стає критичним фактором безперервності бізнес-процесів. Завдяки інтеграції з хмарною екосистемою Compass, додаток виконуватиме роль єдиного джерела правди (Single Source of Truth) для візуалізації архітектурних метаданих, що спрощує комунікацію між крос-функціональними командами та запобігає накопиченню технічного боргу.

1.3.2 Призначення розробки

Головним завданням створюваного додатка Compass-map є надання командам розробників інтуїтивного інтерфейсу для дослідження архітектурних зв'язків, що існують усередині складних мікросервісних інфраструктур. Базовий функціонал стандартного переглядача залежностей, вбудованого в систему Atlassian Compass, не забезпечує достатньої глибини візуалізації для великих корпоративних проєктів. Під час роботи над масштабними системами розробники часто втрачають контекст через необхідність перемикатися між окремими сторінками компонентів, що значно сповільнює процес пошуку потрібного вузла.

Для вирішення цього завдання в основу додатка покладено реалізацію інтерактивного графа, що дозволяє виводити взаємозв'язки між усіма елементами в єдиному робочому просторі. Зокрема, передбачено впровадження функціональності матриці залежностей, де наочно відображатимуться всі прямі та непрямі зв'язки між сервісами. Це дасть змогу командам DevOps миттєво оцінювати масштаб змін та їхній вплив на стабільність усієї архітектури.

Практичне застосування Compass-map полягає у суттєвій оптимізації щоденної діяльності програмістів, адже за допомогою цього інструменту вдасться досягти таких результатів:

- швидке виявлення циклічних залежностей та архітектурних вузьких місць без необхідності вивчення численних текстових файлів конфігурації;

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

- надання розробникам актуального візуального представлення поточного стану системи, що замінює застарілу статичну документацію;
- спрощення процесу додавання нових членів в команду завдяки можливості детального вивчення архітектурної карти системи за мінімальний час;
- підвищення якості прийняття технічних рішень завдяки наявності повної та структурованої інформації про критичні вузли інфраструктури перед початком будь-яких серйозних змін у коді.

На рисунку 1.4 наведено спрощену схему взаємодії розробника з майбутнім інтерфейсом додатка, що ілюструє перехід від лінійного перегляду до комплексного візуального аналізу.

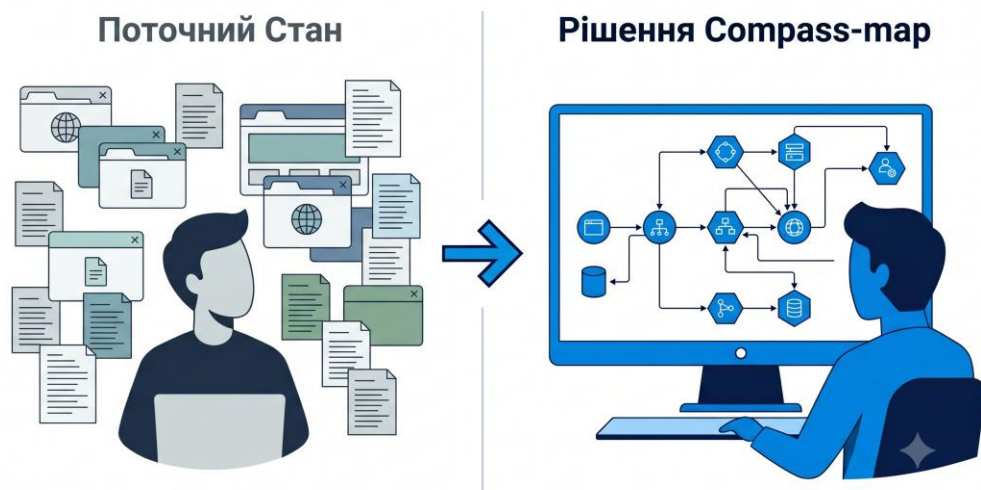


Рисунок 1.4 – Схема концептуальної взаємодії користувача з додатком

Розроблюваний інструмент перетворює хаотичні дані, що надходять через API платформи Compass, у впорядковану структуру, доступну для миттєвого розуміння. Використання сучасних підходів до візуалізації дозволить Compass-map стати невід'ємною частиною робочого процесу, повністю витіснивши стандартні методи аналізу залежностей. Завдяки реалізації логіки клієнтської частини на базі React, інтерфейс забезпечуватиме миттєвий відгук на дії користувача навіть при роботі з тисячами компонентів системи. У результаті інженерний склад отримає

змогу приділяти більше часу написанню якісного коду, замість витрачання ресурсів на тривале вивчення структури розгорнутих сервісів.

1.3.3 Вимоги до функціоналу додатку

Клієнтська частина програми повинна відповідати ряду технічних та користувацьких вимог. Використання сучасного стеку технологій дозволить реалізувати складну логіку інтерфейсу. Програма має виконувати наступні функції:

- візуальне відображення кожного компонента Compass у вигляді окремого вузла на карті. Кожен вузол має містити назву сервісу та його тип;
- малювання ліній зв'язку між компонентами. Лінії повинні показувати напрямок залежності від одного сервісу до іншого;
- реалізація механізму масштабування. Користувач може наближати карту для вивчення деталей або віддаляти її для перегляду загальної структури;
- можливість переміщення по полотну карти за допомогою миші. Це дозволяє працювати з великими графами, які не вміщуються на один екран;
- створення функції пошуку сервісів. При введенні назви у поле пошуку відповідний вузол повинен підсвічуватися або виділятися на карті;
- впровадження алгоритму автоматичного розташування. Програма має самостійно розраховувати позиції вузлів для отримання чистої та зрозумілої схеми;
- додавання режиму матриці залежностей. Це альтернативний спосіб перегляду даних у вигляді таблиці. Рядки та стовпці представляють сервіси, а позначки на перетині вказують на наявність зв'язку між ними;
- використання бібліотеки компонентів Atlaskit. Це гарантує, що кнопки, поля введення та меню будуть виглядати так само як і в інших продуктах Atlassian.

Для реалізації цих функцій обирається бібліотека для візуалізацій дерев. Вона надасть повний контроль над виглядом вузлів та ліній. Можна створювати власні шаблони для різних типів сервісів. Наприклад бази даних можуть виглядати інакше ніж веб-сервіси або бібліотеки. Це значно покращує читабельність карти.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						19
Зм.	Арк.	№ докум.	Підпис	Дата		

Також важливо забезпечити швидку роботу інтерфейсу. Навіть якщо в системі є понад двісті сервісів, карта повинна рухатися плавно без помітних затримок.

Окрім цього, важливою функціональною вимогою є забезпечення гнучких можливостей фільтрації та персоналізації відображення даних. Користувачі повинні мати змогу вибірково приховувати другорядні зв'язки або тимчасово ізолювати певні гілки графа залежностей для глибшого аналізу окремої підсистеми без зайвого візуального навантаження. Також передбачається інтеграція механізму збереження налаштувань інтерфейсу (зокрема масштабу, поточних координат полотна та активних фільтрів) у сесії користувача, що суттєво мінімізує витрати часу на підготовку робочого простору при повторному відкритті карти.

Взаємодія з серверною частиною має відбуватись через спеціальний міст Forge Bridge. Клієнтська програма робить запити до резолверів, які готує інший розробник. Після отримання списку компонентів клієнт повинен перетворити їх у формат об'єктів для бібліотеки візуалізації дерева. Кожен елемент отримує унікальний ідентифікатор та набір властивостей для малювання. Це дозволяє будувати карту динамічно на основі актуальних даних із хмари Atlassian.

1.3.4 Вимоги до програмної документації

У ході розробки необхідно створити набір документів для подальшої підтримки проекту. Описані архітектурні рішення, алгоритми обробки даних та принципи взаємодії між модулями React стають головним джерелом інформації для технічних фахівців які здійснюватимуть подальшу розробку додатку. Для успішного супроводу розробляється детальна документація.

В основу системи опису покладено перелік матеріалів, що фіксують повний життєвий цикл розробки продукту:

- розгорнуте технічне завдання із описом функціональних можливостей інтерфейсу;
- схема архітектури клієнтської частини, де висвітлено ієрархію React-компонентів;

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		20

- специфікація об'єктів даних для взаємодії з API платформи Atlassian.
- посібник для кінцевого користувача з описом роботи алгоритмів візуалізації;
- технічна інструкція з інсталяції та налаштування продукту в системі Atlassian;
- технічний звіт за результатами тестування графічних інтерфейсів на сумісність.

Розробка коду ведеться з урахуванням вимог до модульності та подальшого розширення структури інтерфейсу. Кожен функціональний блок залишається ізольованим, що спрощує процес тестування та зменшує кількість помилок. Обов'язковим етапом оформлення стає внесення інформації про призначення модуля та умови його використання у спеціальний заголовок файлу. Це дозволяє команді чітко розуміти вхідні та вихідні параметри, а також суворі обмеження роботи кожного вузла системи.

Коментування коду виконується з метою досягнення максимальної ясності логічних конструкцій для стороннього розробника. Якісні коментарі пояснюють не те, що робить функція, а те, чому обрано саме такий спосіб реалізації. Дотримання принципу єдиної точки входу та виходу для кожного процедурного блоку підвищує надійність додатка під час інтенсивного навантаження. Завдяки таким суворим правилам документації підтримується висока якість програмного продукту на кожному етапі експлуатації.

1.3.5 Техніко-економічні показники

Створення програмного продукту Compass-map вимагає планування та оцінки витрат трудових, часових і машинних ресурсів. Такий аналіз дозволяє визначити обсяг зусиль, потрібних для успішної реалізації клієнтської частини додатка в умовах робочого середовища платформи Atlassian.

Трудові витрати охоплюють ключові етапи проєктування та безпосереднього написання коду:

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

- аналіз архітектурних особливостей платформи Compass та вимог інтеграції через Forge;
- проектування ієрархічної структури графічного інтерфейсу та компонентів;
- реалізація механізмів побудови дерева залежностей на основі даних API;
- розробка логіки матриці зв'язків та її інтеграція в робочий простір;
- налаштування взаємодії з віддаленим сервером для отримання актуальних даних;
- проведення тестування, відлагодження графічних елементів та виправлення помилок;
- підготовка технічної документації для адміністраторів та кінцевих користувачів.

Загальний обсяг витрат становить орієнтовно 1.5–2 людино-місяці. Цей показник відповідає обсягу робіт, виконаних одним розробником над клієнтською частиною проєкту.

Деталізація трудомісткості за функціональними блоками дозволяє краще зрозуміти складність поставленого завдання. Розробка клієнтської частини, включно з версткою інтерфейсу віджета, алгоритмами візуалізації дерева залежностей та налаштуванням фільтрації матриці, вимагає близько 1–1.2 людино-місяця. Реалізація механізмів обробки запитів від серверної частини, що включає структурування вхідних масивів даних та їхню адаптацію для рендерингу, займає орієнтовно 0.3–0.4 людино-місяця. Етап тестування, перевірки коректності роботи графічних елементів у середовищі Atlassian та оптимізація швидкодії займає близько 0.2–0.4 людино-місяця.

Машинні витрати включають використання апаратного та програмного забезпечення, необхідного для розробки клієнтського віджета:

- робоча станція розробника з ОС Windows/Linux/macOS;
- середовище розробки Visual Studio Code;
- платформа Node.js та менеджер пакетів npm;

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

- система контролю версій Git та сервіс GitHub;
- платформа Atlassian Forge CLI для інсталяції та розгортання;
- веб-браузери Microsoft Edge та Mozilla Firefox для відлагодження інтерфейсу;
- хмарні ресурси Atlassian для моніторингу статусу та тестування розгорнутого додатка.

Орієнтовний обсяг машинного часу для вирішення поставлених задач становить 100–180 годин. Основна частина цього часу витрачається на програмування логіки рендерингу, тестування запитів до API через інструменти Forge та перевірку візуалізації на великих обсягах даних.

Результати техніко-економічних показників проєкту наведено у графічній частині 2026.KBP.122.423.08.00.00 ТП.

1.3.6 Стадії та етапи розробки

Процес розробки додатку включає чітку послідовність етапів, спрямованих на створення надійного інструменту для інтеграції з платформою Atlassian. Логічна побудова роботи дозволяє забезпечити стабільність клієнтського інтерфейсу та його повну відповідність технічним вимогам компанії. Дотримання визначених стадій гарантує високу якість проєкту.

На початковому етапі здійснюється аналіз предметної області та формування вимог до майбутньої системи. Досліджуються функціональні особливості середовища Atlassian Compass та виявляються обмеження, що стосуються побудови візуальних карт залежностей. Визначаються ключові вимоги до швидкодії інтерфейсу та способів відображення великих обсягів мікросервісних даних. Формується технічне завдання.

Другий етап присвячено проєктуванню загальної архітектури та структури інтерфейсу додатка. Визначається ієрархія компонентів React та правила взаємодії з інфраструктурою платформи Forge. Проєктуються механізми отримання даних через API-запити, а також опрацьовуються схеми зберігання тимчасових станів для

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

забезпечення швидкого рендерингу графа залежностей. Створюються макети сторінок.

Під час третього етапу виконується безпосередня реалізація функціональних компонентів системи за допомогою сучасних інструментів розробки. Програмуються основні логічні модулі, включаючи:

- складні алгоритми обробки масивів компонентів системи та візуальної побудови дерева залежностей на основі отриманих від сервера даних;
- спеціальну логіку динамічного формування матриці зв'язків між сервісами, що дозволяє адміністраторам відстежувати критичні вузли інфраструктури;
- інструменти для фільтрації та сортування великих обсягів даних за заданими технічними параметрами для зручного відображення в інтерфейсі;
- графічні інтерфейсні елементи керування для взаємодії користувачів з віджетом безпосередньо в середовищі платформи Atlassian Compass.

Для забезпечення високої зручності інтерфейсу під час реалізації застосовуються сучасні підходи до стилізації та анімації. Використання CSS-модулів дозволяє ізолювати стилі кожного компонента, що запобігає конфліктам класів у великих проєктах. Для плавного переходу станів між режимами дерева та матриці інтегрується бібліотека Framer Motion, що забезпечує природну анімацію трансформації елементів. Покращення інтерфейсу включає впровадження системи адаптивних відступів на базі CSS Flexbox та Grid, а також налаштування колірної палітри згідно зі стандартами Atlassian Design System для створення безшовного візуального досвіду.

Цей етап завершується створенням стабільного прототипу, що повністю підтримує очікувані сценарії поведінки користувачів під час роботи з графічним інтерфейсом.

Четвертий етап присвячено детальному тестуванню розроблених рішень, яке охоплює всі ключові функціональні можливості додатка Compass-map. Після завершення базового етапу програмування виконується комплексне тестування,

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

яке спрямоване на виявлення можливих помилок у логіці клієнтської частини. Під час перевірки роботи дерева залежностей фахівці оцінюють точність відображення всіх вузлів архітектури та плавність їхньої візуалізації в робочому просторі програми. Особлива увага приділяється механізмам згортання та розгортання окремих елементів ієрархії, адже ці операції повинні виконуватися без помітної затримки рендерингу інтерфейсу React.

В режимі матриці проводиться ретельна перевірка сортування масивів даних, щоб переконатися, що кожен вузол займає чітко визначену позицію в загальній сітці.

Функція пошуку компонентів аналізується на предмет миттєвого оновлення результатів, що забезпечує значну зручність для користувача під час роботи зі складними структурами.

Операції зуму та скролу проходять тестування на стабільність при спробах користувача наблизити чи віддалити окремі фрагменти дерева.

Для переконання у функціональності програми тестувальники виконують такі кроки:

- Перевірка реакції на комбінації клавіш під час виконання операцій зуму, що дозволяє уникнути випадкових змін масштабування інтерфейсу та зберегти комфортний рівень деталізації;
- Тестування механізму скролу при відображенні надмірної кількості елементів для виявлення потенційних проблем із продуктивністю браузера та забезпечення плавного переміщення по графі;
- Випробування пошукового запиту, який повинен виділяти конкретні компоненти на фоні загальної схеми, не приховуючи при цьому контекст безпосередніх зв'язків.

Додатково проводиться кросбраузерна перевірка інтерфейсу, щоб виключити будь-які відмінності у відображенні елементів між Microsoft Edge та Mozilla Firefox. Усунення виявлених неточностей під час взаємодії з API-запитами дозволяє оптимізувати швидкість реакції інтерфейсу. Цей комплексний підхід до

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		25

перевірки дозволяє виявити неточності ще до моменту виходу програми, забезпечуючи високий рівень надійності в умовах реальних навантажень.

На завершальному п'ятому етапі здійснюється впровадження додатка в середовище Atlassian та підготовка технічної документації. Виконується фінальна конфігурація інструментів розгортання через Forge CLI, перевіряється документація API та створюється посібник користувача з детальними інструкціями. Результатом стає готовий до повноцінного використання програмний продукт.

1.3.7 Порядок контролю та прийому

Методика перевірки працездатності програмного додатка Compass-тар базується на послідовному виконанні тестів, які підтверджують відповідність клієнтської частини встановленим вимогам. Для проведення контролю створюються умови, які максимально наближені до реального робочого середовища в екосистемі Atlassian. Тестування виконується на робочих станціях, оснащених актуальними веб-браузером з підключенням до розгорнутого через Forge CLI екземпляра програми.

Для успішного проведення тестування встановлюються такі технічні вимоги:

- наявність активного облікового запису в системі Atlassian Compass;
- доступ до API серверної частини, яка розробляється паралельно;
- встановлене середовище виконання для Forge-додатків;
- використання актуальних версій браузерів Microsoft Edge або Mozilla Firefox.

Цикл перевірки функціональності моделює типові дії розробника, який використовує Compass-тар для аналізу архітектури сервісів. Сценарій тестування включає наступні кроки:

- завантаження клієнтського віджета в інтерфейсі платформи та перевірка

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		26

коректності відображення дерева залежностей;

- виконання операцій розгортання та згортання вузлів графа, оцінка швидкості відгуку інтерфейсу;
- перемикання в режим матриці та перевірка коректності відображення зв'язків між компонентами;
- застосування пошуку у таблиці матриці для вибірки окремих сервісів так само як і режимі дерева;
- перевірка роботи механізмів зуму та скролу в обох режимах відображення.

Система вважається успішно протестованою, якщо виконання всіх сценаріїв відбувається без критичних помилок. Важливим критерієм приймання є коректна візуалізація даних на великих масивах інформації та відсутність затримок під час анімації переходів між режимами. При виявленні відхилень розробник проводить аналіз причин, виконує виправлення та повторно запускає тестові сценарії.

Остаточне приймання роботи здійснюється комісією, до складу якої входять представники сторін. Порядок приймання включає такі етапи:

- виклад доповіді про результати розробки та відповідність клієнтської частини технічному завданню;
- демонстрація роботи додатку Compass-map у реальному часі;
- проведення контрольних випробувань функціоналу згідно з розробленими сценаріями;
- відповіді на запитання та обговорення зауважень комісії.

Для фінального приймання представляється робочий веб-додаток, розгорнутий у хмарному середовищі, пояснювальна записка та повний пакет вихідного коду. Успішне проходження комісії підтверджується відповідним актом приймання, що засвідчує готовність продукту до експлуатації.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		27

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка графічного інтерфейсу додатку

Під час проєктування додатку Compass-map було використано сучасну хмарну платформу Atlassian Forge. Вона надає розробникам можливість розгортати додатки безпосередньо в інфраструктурі вендора без необхідності налаштування власних серверів.

Для реалізації складного графічного інтерфейсу було обрано підхід Custom UI. Цей метод дозволяє використовувати стандартні веб-технології, включаючи React та швидкий інструмент збірки Vite. Завдяки цьому вдалося побудувати динамічне середовище.

Під час вибору типу сторінки для розміщення модуля перевага була надана Component Page. Цей вибір зумовлений потребою аналізувати дерево залежностей для конкретного обраного сервісу в системі Compass.

Платформа Atlassian пропонує декілька варіантів точок входу для додатків:

- admin Page - призначена для налаштування системних параметрів адміністраторами;
- component Page - відображається лише в контексті конкретного сервісу (компоненту);
- global Page - забезпечує доступ до функціоналу з головного меню системи.

Структурна схема інтерфейсу додатку зображена на графічній частині 2026.КВР.122.423.08.00.00 СС.

Організація вихідного коду клієнтської частини базується на принципах чіткого розділення відповідальності між окремими логічними модулями програми.

Структура папок проєкту включає головну директорію src із вкладеними папками для компонентів, констант та допоміжних функцій. На рисунку 2.1 представлено ієрархію файлів репозиторію.

					2026.КВР.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		28

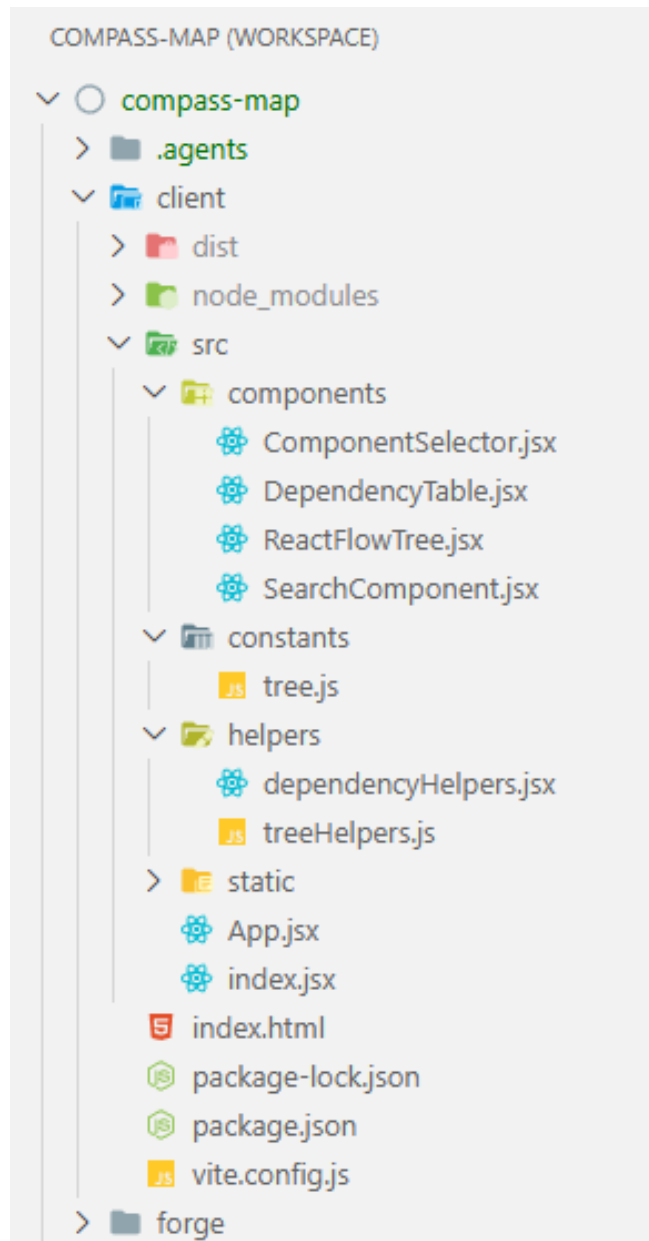


Рисунок 2.1 – Структура файлів та папок клієнтського проєкту

В основу архітектури візуальної частини покладено декларативний підхід популярної бібліотеки React версії 18 [5]. Застосування актуальних можливостей цієї бібліотеки, зокрема нових хуків конкурентного рендерингу, дозволяє значно прискорити відображення великих масивів даних. Налаштування збірки проєкту виконується через конфігураційний файл сучасного інструменту Vite [7], який визначає шляхи до ресурсів. Головним входом у додаток виступає файл index.jsx, який ініціалізує кореневий елемент через метод createRoot для рендерингу. Основна логіка управління внутрішнім станом програми зосереджена в

центральному модулі App.jsx. В ньому за допомогою стандартних хуків useState та useEffect виконується первинне отримання даних через міст Forge. Далі отримана інформація передається до дочірніх вузлів. Код залишається чистим. Повний лістинг коду головного модуля 'App.jsx', який відповідає за ініціалізацію стану додатку та маршрутизацію між режимами відображення, подано в додатку А пояснювальної записки.

Для кращого розуміння взаємодії всіх внутрішніх та зовнішніх модулів системи було розроблено загальну архітектурну модель. Структурна схема архітектури додатка зображена у графічній частині 2026.KBP.122.423.08.00.00 СС

Для створення інтерфейсу, який органічно виглядає в екосистемі продуктів Compass, було застосовано офіційну бібліотеку компонентів Atlassian Design System [8]. Використання готових пакетів @atlaskit забезпечує повну відповідність корпоративним технічним стандартам візуального оформлення всіх елементів.

Було використано готові кнопки, текстові поля, випадаючі списки та складні динамічні таблиці. Це значно прискорило процес верстки сторінок. Користувач отримує зручні якісні елементи інтерфейсу.

Центральний контейнер додатка організовує розміщення всіх елементів управління у верхній частині робочої області сторінки для зручного доступу.

Для вирівнювання блоків використано сучасні примітиви @atlaskit/primitives, такі як Box для відступів та Inline для рядкового розміщення. Це дозволило побудувати гнучку адаптивну сітку. Вона коректно підлаштовується под різні роздільні здатності екранів. Основна навігація в додатку реалізована кнопками.

Важливим елементом управління виступає перемикач режиму відображення, зображений на рисунку 2.2, розташований зліва на верхній панелі інструментів програми. Він побудований на базі елемента ButtonGroup із використанням двох окремих кнопок із різними візуальними станами. Кнопка «Дерево» активує складне графічне представлення вузлів, а кнопка «Матриця» перемикає весь інтерфейс у табличний вигляд. Поточний стан вибору зберігається

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

у змінній `viewMode`. Програма миттєво змінює вигляд екрана.

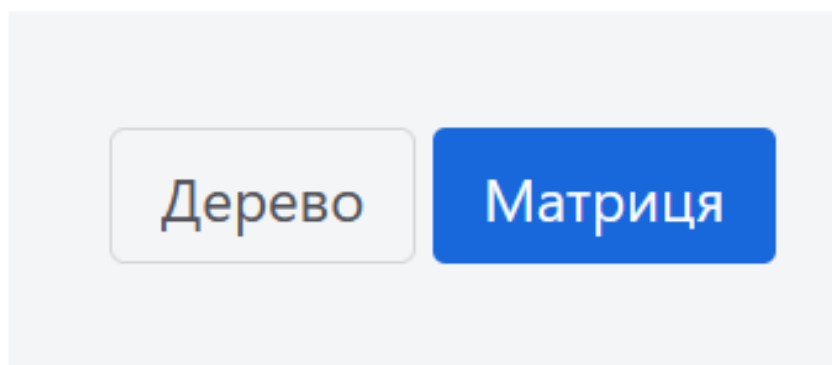


Рисунок 2.2 – Зовнішній вигляд перемикача режимів перегляду

Поле швидкого пошуку елементів інтегровано для забезпечення миттєвої навігації серед сотень сервісів та мікросервісних об'єктів інфраструктури. Технічно цей блок реалізовано в окремому файлі `SearchComponent.jsx` з використанням стандартного текстового поля `@atlaskit/textfield`.

Під час введення будь-яких символів спрацьовує системна подія `onChange`, яка автоматично оновлює стан фільтрації даних. Це допомагає знайти потрібний сервіс.

Результати пошуку підсвічуються синім кольором, поле пошуку зображено на рисунку 2.3.

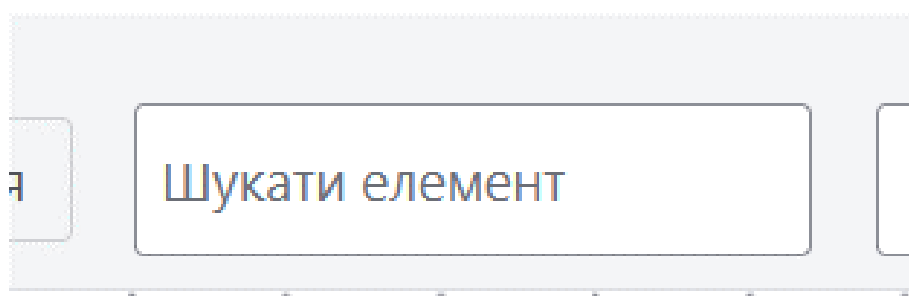


Рисунок 2.3 – Поле пошуку для введення текстового запиту

Фрагмент коду компоненту пошуку представлений в лістингу 2.1.

```

const SearchComponent = ({ value, onChange, placeholder = "Шукати елемент"
}) => {
  return (
    <Box className="search-container">
      <Textfield
        name="search"
        placeholder={placeholder}
        value={value}
        onChange={(e) => onChange(e.target.value)}
      />
    </Box>
  );
};

```

Лістинг 2.1 – Фрагмент код компоненту пошуку

Для вибору іншого кореневого елемента при побудові ієрархії залежностей використовується випадаючий список у модулі `ComponentSelector.jsx`. Цей інструмент базується на можливостях бібліотеки `@atlaskit/select`, яка підтримує вбудовану фільтрацію та асинхронну роботу зі списками.

Призначенням цього блоку є надання користувачеві можливості швидко змінити фокус дослідження поточної архітектури. Користувач може обрати будь-який компонент `Compass`. Дані завантажуються через метод `handleComponentChange`. Лістинг коду методу наведений у лістингу 2.2.

```

const handleComponentChange = (componentId) => {
  setSelectedComponentId(componentId);
  setIsLoading(true);
  invoke('getComponent', { componentId })
    .then((data) => { setOneComponent(data);
      setIsLoading(false); })
    .catch((err) => {console.error('Failed to fetch component',
err); setIsLoading(false);});});
};

```

Лістинг 2.2 – Метод заміни компоненту при виборі з випадаючого списку

Вигляд компоненту випадаючого списку зображено на рисунку 2.4.

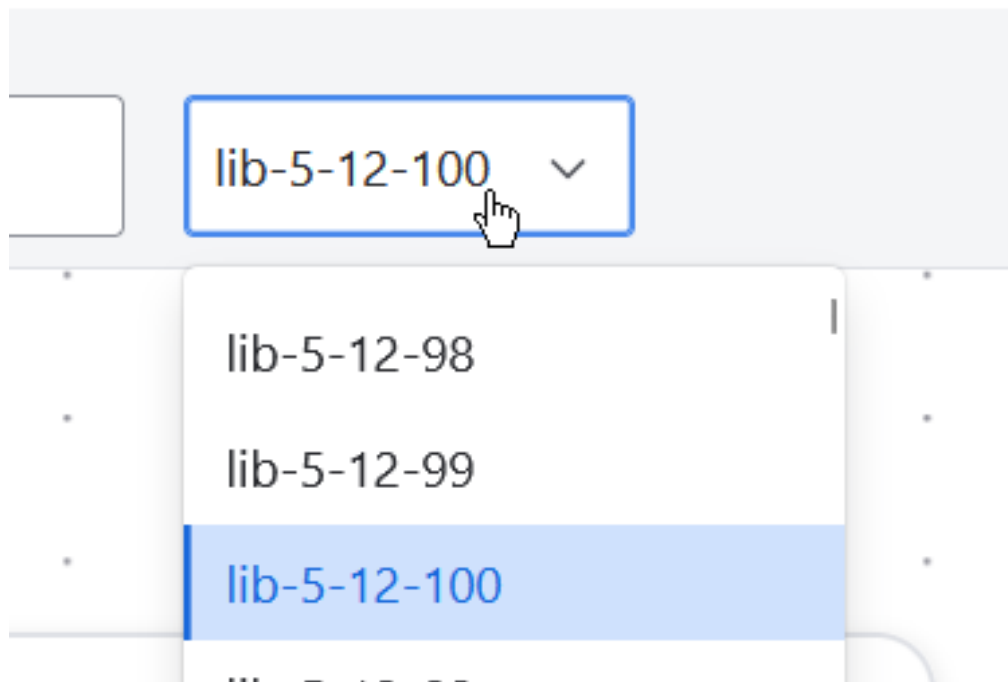


Рисунок 2.4 – Випадаючий список вибору компоненту

Програмна логіка взаємодії всіх частин інтерфейсу побудована на принципах одностороннього потоку інформації від батьківського компонента до дочірніх. Коли людина змінює значення в селекторі, функція ініціює асинхронний виклик `invoke('getComponent')` з пакета `@forge/bridge`. Це запускає запит до серверної частини для отримання структури зв'язків нового об'єкта. Після успішного завершення операції стан `isLoading` зміниться на `false`. Весь інтерфейс автоматично оновлюється.

На головному екрані в активному режимі дерева відображається складна ієрархічна структура всіх пов'язаних об'єктів обраної програмної системи. У альтернативному режимі матриці виводиться компактна сітка залежностей для проведення швидкого технічного аудиту всієї інфраструктури.

Структурна схема функцій додатку зображена на графічній частині 2026.KBP.122.423.08.00.00 СС.

Детальний опис алгоритмів автоматичної побудови графа та логіки формування таблиці наведено в наступних розділах. Там розкрито складні математичні методи. Це допоможе краще зрозуміти програму.

Розробка стилів для графічного інтерфейсу велася з використанням стандартних CSS-файлів, які імпортуються безпосередньо в JavaScript-файли компонентів. Для запобігання конфліктам імен у великих проєктах застосовано методику ізоляції стилів через специфічні селектори класів. Налаштування збірки проєкту виконується через конфігураційний файл vite.config.js, який визначає шляхи до ресурсів та параметри оптимізації коду. Завдяки використанню Fast Refresh у процесі розробки будь-які зміни в коді миттєво відображаються у вікні браузера. Це значно прискорює створення інтерфейсу. Система Forge Bridge забезпечує безпеку.

Такий підхід до побудови графічної частини дозволив створити масштабований інструмент, здатний працювати з великою кількістю архітектурних даних. Кожний елемент інтерфейсу виконує чітко визначену функцію, забезпечуючи високу інформативність та зручність для кінцевого користувача. Використання сучасного стеку технологій гарантує стабільність роботи додатка у хмарному середовищі Atlassian Compass протягом тривалого часу експлуатації. Програма відповідає всім технічним нормам.

2.2 Проєктування та створення дерева залежностей додатка

Для візуального представлення складних ієрархічних структур у додатку Compass-map було реалізовано динамічний граф. В основі логіки відображення лежить побудова розгалуженої мережі, яка дозволяє інженерам наочно простежити шлях проходження даних між окремими сервісами інфраструктури.

Під час формування графічної схеми кожен елемент системи стає окремим вузлом, а існуючі логічні зв'язки між ними перетворюються на лінії з'єднання. Така модель забезпечує швидке розуміння архітектури.

В основу технічної реалізації цього модуля покладено популярну бібліотеку

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						34
Зм.	Арк.	№ докум.	Підпис	Дата		

ReactFlow [9]. Цей інструмент надає потужний набір засобів для керування станом полотна, підтримки операцій масштабування та обробки подій миші. Оскільки стандартні методики рендерингу не дозволяють самостійно розраховувати оптимальні координати елементів, до проєкту було інтегровано бібліотеку Dagre [10]. Вона виконує функцію математичного ядра для автоматичного компоновання графа. Саме цей механізм запобігає перекриттю вузлів один одним під час завантаження великих обсягів даних. Координати обчислюються миттєво.

Процес формування візуальної структури починається з виконання коду, зосередженого в файлі treeHelpers.js. Зокрема, функція buildGraph відповідає за первинну обробку масивів інформації, що надходять від API Atlassian Compass. Вона рекурсивно проходить через кожен об'єкт, створюючи відповідні записи в мапах вузлів та ребер. Детальний лістинг коду допоміжного файлу treeHelpers.js наведено в додатку Б пояснювальної записки.

Наступна функція getLayoutedElements використовує алгоритми Dagre для точного розрахунку позицій на екрані. Її реалізація подана на лістингу 2.3.

```
export const getLayoutedElements = (nodes, edges, direction = 'TB') => {
  const isHorizontal = direction === 'LR'; dagreGraph.setGraph({ rankdir:
direction }); nodes.forEach((node) => {dagreGraph.setNode(node.id, { width:
NODE_WIDTH, height: NODE_HEIGHT });});
  edges.forEach((edge) => {
dagreGraph.setEdge(edge.source, edge.target);}); dagre.layout(dagreGraph);
  nodes.forEach((node) => {const nodeWithPosition =
dagreGraph.node(node.id);
    node.targetPosition = isHorizontal ? 'left' : 'top';
    node.sourcePosition = isHorizontal ? 'right' : 'bottom';
    node.position = {
      x: nodeWithPosition.x - NODE_WIDTH / 2,
      y: nodeWithPosition.y - NODE_HEIGHT / 2,
    }; return node;}); return { nodes, edges };;};
```

Лістинг 2.3 – Функція розрахунку координат вузла

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						35
Зм.	Арк.	№ докум.	Підпис	Дата		

Для управління станами відображення застосовано функцію `toggleVisibility`, яка дозволяє приховувати або показувати цілі гілки дерева залежно від дій користувача (повний лістинг коду допоміжного файлу `treeHelpers.js` наведено в додатку Б).

Для забезпечення візуальної стабільності інтерфейсу в системі зафіксовано базові геометричні константи. Зокрема, змінні `NODE_WIDTH` та `NODE_HEIGHT` визначають стандартні розміри прямокутних блоків для кожного сервісу. Параметр `DEFAULT_DIRECTION` встановлює напрямок росту дерева зліва направо, що є найбільш зручним форматом для читання технічних схем. Кожен вузол має чіткі межі.

Особливе значення в архітектурі має кастомний компонент `CustomNode`. Він розширює стандартні можливості `ReactFlow`, додаючи специфічні елементи для `Compass`. У середині цього блоку розміщено іконку типу сервісу, його офіційну назву та технічну категорію. Для з'єднання з іншими об'єктами реалізовано точки входу та виходу, так звані хендли (handles). Окремо впроваджено функціональну кнопку згортання, яка дозволяє ховати дочірні елементи для економії робочого простору. На рисунку 2.5 продемонстровано вигляд такого кастомного вузла.

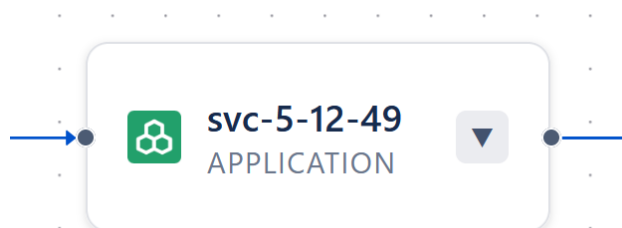


Рисунок 2.5 – Вигляд кастомного вузла графічного інтерфейсу

Повний лістинг коду компонента `ReactFlowTree.jsx`, який реалізує візуалізацію інтерактивного дерева та включає кастомний вузол `CustomNode`, подано в додатку В пояснювальної записки.

Під час створення візуального представлення було орієнтовано на принципи

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		36

мінімалістичного дизайну. Завдяки використанню асинхронного рендерингу дерево плавно оновлюється при зміні вхідних даних. Користувач може вільно переміщуватися по графі за допомогою курсору. Здійснювати масштабування колесом миші або кнопками в кутку зліва. Всі технічні деталі реалізації алгоритмів обробки подій дозволяють системі витримувати значні навантаження. Програма працює стабільно. На рисунку 2.6 можна спостерігати фінальний вигляд розгорнутого дерева залежностей із декількома рівнями вкладеності сервісів.

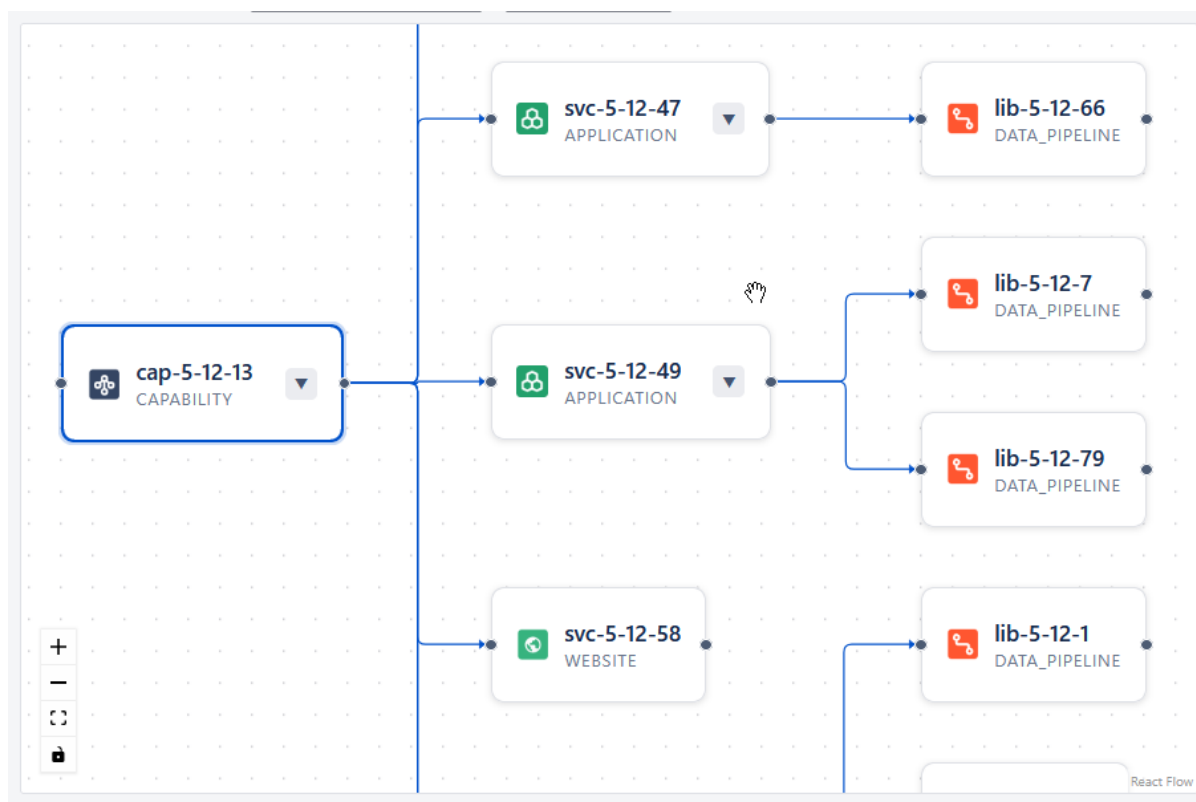


Рисунок 2.6 – Візуалізація багаторівневого дерева архітектурних зв’язків

Застосування бібліотеки ReactFlow у поєднанні з Dagre дозволило вирішити критичну проблему хаотичного розміщення елементів. Система автоматично адаптується під будь-яку кількість вхідних компонентів. В результаті отримано якісний інструмент для глибокого аудиту системи. Наявність інтерактивних елементів всередині кожного вузла підвищує загальну інформативність схеми. Динамічне дерево стає основою клієнтської частини додатка. Створений код легко масштабується.

2.3 Проєктування та створення матриці зв'язків компонентів

Альтернативним та дуже інформативним способом візуалізації архітектурних даних у додатку Compass-map є таблична матриця зв'язків. На відміну від ієрархічного дерева, цей режим допомагає одночасно аналізувати взаємодію між сервісами системи у одній структурі. Такий підхід стає корисним під час проведення технічних аудитів, коли інженеру потрібно швидко знайти відсутність зв'язків або виявити зайві з'єднання. В основу розробки цього модуля покладено створення цифрової сітки. За допомогою матриці відображається перетин компонентів.

Технічна побудова цього блоку базується на використанні компонента `DynamicTable` з бібліотеки `@atlaskit`. Цей інструмент працює дуже швидко навіть при відображенні великої кількості рядків та підтримує вбудоване прокручування. Для підготовки даних до виводу у файлі `dependencyHelpers.jsx` було написано алгоритм перетворення лінійних масивів об'єктів у двовимірну структуру. Зокрема, функція `processAllComponents` виконує сканування всіх отриманих від сервера сервісів, розподіляючи їх за технічними категоріями та модулями. Програмним шляхом рядки таблиці формуються динамічно. Повний лістинг коду файлу `dependencyHelpers.jsx` наведено в додатку Г пояснювальної записки.

Щоб зробити аналіз зручним, у матриці впроваджено візуальні індикатори стану. Кожна комірка на перетині стовпця та рядка містить позначку, яка підтверджує логічний зв'язок між елементами.

Додатково було розроблено алгоритм розрахунку рівня покриття залежностями, результати якого виводяться у верхній частині інтерфейсу. Це дозволяє керівникам проєктів швидко оцінювати густоту зв'язків сервісів у архітектурі.

На рисунку 2.7 продемонстровано загальний вигляд сформованої матриці.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						38
Зм.	Арк.	№ докум.	Підпис	Дата		

Дерево	Матриця	Шукати елемент					
Загальний відсоток залежностей: 1.8%			Всього Capabilities: 23		Всього Модулів: 192		
Модулі \ Capabilities	-5-12-1	cap-5-12-10	cap-5-12-11	cap-5-12-12	cap-5-12-13	cap-5-12-14	cap-5-12-15
svc-5-12-49					✓		
svc-5-12-5	✓						
svc-5-12-50							
svc-5-12-51							
svc-5-12-52							
svc-5-12-53			✓				
svc-5-12-54							
svc-5-12-55							✓
svc-5-12-56						✓	
svc-5-12-57							✓
svc-5-12-58					✓		
Покриття модулями (%)	1.6%	3.1%	1.6%	1.6%	3.6%	1.0%	1.6%

Рисунок 2.7 – Інтерфейс матриці зв'язків компонентів

Програмний код модуля `DependencyTable.jsx` містить логіку пошуку, що дозволяє користувачеві прибирати непотрібні дані за текстовим запитом. Під час введення назви сервісу у поле пошуку спрацьовує метод `useMemo`, який перераховує видимі рядки таблиці без перезавантаження сторінки браузера. Повний лістинг коду компоненту відображення матриці `DependencyTable.jsx` наведено в додатку Е пояснювальної записки.

На лістингу 2.4 наведено фрагмент функції розрахунку статистики.

```
const { capabilityUsage, moduleUsage, totalConnections } = useMemo(() => {
  const capUsage = {}; const modUsage = {}; let connections = 0;
  // ... логіка підрахунку активних зв'язків
  dependencies.forEach(dep => {
    const [mod, cap] = dep.split('->');
    if (capabilitiesMap.has(cap) && visibleOthers.has(mod)) {
      capUsage[cap] = (capUsage[cap] || 0) + 1;
      connections++;
    }
  });
  return { capabilityUsage: capUsage, totalConnections: connections };
}, [dependencies, capabilitiesMap, sortedOthers]);
```

Лістинг 2.4 – Фрагмент коду розрахунку статистики використання компонентів

Досить висока стійкість до великих обсягів інформації є важливою рисою матриці. Завдяки фіксованим заголовкам та першій колонці розробники можуть гортати таблицю, не втрачаючи назв сервісів. Весь процес сортування виконується клієнтом, що розвантажує сервер та забезпечує миттєвий відгук на дії користувача. Клієнтська частина обробляє дані самостійно.

Зокрема, було реалізовано механізм підсвічування порожніх зон, де очікувалися зв'язки, але вони не були виявлені системою Compass. На рисунку 2.8 можна спостерігати роботу вбудованого пошуку, які звужують область аналізу.

Модулі \ Capabilities	cap-5-12-17	cap-5-12-18	cap-5-12-19	cap-5-12-2	cap-5-12-20	cap-5-12-3	cap-5-12-4
svc-5-12-3			✓				
svc-5-12-30					✓		
svc-5-12-31	✓						
svc-5-12-32							
svc-5-12-33			✓				
svc-5-12-34				✓			
svc-5-12-35						✓	
svc-5-12-36							
svc-5-12-37			✓				
svc-5-12-38							
svc-5-12-39						✓	
Покриття модулями (%)	9.1%	0.0%	27.3%	9.1%	9.1%	18.2%	0.0%

Рисунок 2.8 – Результат застосування пошуку у режимі матриці

У поєднанні графічне дерево та аналітична матриця роблять Compass-map корисним інструментом. Кожен режим доповнює інший, створюючи картину стану програмного продукту. Через бібліотеку React реалізація табличного вигляду дозволила досягти гнучкості налаштувань інтерфейсу. Поточна система працює швидко. Всі технічні рішення щодо побудови даних у матриці дозволяють розвивати додаток при зростанні кількості сервісів у компанії. Створена програма дотримується стандартів розробки Atlassian.

2.4 Реалізація механізмів отримання інформації від віддаленого сервера

Для стабільного функціонування клієнтського інтерфейсу Compass-мар було розроблено надійну систему обміну даними між браузером та серверною частиною. Оскільки розробка бекенда виконується іншим фахівцем, основна увага в цій роботі зосереджена на правильному формуванні вихідних запитів та ефективній обробці вхідних пакетів.

Для забезпечення безпеки передачі інформації всередині інфраструктури Atlassian застосовується спеціалізований міст зв'язку Forge Bridge. Саме цей механізм дозволяє виконувати асинхронні виклики до хмарних функцій, минаючи складні налаштування політик безпеки браузера. Запити виконуються прозоро.

Архітектура додатку зображена на графічній частині (2026.KBP.122.423.08.00.00 АД). На ній відображено зв'язок клієнтської частини з сервером.

Процес отримання архітектурних даних починається в момент ініціалізації головного компонента системи. У коді файлу App.jsx реалізовано метод invoke, який надсилає технічний ідентифікатор потрібного сервісу на віддалений вузол обробки. Після успішного завершення операції сервер повертає структурований об'єкт у форматі JSON. Кожна така відповідь містить повний набір атрибутів, необхідних для візуалізації, включаючи унікальні ідентифікатори об'єктів, їхні технічні типи та масиви існуючих зв'язків. Формат даних залишається строгим. Лістинг коду контролера запитів (резолвера) component-page.js, який обробляє ці виклики на серверній частині, наведено в додатку Є пояснювальної записки.

Типова структура вхідного масиву інформації про архітектуру сервісу виглядає так:

- id - унікальний ключ елемента в базі даних Compass;
- name - офіційне найменування сервісу для відображення в інтерфейсі;
- typeId - технічна категорія об'єкта (наприклад, SERVICE або CAPABILITY);

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						41
Зм.	Арк.	№ докум.	Підпис	Дата		

– relationships - вкладений список об'єктів, що вказують на існуючі залежності.

На рисунку 2.9 наведено діаграму послідовності, яка ілюструє повний цикл проходження мережевого сигналу від дії користувача до моменту оновлення графічного екрана.

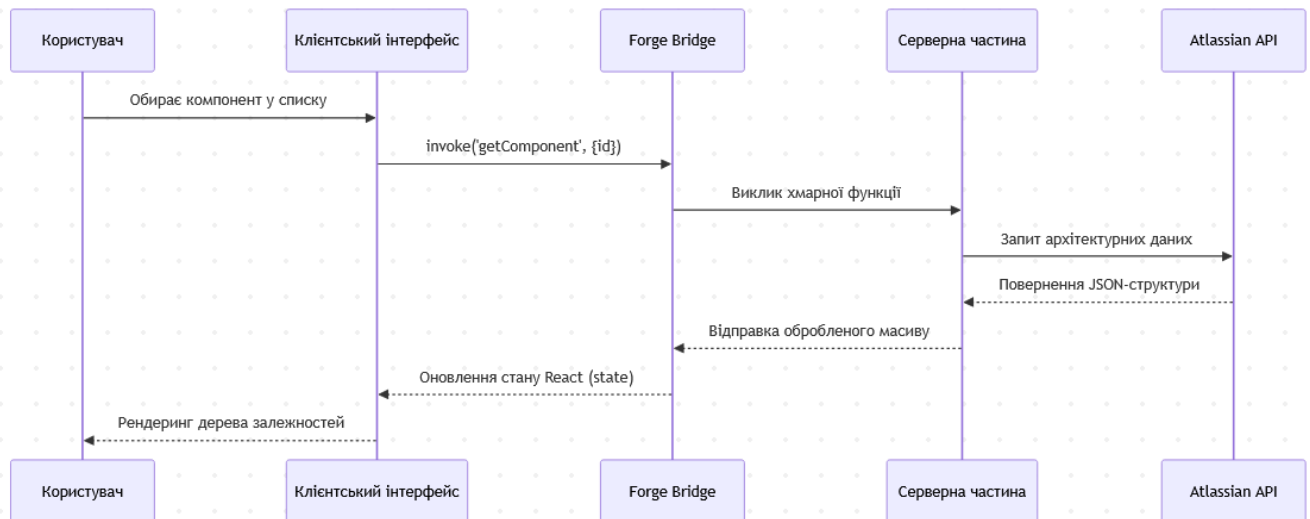


Рисунок 2.9 – Діаграма послідовності обміну даними

Під час розробки було приділено значну увагу механізмам запобігання помилкам при роботі з нестабільною мережею. У коді застосовано ланцюжки обробки обіцянок (Promises) із використанням методу .catch для перехоплення виняткових ситуацій. Якщо сервер не повертає дані вчасно, інтерфейс автоматично виводить інформаційне повідомлення про помилку завантаження. Це дозволяє користувачеві зберегти контроль над програмою.

Отримані масиви даних зберігаються у внутрішньому стані React за допомогою хука useState. Це забезпечує реактивне оновлення всіх дочірніх компонентів, включаючи дерево залежностей та аналітичну матрицю, відразу після завершення мережевого запиту.

Завдяки такому підходу вдалося досягти високої швидкості роботи додатку навіть при значній кількості мікросервісів у системі. Програма швидко парсить великі JSON-файли. Ефективність обміну даними підтверджена результатами

практичного використання інструменту розробниками. Проєкт повністю інтегрований у хмару.

Формат даних JSON-відповіді від серверної частини програми. Зображена на графічній частині (2026.KBP.122.423.08.00.00 БД) діаграма формату даних відображає структуру даних отримуваних від сервера.

Використання асинхронної моделі взаємодії дозволило розділити процеси відображення інтерфейсу та очікування інформації. Користувач не відчуває блокування екрана під час тривалих запитів до бази даних Atlassian.

Впроваджена схема обміну даними є оптимальною для хмарних архітектур сучасного типу. У процесі роботи додаток отримуватиме необхідну інформацію про компоненти через захищений GraphQL інтерфейс хмарної платформи [6]. Клієнтська частина працює надійно. Всі вимоги щодо інтеграції з Forge повністю виконані.

2.5 Тестування додатку

Завершальним етапом розробки клієнтської частини додатка Compass-map стала комплексна перевірка працездатності всіх функціональних блоків. Для забезпечення високої якості програмного продукту було застосовано комбінований підхід, що включає як стандартні методи верифікації коду, так і сценарії ручного тестування інтерфейсу.

Під час проведення випробувань особлива увага приділялася стабільності рендерингу при значних обсягах вхідних даних, оскільки реальні корпоративні архітектури можуть налічувати сотні взаємопов'язаних сервісів.

Процес тестування розпочався з аналізу продуктивності системи при роботі з великою кількістю компонентів. Було встановлено, що використання бібліотеки ReactFlow у поєднанні з алгоритмами Dagre дозволяє підтримувати плавність роботи інтерфейсу навіть при відображенні понад 500 вузлів одночасно. Однак під час випробувань було виявлено певні технічні обмеження додатку. Зокрема, при екстремальному навантаженні (понад 2000 елементів) спостерігається незначне

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						43
Зм.	Арк.	№ докум.	Підпис	Дата		

зростання часу первинного розрахунку координат вузлів. Це зумовлено однопотоковою природою обчислень у браузері, що було зафіксовано як область для майбутнього впровадження Web Workers.

Для підвищення надійності в коді реалізовано багаторівневу обробку помилок. У разі виникнення збоїв під час мережевого запиту або отримання некоректної JSON-структури, система не припиняє роботу, а виводить інформаційне сповіщення через компоненти Atlassian Flag.

Окремо досліджено випадки, коли шукані компоненти відсутні в базі даних або не мають жодних зв'язків. У таких ситуаціях програма відображає спеціальний стан «Empty View», що запобігає появі порожнього білого екрана та зберігає навігаційні елементи доступними для користувача.

Повідомлення про відсутність шуканого компонента зображена на рисунку 2.10.

Дерево Матриця Яблуко

Загальний відсоток залежностей: 0% Всього Capabilities: 23 Всього Модулів: 0

Модулі \ Capabilities	cap-5-12-1	cap-5-12-10	cap-5-12-11	cap-5-12-12	cap-5-12-13	cap-5-12-14	sa
Немає даних для таблиці							

Рисунок 2.10 – Вивід повідомлення про відсутність шуканого компонента

Важливою частиною перевірки стало ручне тестування, яке виконувалося шляхом послідовного здійснення таких дій розробником:

- вибір початкового компонента через випадаючий список та ініціалізація завантаження даних;
- перегляд згенерованого дерева залежностей із візуальним контролем правильності всіх ліній зв'язку;

- перемикання в режим матриці для оцінки коректності табличного представлення ієрархії;
- виконання текстового пошуку сервісу, що супроводжується плавною анімацією переміщення камери до шуканого вузла;
- перевірка наявності яскравого підсвічування знайденого об'єкта синім кольором для швидкої ідентифікації;
- тестування роботи кнопки згортання та розгортання вузлів для керування деталізацією схеми архітектури;
- вибір іншого кореневого компонента для підтвердження повної очистки попереднього стану та коректного перемальовування графа.

На рисунку 2.11 продемонстровано візуальний відгук системи під час активації функції підсвічування знайденого елемента.

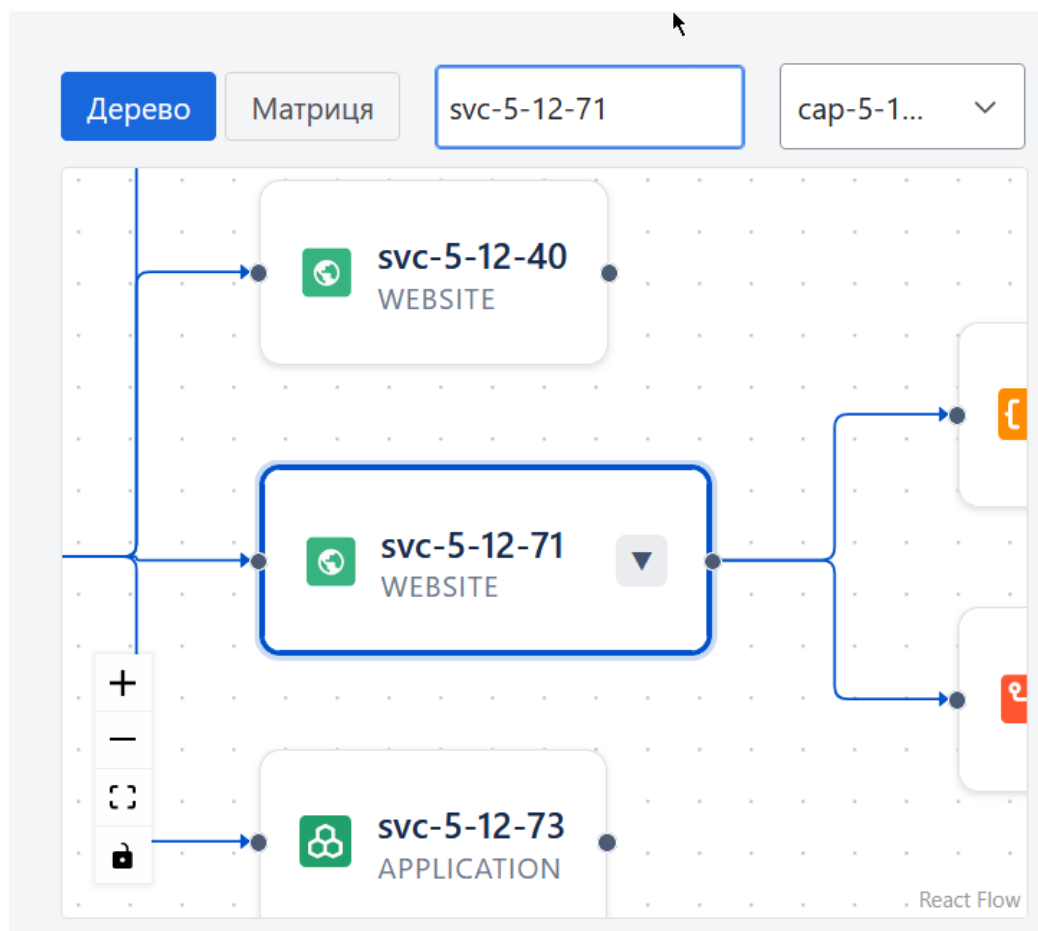


Рисунок 2.11 – Підсвічування компонента в результаті успішного пошуку

Додатково було запропоновано та проведено стрес-тести на переривання мережевого з'єднання в момент активного рендерингу. Система успішно обробила розрив сесії, зберігши локальні дані та надавши можливість повторного запиту після відновлення зв'язку. Результати всіх етапів перевірки зведено в загальну таблицю 2.1, що підтверджує готовність клієнтської частини до експлуатації.

Таблиця 2.1 – Результати функціонального тестування компонентів інтерфейсу

Назва тесту	Опис перевірки	Статус
Initial Load	Перевірка завантаження початкового дерева	Успішно
Toggle View	Перемикання між деревом та матрицею	Успішно
Search Zoom	Плавне наближення до знайденого вузла	Успішно
Node Collapse	Приховування дочірніх елементів гілки	Успішно
Filter Matrix	Фільтрація рядків у табличному режимі	Успішно
Error Catching	Обробка 500-ї помилки від сервера	Успішно

Завдяки ретельному підходу до тестування вдалося досягти стабільної поведінки всіх інтерактивних елементів. Клієнтська програма відповідає встановленим технічним нормам та демонструє високу відмовостійкість.

Використані методики дозволили виявити та виправити дрібні неточності у логіці анімацій ще до етапу фінальної презентації проєкту. Програмний продукт повністю виконує покладені на нього завдання.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції додатку в системі Atlassian

Для успішного розгортання програмного продукту Compass-map у системі Atlassian Compass необхідно виконати чітку послідовність технічних дій. На етапі активної розробки та проведення випробувань інсталяція здійснюється безпосередньо з вихідних файлів проєкту через командний рядок операційної системи.

Після виходу офіційного релізу в глобальному магазині Atlassian Marketplace цей процес суттєво спроститься для кінцевих споживачів. Тоді адміністратори зможуть додати новий інструмент у свій робочий простір лише одним натисканням кнопки на офіційній сторінці продукту. Завантаження потрібних модулів відбуватиметься повністю автоматично.

Зараз для ручного встановлення виконайте наступні кроки:

– Зареєструйте новий обліковий запис розробника на офіційному порталі Atlassian Developer, або увійдіть в існуючий. Перейдіть у власні налаштування профілю до розділу «Токени API» та згенеруйте унікальний ключ доступу. Обов'язково збережіть цей секретний рядок у надійному місці, адже він знадобиться для виконання подальшої авторизації через консоль. На рисунку 3.1 продемонстровано інтерфейс керування ключами безпеки;

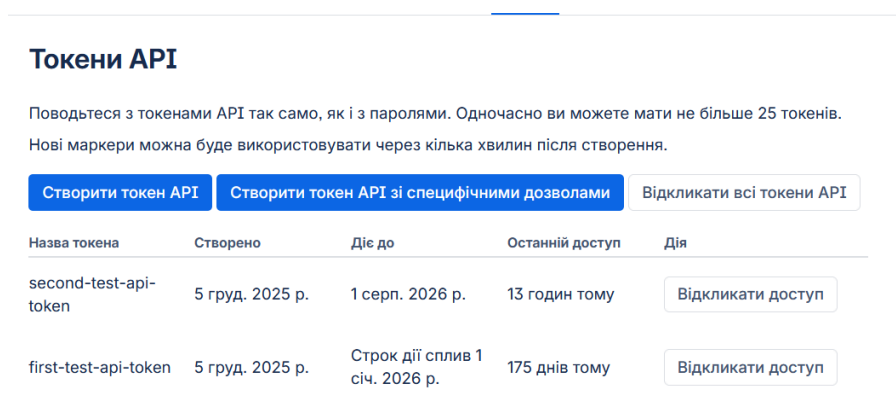


Рисунок 3.1 – Інтерфейс створення токена API для розробника

- встановіть інтерфейс командного рядка Forge CLI на ваш локальний робочий комп'ютер. Для запуску процесу інсталяції виконайте у терміналі таку коротку команду «`npm install -g @forge/cli`»;
- авторизуйтеся у хмарній системі через відкрите вікно терміналу за допомогою вашої робочої електронної пошти та згенерованого раніше унікального токена. Для цього введіть команду «`forge login`» та чітко дотримуйтесь вказівок на моніторі. Вікно командного рядка з успішно завершеним процесом авторизації подано нижче на рисунку 3.2;

```

mpass-map\compass-map> forge login
Log in to your Atlassian account
Press Ctrl+C to cancel.

? Enter your email: [redacted]@gmail.com

Next, enter your Forge CLI scoped token. If y
atlassian.com/forg-cli-api-token

? Enter your Atlassian API token: [hidden]

√ Logged in as Artem Kravchenko.

Now try forge create to start a new app.
PS C:\Users\Artem\Documents\Учеба 03 ТФК-ТНТУ
mpass-map\compass-map>

```

Рисунок 3.2 - Успішна авторизація через консоль

- завантажте повний вихідний код програми Compass-map у порожню директорію та за допомогою терміналу перейдіть до неї. Тепер встановіть усі базові залежності для роботи серверної частини додатку. Для цього перейдіть у папку `/forge` та виконайте вказану системну команду: «`npm install`»;
- зареєструйте створений додаток у хмарній інфраструктурі компанії Atlassian для миттєвого отримання власного унікального ідентифікатора. Виконайте команду реєстрації `forge register` та оберіть технічну назву для вашого

нового проєкту. Процес автоматичної генерації ідентифікатора зображено на рисунку 3.3;

```
PS C:\Users\Artem\Documents\Учеба 03 ТФК-ТНТУ Курс 4 Семестр 2\Дипломна
робота Compass-map\compass-map\forge> forge register

Press Ctrl+C to cancel.

You are about to assign your app to a Developer Space. Once you assign y
our app to the selected Developer Space, all usage and billing for the a
pp will be managed through this space.

Please select the Developer Space you want to assign your app to. To lea
rn more about your Developer Space, go to: https://developer.atlassian.com/platform/forge/developer-space/developer-spaces-introduction

? Select or create a Developer Space: test app 2 2025-12-23

✓ You are assigning your app to this Developer Space: test app 2 2025-12
-23.

Registering the app to you.

Name your app. The app name can include dashes, spaces, and underscores.

? Enter a name for your app: Compass-map
```

Рисунок 3.3 –Реєстрація додатку

– відкрийте конфігураційний файл проєкту у редакторі коду. Введіть отриманий під час виконання попереднього кроку унікальний ідентифікатор APP_ID у спеціально призначене для нього поле файлу .env, що містить змінні середовища. Ретельно перевірте правильність написання всіх символів та збережіть оновлений файл;

– після налаштування середовища перейдіть до директорії клієнтської частини /client для підготовки графічного інтерфейсу. Виконайте встановлення бібліотек та здійсніть фінальну збірку статичних ресурсів. Для цього послідовно запустіть такі команди: «npm install», «npm run build». Результат успішної збірки клієнтської частини проєкту зображено на рисунку 3.4;

```
C:\Users\Artem\Documents\Учеба 03 ТФК-ТНТУ Курс 4 Семестр 2\Дипломна робота Додатки Compass\compass-map\client>npm run build

> component-page-ui@1.0.0 build
> vite build

vite v5.4.21 building for production...
✓ 1 modules transformed.
dist/index.html 0.35 kB | gzip: 0.22 kB
✓ built in 55ms

C:\Users\Artem\Documents\Учеба 03 ТФК-ТНТУ Курс 4 Семестр 2\Дипломна робота Додатки Compass\compass-map\client>
```

Рисунок 3.4 – Результат успішної збірки фронтенд-модуля

- поверніться до робочої папки /forge та виконайте повне розгортання коду в хмарі Atlassian. Використовуйте для цього консольну команду «forge deploy». Ця операція пакує всі ваші файли та відправляє їх на віддалений сервер вендора. Система миттєво підтвердить успішне завершення операції;
- інсталюйте щойно розгорнутий додаток у ваш обраний робочий простір всередині платформи Atlassian Compass. Для цього введіть команду «forge install» та оберіть потрібний робочий сайт із запропонованого консоллю списку. Командний рядок із фінальним процесом інсталяції додатку та переліком наданих прав зображено на рисунку 3.5;

```
compass-map\compass-map\forge> forge install
Select an Atlassian app or platform tool.

? Select an Atlassian app or platform tool: Compass

Enter your site. For example, your-domain.atlassian.net

? Enter the site URL: artemkravchenko5000.atlassian.net

Installing your app onto an Atlassian site.
Press Ctrl+C to cancel.

Your app will be installed with the following scopes:
- read:component:compass
- write:component:compass

Your app will exchange data with the following urls:
- compass-ui.prod-east.frontend.public.atl-paas.net

? Do you want to continue? Yes

x Installing your app in Compass on to development...
```

Рисунок 3.5 – Завершення процесу інсталяції в цільовому просторі

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						50
Зм.	Арк.	№ докум.	Підпис	Дата		

Ретельно переконайтеся у повній відсутності повідомлень про критичні помилки в робочому вікні консолі.

Після успішного завершення всіх етапів налаштувань розроблений додаток Compass-mp відразу з'явиться в графічному інтерфейсі на сторінці компоненту Compass. Ви побачите нову вкладку у боковому меню навігації зліва. Програма повністю готова до роботи.

3.2 Інструкція з використання додатку

Розроблений інструмент аналізу архітектури інтегрується безпосередньо у звичний робочий простір платформи Atlassian. Після успішного розгортання додатку фахівці компанії отримують зручний доступ до візуальної схеми сервісів. Для початку роботи не потрібно завантажувати сторонні програми або проходити складну процедуру налаштування. Графічний інтерфейс створено інтуїтивно зрозумілим. Для запуску та подальшого використання додатку виконайте наведені нижче дії:

- відкрийте головну сторінку хмарного середовища Atlassian Compass у своєму веб-браузері. Знайдіть розділ «Компоненти» на верхній панелі, перейдіть в нього, кнопка розділу компонентів зображена на рисунку 3.6;

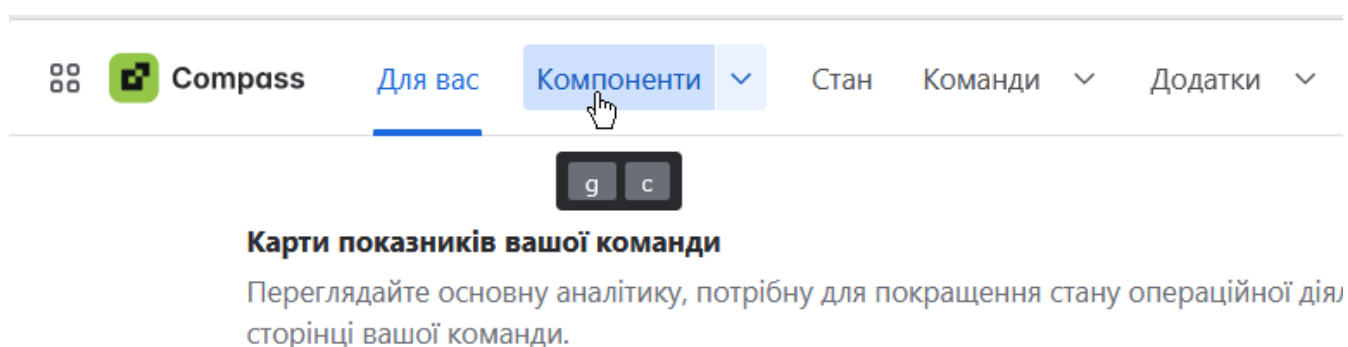


Рисунок 3.6 – Перехід на розділ вибору компонентів

- далі зі всіх наявних сервісів та оберіть потрібний базовий компонент із загального списку;

– на відкритій сторінці обраного об'єкта перейдіть до бокового меню навігації. Там натисніть на вкладку з назвою додатка Compass-map для його запуску, розміщення кнопки відкриття додатку зображено на рисунку 3.7;

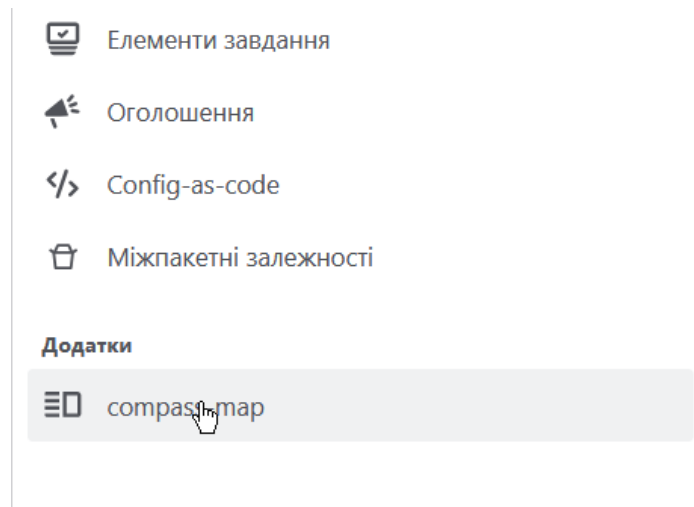


Рисунок 3.7 – Кнопка запуску додатку на панелі

– після запуску додатку, перед вами відразу з'явиться головний екран. За замовчуванням програма завантажує режим ієрархічного дерева залежностей для обраного сервісу. Вікно додатку зображене на рисунку 3.8;

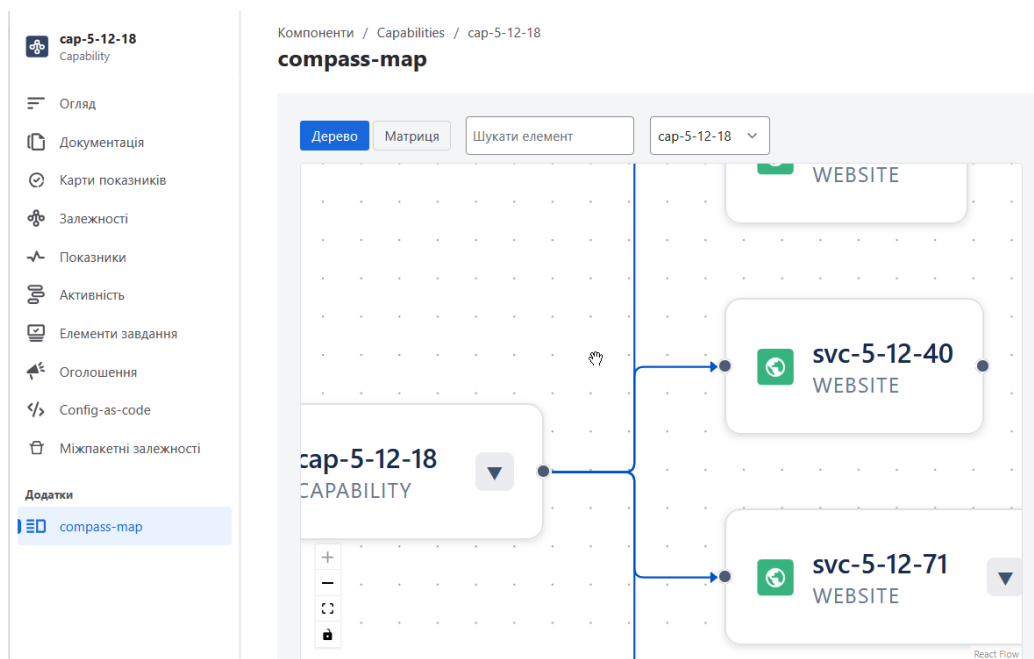


Рисунок 3.8 - Головний екран додатка в режимі дерева залежностей

– Ви можете вільно переміщуватися по полотну за допомогою затиснутої лівої кнопки миші, керувати масштабом використовуючи кнопки розміщені з лівої сторони знизу або використовуючи колесо миші, кнопки для керування масштабом зображені на рисунку 3.9;

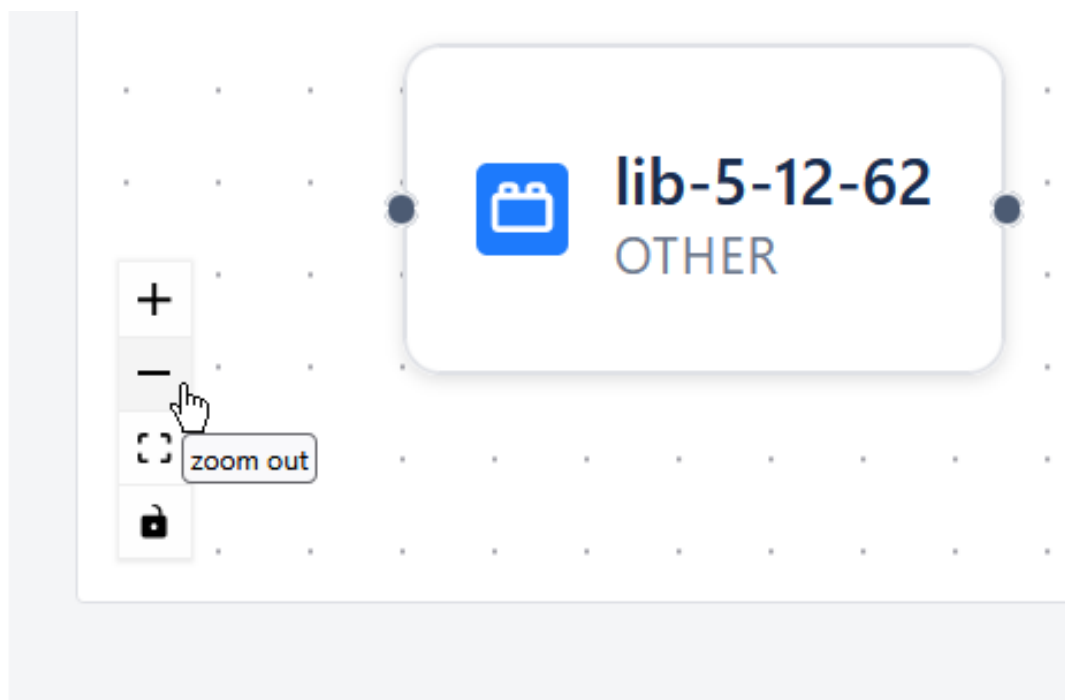


Рисунок 3.9 – Кнопки керування масштабом полотна

– щоб детальніше вивчити складну архітектуру, використовуйте елементи управління на самих вузлах дерева. Натискайте на невеликі кнопки зі стрілками розміщених на вузлах, для миттєвого згортання або розгортання окремих дочірніх гілок графа. Це допоможе звільнити місце на екрані монітора;

– для проведення комплексного аудиту всієї доступної інфраструктури перемкніть режим відображення. Натисніть кнопку «Матриця» на верхній панелі керування. Система швидко сформує зведену таблицю перехресних зв'язків. Вигляд типу відображення «Матриця» зображено на рисунку 3.10;

Дерево	Матриця	Шукати елемент			
Загальний відсоток залежностей: 1.8%		Всього Capabilities: 23	Всього Модулів: 192		
Модулі \ Capabilities	cap-5-12-1	cap-5-12-10	cap-5-12-11	cap-5-12-12	cap-5-12-13
svc-5-12-19					
svc-5-12-2					
svc-5-12-20	✓				
svc-5-12-21					
svc-5-12-22		✓			
svc-5-12-23	✓				
svc-5-12-24					
svc-5-12-25					
svc-5-12-26					✓
svc-5-12-27					
svc-5-12-28	✓				
svc-5-12-29					
Покриття модулями (%)	2.6%	3.1%	1.6%	1.6%	3.6%

Рисунок 3.10 - Перегляд даних у режимі матриці зв'язків

- під час пошуку потрібного сервісу в обох режимах скористайтесь вбудованим текстовим полем. Почніть вводити назву мікросервісу. Програма автоматично відфільтрує зайві рядки в матриці або підсвітить знайдений вузол на графі дерева. При цьому камера плавно сфокусується на потрібному елементі;
- якщо виникла потреба проаналізувати інший системний вузол, необов'язково повертатися до головного каталогу Compass. Просто відкрийте випадаючий список у верхній правій частині робочого вікна додатка. Знайдіть там назву нового сервісу та натисніть на неї, вигляд випадаючого списку подано на рисунку 3.11;

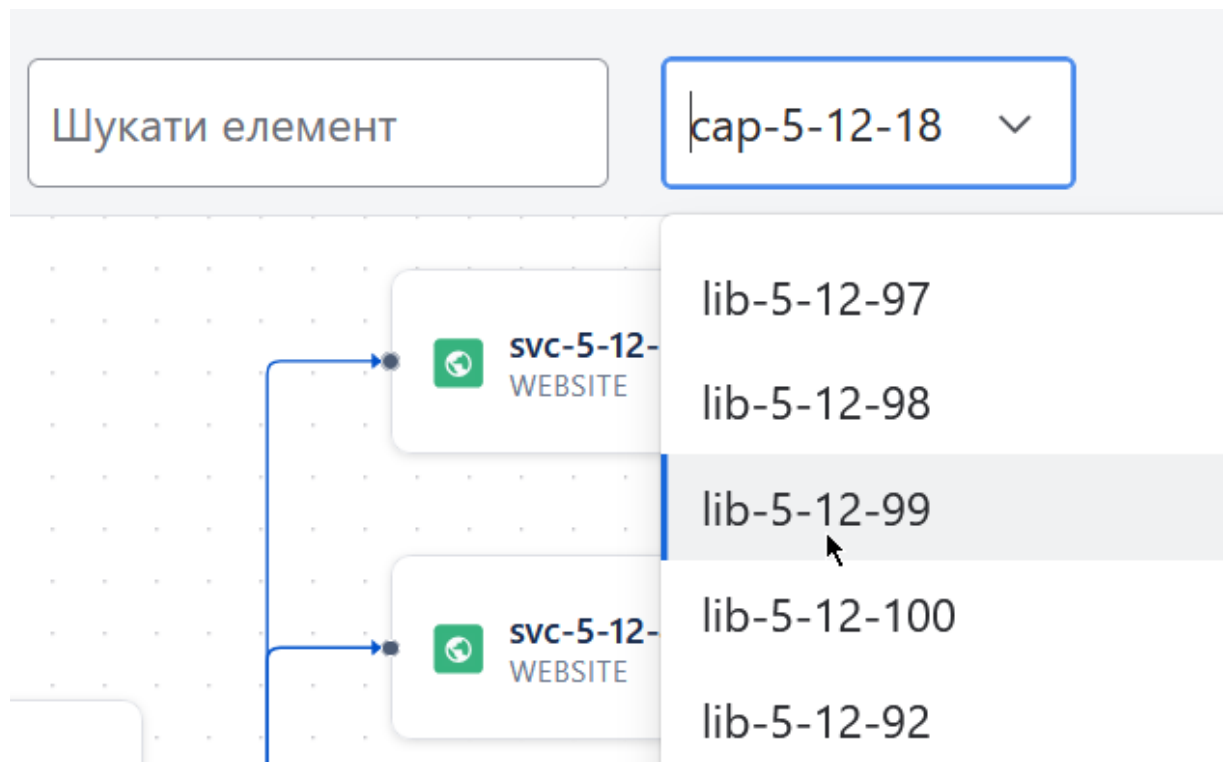


Рисунок 3.11 - Використання випадаючого списку для зміни компонента

Після вибору іншого елемента система самостійно надішле мережевий запит до сервера. Дерево миттєво перебудується з урахуванням нових архітектурних даних. Завдяки таким інструментам швидкої навігації процес вивчення інфраструктури проекту стає набагато простішим та комфортнішим для кожного технічного фахівця компанії. Виконання повсякденних завдань значно пришвидшується.

3.3 Інструкція з популяризації та підтримки додатку

Після успішного завершення етапу розробки та тестування виникає необхідність забезпечення стабільного життєвого циклу програмного продукту. Для залучення активної аудиторії користувачів та гарантування безперебійної роботи розроблено детальний план просування.

Правильна стратегія розповсюдження дозволяє швидко знайти зацікавлених інженерів. Процес підтримки забезпечує тривалу актуальність коду.

Популяризація додатку Compass-map здійснюється через кілька

паралельних каналів, щоб максимально охопити цільову аудиторію розробників та архітекторів. Першим і головним шляхом є офіційна публікація продукту в глобальному каталозі Atlassian Marketplace. Для цього готується детальний опис функціональних можливостей, створюються якісні знімки екрана та записується коротке відео з демонстрацією роботи дерева залежностей.

Проходження процедури сертифікації від вендора підвищує рівень довіри до розширення серед нових користувачів. Продукт стає доступним усьому світу. Окрім цього, важливим аспектом є створення та ведення публічного вебсайту документації на базі Confluence або GitBook, що містить покрокові керівництва користувача, відповіді на типові запитання (FAQ) та описи сценаріїв інтеграції. Для залучення органічного трафіку застосовуються методи оптимізації сторінки додатку (ASO — App Store Optimization) з використанням ключових слів, таких як «Jira dependency tree», «microservices architecture mapping» та «architectural audit».

Першим і головним шляхом є офіційна публікація продукту в глобальному каталозі Atlassian Marketplace. Для цього готується детальний опис функціональних можливостей, створюються якісні знімки екрана та записується коротке відео з демонстрацією роботи дерева залежностей. Проходження процедури сертифікації від вендора підвищує рівень довіри до розширення серед нових користувачів. Продукт стає доступним усьому світу.

Другий шлях розповсюдження орієнтований на внутрішні корпоративні мережі. У великих ІТ-компаніях впровадження нових інструментів відбувається через проведення внутрішніх технічних презентацій або воркшопів. Відповідальні особи готують розсилку для профільних відділів з детальним поясненням переваг використання матриці зв'язків під час проведення архітектурних аудитів. Швидке розгортання через Forge CLI дозволяє командам миттєво протестувати функціонал на власному ізольованому просторі.

Враховуючи суворі вимоги до інформаційної безпеки у корпоративному сегменті, архітектура додатку розроблена на платформі Atlassian Forge. Це гарантує виконання коду у захищеному пісочнику (sandbox) на серверах Atlassian та

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						56
Зм.	Арк.	№ докум.	Підпис	Дата		

дотримання стандартів збереження конфіденційності (зокрема GDPR та SOC 2), оскільки дані не передаються на сторонні сервери. Такий підхід суттєво полегшує проходження внутрішніх аудитів безпеки (SecOps) у великих організаціях.

Третій напрямок передбачає активну взаємодію з відкритими технічними спільнотами. Публікація статей на профільних ресурсах (наприклад, Хабр або Medium) з описом вирішення проблеми хаосу в мікросервісах привертає увагу незалежних фахівців. Розміщення частини вихідного коду на платформі GitHub заохочує інших програмістів долучатися до обговорення архітектурних рішень. Відкритий діалог формує лояльну базу прихильників.

Для забезпечення тривалої життєздатності системи розроблено суворий регламент технічної підтримки. Оновлення залежностей та виправлення знайдених помилок проводиться на регулярній основі. План підтримки включає такі обов'язкові заходи:

- Налаштування автоматизованої системи збору звітів про критичні збої в роботі клієнтського інтерфейсу (наприклад, через сервіс Sentry);
- Створення відкритого репозиторію для збору відгуків (Issue Tracker), де користувачі можуть самостійно фіксувати знайдені баги або пропонувати нові функції;
- Регулярний моніторинг змін у документації Atlassian API для своєчасної адаптації механізмів отримання даних;
- Формування графіка випуску міnorних оновлень (один раз на місяць) для стабілізації роботи та мажорних релізів (раз на пів року) для додавання об'ємного функціоналу.

Для обробки запитів користувачів впроваджується модель SLA (Service Level Agreement), відповідно до якої всі звернення класифікуються за рівнем критичності. Для критичних інцидентів (блокування роботи інтерфейсу) час реакції становить до 24 годин із випуском термінового виправлення (hotfix).

Для значних несправностей (порушення відображення окремих зв'язків) виправлення розробляється протягом 5 робочих днів, а побажання щодо нового

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		57

функціоналу реєструються в беклозі для оцінки та пріоритезації у наступних планових релізах.

Подальше покращення інтерфейсу базується виключно на аналізі реального користувацького досвіду. Якщо метрики показують, що інженери частіше використовують табличний вигляд замість графа, пріоритет розробки автоматично зміщується в бік розширення можливостей сортування матриці. Важливою складовою еволюції продукту є використання вбудованої анонімної аналітики використання (Forge Analytics) для відстеження взаємодії користувачів з інтерфейсом без порушення приватності.

Заплановано поступове впровадження додаткових аналітичних інструментів, таких як розрахунок критичності конкретного вузла при його відключенні. Робота над проєктом ніколи не зупиняється.

У довгостроковій перспективі запланована розробка інтелектуального помічника на базі машинного навчання, який зможе автоматично аналізувати топологію графів зв'язків, виявляти потенційні вузькі місця (bottlenecks) та рекомендувати оптимальні шляхи рефакторингу мікросервісної архітектури. Завдяки налагодженому процесу збору зворотного зв'язку Compass-map буде здатний швидко еволюціонувати відповідно до реальних потреб сучасних інженерних команд.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		58

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини даного кваліфікаційного проєкту є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки клієнтської частини інтеграційного додатка для платформи Atlassian Compass (Далі Compass-map), прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Об'єктом розробки є додаток «Compass-map (клієнт)»

Розрахунок вартості розробки виконується в декілька етапів:

- описати технологічний процес розробки із зазначенням трудомісткості кожної операції;
- визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;
- обчислити витрати на електроенергію;
- нарахувати суму амортизаційних відрахувань;
- визначити суму накладних витрат;
- скласти кошторис та визначити собівартість робіт;
- розрахувати ціну робіт;
- визначити економічну ефективність та термін окупності.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення розробки додатку

В цьому підрозділі розглянемо основні етапи технологічного процесу для розробки додатку

Для визначення загальної тривалості проведення робіт доцільно дані витрат часу по окремих операціях технологічного процесу зведемо у таблицю 4.1.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		59

Таблиця 4.1 - Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Планування та аналіз	Кер. проєкту Рm	8
		Інженер (1)	4
		Інженер (2)	4
2	Розробка технічного завдання	Кер. проєкту (Рm)	6
		Інженер (1)	4
		Інженер (2)	4
3	Дизайн інтерфейсу	Інженер (1)	16
4	Розробка функціоналу	Інженер (1)	40
		Інженер (2)	40
5	Тестування та відладка	Інженер (1)	8
		Інженер (2)	8
6	Документування	Кер. проєкту (Рm)	4
		Інженер (1)	2
		Інженер (2)	2
7	Розгортання та підтримка	Інженер (1)	4
		Інженер (2)	6
8	Управління проєктом	Кер. проєкту (Рm)	16
Разом			176

Загальний час по спеціалістах:

- керівник проєкту (РМ): 34 години;
- інженер 1 (Клієнт): 78 годин;
- інженер 2 (Сервер): 64 години.

Сумарний час виконання операцій технологічного процесу становить 176 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

У даному підрозділі проводиться аналіз і розрахунок витрат, пов'язаних з оплатою праці та відрахуваннями на соціальні заходи, що необхідні для розробки Compass-мар.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{\text{осн.}} = T_c \cdot K_r \quad (4.1)$$

де: T_c – тарифна ставка, для керівника проекту (Рм) – 350 грн./год., інженера (1) – 250 грн./год., інженера (2) – 300 грн./год, K_r – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

Керівника проекту (Рм) $З_{\text{осн.}} = 34 \cdot 350 = 11\,900$ грн.

Інженера (1) $З_{\text{осн.}} = 78 \cdot 250 = 19\,500$ грн.

Інженера (2) $З_{\text{осн.}} = 64 \cdot 300 = 19\,200$ грн.

Сумарна основна заробітна плата становить

$$З_{\text{осн.}} = 11\,900 + 19\,500 + 19\,200 = 50\,600 \text{ грн.}$$

Додаткова заробітна плата становить 10 – 15 % від суми основної заробітної плати.

$$З_{\text{дод.}} = З_{\text{осн.}} \cdot K_{\text{допл.}} \quad (4.2)$$

де: $K_{\text{допл.}}$ – коефіцієнт додаткових виплат працівникам.

Отже додаткова заробітна плата по категоріях працівників становить:

– Керівника проекту $З_{\text{дод2}} = 11\,900 \cdot 0,1 = 1\,190$ грн.

– Інженера (1) $З_{\text{дод3}} = 19\,500 \cdot 0,1 = 1\,950$ грн.

– Інженера (2) $З_{\text{дод4}} = 19\,200 \cdot 0,1 = 1\,920$ грн.

Загальна додаткова заробітна плата становить:

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		61

$$З_{\text{дод}} = 1\,190 + 1\,950 + 1\,920 = 5\,060 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($B_{o.n.}$) визначаються за формулою:

$$B_{o.n.} = З_{\text{осн.}} + З_{\text{дод.}} \quad (4.3)$$

$$B_{o.n.} = 50\,600 + 5\,060 = 55\,660 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$B_{\text{есв}} = B_{o.n.} * 0,22 \quad (4.4)$$

$$B_{\text{есв}} = 55\,660 * 0,22 = 12\,245 \text{ грн}$$

Проведені розрахунки витрат на оплату праці наведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці, грн. $6 = 3+4+5$
		Тариф на ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
		1	2	3	4	5	6
1	Кер. проєкту (Рм)	350	34	11 900	1 190		
2	Інженера (1)	250	78	19 500	1 950		
3	Інженера (2)	300	64	19 200	1 920		
Разом				50 600	5 060	12 245	67 905

Отже, загальні витрати на оплату праці становлять 67 905 грн.

4.3 Розрахунок витрат на електроенергію

Розрахуємо вартість електроенергії. Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_{\text{в}} = W * T * S \quad (4.5)$$

де: W – необхідна потужність, кВт; T – кількість годин роботи обладнання;

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		62

S – вартість кіловат-години електроенергії (приймаємо 15, 94 грн).

В нашій системі є 1 ПК. Витрати на електроенергію для цього комп'ютера обчислимо окремо, взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт описаних в таблиці 4.1, і споживані потужності обладнанням: комп'ютер (ноутбук) – 0,100 кВт/год.

$$З_{ек} = 0,100 * 176 * 15,94 = 281 \text{ грн.}$$

Витрати на електроенергію становлять 281 грн.

4.4 Розрахунок суми амортизаційних відрахувань додатку

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення

Комп'ютери та оргтехніка належать до четвертої групи основних фондів.

Амортизація на них нараховується лише в випадку, якщо мінімально допустимі строки їх корисного використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{БВ * Н_A}{100\%} * T, \quad (4.6)$$

де: A – амортизаційні відрахування за звітний період, грн.;

БВ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

Н_A – норма амортизації, 0,04 %.

Оскільки для написання програми та її тестування використовується один ПК, вартістю 50000,00 грн., то сума амортизаційних відрахувань становитиме:

$$A = \frac{30\,000,00 * 0,04}{150} * 176 = 1\,408 \text{ грн.}$$

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						63
Зм.	Арк.	№ докум.	Підпис	Дата		

4.5 Обчислення накладних витрат

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці. В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$H_B = B_{o.п.} * 0,2..0,6 \quad (4.7)$$

де: H_B – накладні витрати.

$$H_B = 55\,660 * 0,4 = 22\,264 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості додатку

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.4.

Таблиця 4.4 - Кошторис витрат додатку

	Зміст витрат	Сума, грн.	В % до загальної суми
1.	Витрати на оплату праці	67 905	73,89
2.	Витрати на електроенергію	281	0,31
3.	Амортизаційні відрахування	1 408	1,57
4.	Накладні витрати	22 264	24,23
5.	Собівартість	91 858	100

Собівартість (C_B) НДР розрахуємо за формулою:

$$C_B = B_{o.п.} + B_{c.з} + 3e + A + H_B \quad (4.8)$$

Отже, собівартість дорівнює $C_B = 91\,858$ грн.

4.7 Розрахунок ціни додатку

Розрахунок ціни науково-дослідної роботи включає в себе урахування різноманітних факторів, таких як рівень рентабельності, собівартість та податкова ставка.

Ціну робіт можна визначити за формулою:

$$Ц = C_{\text{в}} * (1 + P_{\text{рен}}) * (1 + \text{ПДВ}), \quad (4.9)$$

де: $C_{\text{в}}$ – собівартість; $P_{\text{рен}}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість.

$$Ц = 91\,858 * (1 + 0,3) * (1 + 0,2) = 143\,298 \text{ грн}$$

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності (Ток).

$$\text{ЧТВ} = -C_{\text{в}} + \sum_{i=1}^t \frac{\Gamma_{\text{п}}}{(1+i)^i}, \quad (4.10)$$

де: $C_{\text{в}}$ – собівартість розробки; $\Gamma_{\text{п}}$ – грошовий потік за t – ий рік; t – відповідний рік проекту; i – величина дисконтної ставки (10...15%).

$$\text{ЧТВ} = -91858 + \frac{51440}{(1+0,1)^1} + \frac{51440}{(1+0,1)^2} + \frac{51440}{(1+0,1)^3} = 36094 \text{ грн}$$

Якщо $\text{ЧТВ} \geq 0$, то проект може бути рекомендований до впровадження.

Термін окупності визначається за формулою:

$$T_{\text{ок}} = T_{\text{пв}} + \frac{H_{\text{в}}}{\Gamma_{\text{пр}}} \quad (4.11)$$

де: $T_{\text{пв}}$ – період до повного відшкодування витрат, років; $H_{\text{в}}$ – невідшкодовані витрати на початок року, грн.; $\Gamma_{\text{пр}}$ – грошовий потік на початок

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						65
Зм.	Арк.	№ докум.	Підпис	Дата		

року, грн.

$$T_{\text{ок}} = 2 + \frac{2581}{51440} = 2,05 \text{р.}$$

Всі дані внесемо в зведену таблицю 4.5.

Таблиця 4.5 – Техніко-економічні показники Compass-мар

№ п/п	Показник	Значення
1.	Собівартість, грн.	91 858
2.	Плановий прибуток або грошовий потік, грн.	51440
3.	Ціна, грн.	143298
4.	Чиста теперішня вартість, грн.	36094
5.	Термін окупності, рік	2,05

Прибутковість проєкту та термін окупності свідчать про його фінансову ефективність та здатність повернути капітальні вкладення протягом 2,05 року. Отже, на основі отриманих показників можна зробити висновок, що розробка додатку Compass-мар є доцільною з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Поняття праці та охорони праці

Праця є фундаментальною основою життєдіяльності людини та розвитку суспільства. Праця - це цілеспрямована, свідома діяльність людини, у процесі якої вона видозмінює і пристосовує предмети природи для задоволення своїх потреб, створюючи матеріальні та духовні блага. У процесі виробничої діяльності людина постійно взаємодіє з предметами праці, обладнанням та навколишнім виробничим середовищем. Ця взаємодія нерідко супроводжується впливом небезпечних та шкідливих виробничих факторів, що зумовлює гостру необхідність запровадження комплексу захисних заходів.

Охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності. Згідно із Законом України «Про охорону праці», головною метою цієї системи є створення безпечних і нешкідливих умов праці на кожному робочому місці.

Основні напрямки охорони праці на підприємстві реалізуються через такі заходи [11]:

- правові заходи - це створення та дотримання нормативно-правової бази (закони, правила, інструкції, стандарти), що регулює відносини у сфері безпеки праці та встановлює відповідальність сторін;
- організаційно-технічні заходи - це впровадження безпечних технологічних процесів, модернізація обладнання, автоматизація та механізація важких чи небезпечних робіт, проведення інструктажів і навчання працівників безпечним методам роботи;
- санітарно-гігієнічні заходи - це нормалізація параметрів мікроклімату, виробничого освітлення, зниження рівня шуму, вібрації, випромінювань, а також

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						67
Зм.	Арк.	№ докум.	Підпис	Дата		

забезпечення належної якості повітря робочої зони;

– лікувально-профілактичні заходи - це проведення попередніх та періодичних медичних оглядів працівників, забезпечення спецодягом, спецвзуттям та іншими засобами індивідуального захисту (ЗІЗ), надання лікувально-профілактичного харчування.

Створення належних умов праці не лише зберігає фізичне здоров'я та психологічний стан працівників, але й сприяє підвищенню продуктивності праці, зниженню плинності кадрів та рівня професійного травматизму, що безпосередньо впливає на загальну економічну ефективність підприємства.

5.2 Робоча загальнообмінна штучна вентиляція

Для забезпечення належних санітарно-гігієнічних умов на робочих місцях, зокрема для нормалізації складу та параметрів повітряного середовища, найважливішим інженерно-технічним заходом є вентиляція. Вентиляція - це організований повітрообмін, що полягає у видаленні з виробничого приміщення забрудненого або нагрітого повітря та подачі замість нього свіжого.

За способом переміщення повітряних мас вентиляцію поділяють на природну та штучну (механічну). Штучна вентиляція здійснюється за допомогою спеціального обладнання (вентиляторів, моторів, ежекторів), яке створює необхідний перепад тиску. На відміну від природної, штучна вентиляція не залежить від метеорологічних умов (температури зовнішнього повітря, швидкості вітру), дозволяє очищати повітря перед викидом в атмосферу або перед подачею в приміщення, нагрівати, охолоджувати чи зволожувати його, а також подавати повітря по повітроводах на значні відстані.

За зоною обслуговування штучна вентиляція поділяється на місцеву, загальнообмінну та комбіновану.

Загальнообмінна вентиляція - призначена для підтримки нормального повітряного середовища у всьому об'ємі робочої зони приміщення [12]. Вона застосовується у тих випадках, коли шкідливі речовини (пил, гази, пари),

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						68
Зм.	Арк.	№ докум.	Підпис	Дата		

надлишкове тепло або волога виділяються рівномірно по всьому об'єму приміщення, і немає можливості локалізувати їх в місцях утворення за допомогою місцевих витяжок. Принцип дії загальнообмінної вентиляції полягає в розбавленні шкідливих виділень шляхом подачі великих об'ємів чистого повітря до рівня, нижчого за гранично допустимі концентрації (ГДК).

Робоча загальнообмінна вентиляція - це система, яка функціонує постійно протягом усього робочого часу підприємства чи цеху. Вона забезпечує стабільний повітрообмін, необхідний для безпечної праці людей і правильного перебігу технологічних процесів (на відміну від аварійної вентиляції, яка вмикається епізодично, лише у разі раптового та масового викиду шкідливих речовин).

За принципом організації повітрообміну робоча загальнообмінна штучна вентиляція буває трьох типів:

- припливна вентиляція - примусово нагнітає свіже повітря у приміщення, створюючи незначний надлишковий тиск, через який забруднене повітря витісняється назовні через вікна або спеціальні витяжні отвори;

- витяжна вентиляція - механічно видаляє забруднене повітря з приміщення, створюючи розрідження (знижений тиск), завдяки чому чисте повітря надходить ззовні або з суміжних чистіших приміщень, що робить цей тип ефективним для приміщень, з яких неприпустиме перетікання шкідливих речовин чи неприємних запахів в інші зони будівлі;

- припливно-витяжна вентиляція - одночасно використовує і припливні, і витяжні агрегати, що дозволяє точно збалансувати об'єми повітря, яке подається та видаляється, будучи найбільш поширеною системою на виробництві;

Основним критерієм для розрахунку та проектування загальнообмінної вентиляції є кратність повітрообміну - величина, яка показує, скільки разів протягом однієї години все повітря в приміщенні повинно повністю замінитися на свіже. Розрахунок робочої вентиляції проводиться відповідно до Державних будівельних норм (ДБН) з урахуванням характеру виробництва, кількості працівників та обсягів виділення шкідливих речовин.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		69

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно створено клієнтську частину інтеграційного додатку для платформи Atlassian Compass під назвою «Compass-map». Проєкт повністю реалізовано згідно із затвердженими вимогами технічного завдання. Робота охоплює повний цикл проєктування візуальної оболонки: від розробки первинних макетів інтерфейсу до написання логіки рендерингу складних графічних елементів.

Під час виконання практичної частини було проведено глибокий аналіз предметної області та визначено ключові обмеження хмарної інфраструктури Atlassian. Розроблено модульну структуру графічного інтерфейсу. Спроєктовано логіку взаємодії користувача з елементами керування, забезпечивши зручну навігацію. Особливу увагу приділено алгоритмам візуалізації великих масивів інформації без втрати продуктивності браузера.

У межах програмної реалізації було використано сучасний стек веб-технологій, бібліотеку React версії 18 та інструмент збірки Vite. Для побудови інтерактивного дерева залежностей застосовано рішення з бібліотеки ReactFlow у поєднанні з математичним ядром Dagre для автоматичного розрахунку координат вузлів. Розроблено механізми безпечного отримання архітектурних даних від віддаленого сервера через вбудований інтерфейс Forge Bridge. Написаний код миттєво обробляє вхідні JSON-пакети.

Клієнтський інтерфейс забезпечує інженерам комфортний перегляд архітектури мікросервісів у двох різних форматах: через наочний граф та аналітичну табличну матрицю. Функції швидкого текстового пошуку та фільтрування дозволяють миттєво знаходити потрібні вузли серед сотень інших елементів системи. Завдяки правильному використанню компонентів Atlassian Design System додаток виглядає як органічна частина середовища Compass.

У процесі роботи було самостійно розроблено алгоритми рендерингу, кастомні вузли графа та логіку сортування табличних даних. Також було підготовлено необхідні структурні схеми та блок-схеми, що детально описують

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						70
Зм.	Арк.	№ докум.	Підпис	Дата		

принципи роботи візуальної частини. Написана програма функціонує без збоїв.

Отримані результати підтверджують повне виконання поставлених на початку роботи завдань.

Створена клієнтська оболонка є повністю готовим до використання інструментом для проведення глибоких архітектурних аудитів інфраструктури розробки.

Подальший розвиток проєкту може включати розширення функціоналу візуалізації. Планується додавання нових режимів перегляду та впровадження інструментів для експорту згенерованих схем у популярні графічні формати. Напрямок покращення інтерфейсу залежить від аналізу реального користувацького досвіду.

Головним результатом виконання цієї роботи стало набуття практичних навичок проєктування складних користувацьких інтерфейсів (SPA) для закритого корпоративного середовища.

Під час написання коду було закріплено розуміння процесів асинхронного обміну даними та організації внутрішнього стану React-компонентів. Набуто досвід створення інтерактивної графіки на полотні браузер.

Окрім написання коду, було суттєво вдосконалено навички складання правильної технічної документації та роботи з хмарним середовищем Atlassian. Це дало чітке розуміння всіх етапів розробки програмних рішень від затвердження ідеї до релізу готового продукту.

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		71

ПЕРЕЛІК ПОСИЛАНЬ

- 1) Richardson C. Microservices Patterns: With examples in Java, Second Edition. Shelter Island : Manning Publications, 2024. 550 p.
- 2) Suleymani T. Hands-On Microservices with JavaScript. Birmingham : Packt Publishing, 2024. 420 p.
- 3) Atlassian Compass Developer Documentation : вебсайт. URL: <https://developer.atlassian.com/compass/> (дата звернення: 10.06.2026).
- 4) Atlassian Forge Documentation : вебсайт. URL: <https://developer.atlassian.com/platform/forge/> (дата звернення: 10.06.2026).
- 5) React Official Documentation : вебсайт. URL: <https://react.dev/> (дата звернення: 10.06.2026).
- 6) Wieruch R. The Road to GraphQL. Berlin : Independently published, 2023. 350 p.
- 7) Vite – Next Generation Frontend Tooling : вебсайт. URL: <https://vitejs.dev/> (дата звернення: 10.06.2026).
- 8) Atlassian Design System : вебсайт. URL: <https://atlassian.design/> (дата звернення: 10.06.2026).
- 9) React Flow (xyflow) Documentation : вебсайт. URL: <https://reactflow.dev/> (дата звернення: 10.06.2026).
- 10) Dagre – Directed graph layout for JavaScript : вебсайт. URL: <https://github.com/dagrejs/dagre> (дата звернення: 10.06.2026).
- 11) Ткачук К.Н., Халімовський М.О., Зацарний В.В. Основи охорони праці. Київ : Основа, 2006. 448 с. 2. ДБН В.2.5-67:2013
- 12) Опалення, вентиляція та кондиціонування : вебсайт. URL: https://e-construction.gov.ua/laws_detail/3074971619479783152 (дата звернення: 13.06.2026).

					2026.КВР.122.423.08.00.00 ПЗ	Арк.
						72
Зм.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Додаток А. Лістинг файлу «App.jsx»

```
import React, { useEffect, useState } from 'react';
import { invoke } from '@forge/bridge';
import './static/css/style.css';
import Button from '@atlaskit/button/new';
import ButtonGroup from '@atlaskit/button/button-group';
import { Box, Inline, Stack } from '@atlaskit/primitives';
import DependencyTableComponent from './components/DependencyTable.jsx';
import ReactFlowTree from './components/ReactFlowTree.jsx';
import ComponentSelector from './components/ComponentSelector.jsx';
import SearchComponent from './components/SearchComponent.jsx';
function App() {
  const [oneComponent, setOneComponent] = useState(null);
  const [allComponents, setAllComponents] = useState(null);
  const [viewMode, setViewMode] = useState('tree');
  const [selectedComponentId, setSelectedComponentId] = useState('');
  const [searchTerm, setSearchTerm] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  useEffect(() => {
    invoke('getComponent')
      .then((data) => {
        setOneComponent(data);
        if (data && data.id) {
          setSelectedComponentId(data.id);
        }
      })
      .catch((err) =>
        console.error('Failed to fetch data getOneComponent', err));
  }, []);
  useEffect(() => {
    invoke('getAllComponents')
      .then(setAllComponents)
      .catch((err) => console.error('Failed to fetch data
```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						73
Зм.	Арк.	№ докум.	Підпис	Дата		

```

getAllComponents', err));
  }, []);
  const handleComponentChange = (componentId) => {
    setSelectedComponentId(componentId);
    setIsLoading(true);
    invoke('getComponent', { componentId })
      .then((data) => {
        setOneComponent(data);
        setIsLoading(false);
      })
      .catch((err) => {
        console.error('Failed to fetch component', err);
        setIsLoading(false);      });    });
  return (
    <Box padding="space.200">
      <Stack space="space.100">
        <Inline          space="space.200"          alignBlock="center"
spread="between">
          <Inline space="space.200" alignBlock="center">
            <ButtonGroup>
              <Button
                appearance={viewMode === 'tree' ? 'primary'
: 'default'}
                onClick={() => setViewMode('tree')}
              >
                Дерево
              </Button>
              <Button
                appearance={viewMode === 'table' ? 'primary'
: 'default'}
                onClick={() => setViewMode('table')}
              >
                Матриця
              </Button>

```

```

        </ButtonGroup>
        <SearchComponent                                value={searchTerm}
onChange={setSearchTerm} />
    </Inline>
    {viewModel === 'tree' && (
        <ComponentSelector
            allComponents={allComponents}
            selectedId={selectedComponentId}
            onSelect={handleComponentChange}
            isLoading={isLoading}
        />
    )}
</Inline>
<Box>
    {isLoading ? (
        <Box padding="space.500">
            <Inline                                alignBlock="center"
alignInline="center">
                Оновлення дерева...
            </Inline>
        </Box>
    ) : (
        viewModel === 'tree' ? (
            <ReactFlowTree                                data={oneComponent}
searchTerm={searchTerm} />
        ) : (
            <DependencyTableComponent    data={allComponents}
searchTerm={searchTerm} />
        )
    )}
</Box>
</Stack>
</Box>
);} export default App;

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						75
Зм.	Арк.	№ докум.	Підпис	Дата		

Додаток Б. Лістинг файлу «treeHelpers.js»

```
import dagre from 'dagre';
import { MarkerType } from 'reactflow';
import { NODE_WIDTH, NODE_HEIGHT, PRIMARY_COLOR } from '../constants/tree';

const dagreGraph = new dagre.graphlib.Graph();
dagreGraph.setDefaultEdgeLabel(() => ({}));

export const getLayoutedElements = (nodes, edges, direction = 'TB') => {
  const isHorizontal = direction === 'LR';
  dagreGraph.setGraph({ rankdir: direction });

  nodes.forEach((node) => {
    dagreGraph.setNode(node.id, { width: NODE_WIDTH, height: NODE_HEIGHT });
  });

  edges.forEach((edge) => {
    dagreGraph.setEdge(edge.source, edge.target);
  });

  dagre.layout(dagreGraph);

  nodes.forEach((node) => {
    const nodeWithPosition = dagreGraph.node(node.id);
    node.targetPosition = isHorizontal ? 'left' : 'top';
    node.sourcePosition = isHorizontal ? 'right' : 'bottom';

    node.position = {
      x: nodeWithPosition.x - NODE_WIDTH / 2,
      y: nodeWithPosition.y - NODE_HEIGHT / 2,
    };
  });
}
```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						76
Зм.	Арк.	№ докум.	Підпис	Дата		


```

        return node;
    });

    return { nodes, edges };
};

export const searchInComponent = (comp, lowerTerm) => {
    if (!comp) return false;
    if (comp.name && comp.name.toLowerCase().includes(lowerTerm)) return
true;
    if (comp.relationships) {
        return comp.relationships.some((rel) =>
searchInComponent(rel.startNode, lowerTerm));
    }
    return false;
};

const createNode = (comp, hasChildren, isCollapsed) => ({
    id: comp.id,
    type: 'custom',
    data: {
        label: comp.name,
        type: comp.typeId,
        iconUrl: comp.iconUrl,
        hasChildren,
        isCollapsed
    },
    position: { x: 0, y: 0 },
    hidden: false
});

const createEdge = (sourceId, targetId) => ({
    id: `e-${sourceId}-${targetId}`,
    source: sourceId,

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						77
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    target: targetId,
    type: 'smoothstep',
    markerEnd: { type: MarkerType.ArrowClosed, color: PRIMARY_COLOR },
    style: { stroke: PRIMARY_COLOR },
    hidden: false
  });

export const buildGraph = (comp, collapsedIds, nodesMap, edgesMap, searchTerm
= '') => {
  if (!comp || !comp.id) return;

  const hasChildren = comp.relationships && comp.relationships.length > 0;
  const isCollapsed = collapsedIds.has(comp.id);

  if (!nodesMap.has(comp.id)) {
    const isSearchMatch = searchTerm && comp.name &&
comp.name.toLowerCase().includes(searchTerm.toLowerCase());
    const node = createNode(comp, hasChildren, isCollapsed);
    if (isSearchMatch) {
      node.data.isSearchMatch = true;
    }
    nodesMap.set(comp.id, node);
  }

  if (hasChildren) {
    comp.relationships.forEach((rel) => {
      const startNode = rel.startNode;
      if (startNode) {
        buildGraph(startNode, collapsedIds, nodesMap, edgesMap,
searchTerm);

        const edgeId = `e-${comp.id}-${startNode.id}`;
        if (!edgesMap.has(edgeId)) {
          edgesMap.set(edgeId, createEdge(comp.id,
startNode.id));

```

					2026.KBP.122.423.08.00.00 ПЗ	Арх.
Зм.	Арх.	№ докум.	Підпис	Дата		78

```

        }
    }
    });
}
};

export const traverseVisibility = (nodeId, isVisible, nodesMap, edgesMap,
collapsedIds, visibleNodeIds, searchTerm = '') => {
    const node = nodesMap.get(nodeId);
    if (!node) return;

    node.hidden = !isVisible;
    if (isVisible) visibleNodeIds.add(nodeId);

    const isCollapsed = searchTerm ? false : collapsedIds.has(nodeId);
    const childEdges = Array.from(edgesMap.values()).filter(e => e.source
=== nodeId);

    childEdges.forEach(edge => {
        edge.hidden = !isVisible || isCollapsed;
        traverseVisibility(edge.target, isVisible && !isCollapsed, nodesMap,
edgesMap, collapsedIds, visibleNodeIds, searchTerm);
    });
};

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		79

Додаток В. Лістинг файлу «ReactFlowTree.jsx»

```
import React, { useEffect, useState, useMemo, useCallback } from 'react';
import ReactFlow, {
  useNodesState,
  useEdgesState,
  Controls,
  Background,
  Handle,
  Position,
} from 'reactflow';
import 'reactflow/dist/style.css';
import { Stack } from '@atlaskit/primitives';
import {
  getLayoutedElements,
  buildGraph,
  traverseVisibility
} from '../helpers/treeHelpers';
import { DEFAULT_DIRECTION, NODE_WIDTH, NODE_HEIGHT } from
'../constants/tree';

const sampleData = {
  id: 'loading',
  name: 'Завантаження...',
  type: 'SERVICE',
  iconUrl: '',
  relationships: [],
};

const CustomNode = ({ data }) => {
  return (
    <div className={`custom-node ${data.hasChildren ? 'can-expand' : ''}
    ${data.isSearchMatch ? 'search-match' : ''}`}>
      <Handle type="target" position={Position.Left}

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		80

```

className="handle-style" />
      {data.iconUrl ? (
        <img src={data.iconUrl} alt={data.type} className="custom-
node-icon" />
      ) : (
        <div className="custom-node-icon-placeholder" />
      )}
    <div className="custom-node-content">
      <div className="custom-node-label">{data.label}</div>
      <div className="custom-node-type">{data.type}</div>
    </div>
    {data.hasChildren && (
      <div className="expand-icon-wrapper">
        {data.isCollapsed ? '►' : '▼'}
      </div>
    )}
    <Handle type="source" position={Position.Right}
className="handle-style" />
  </div>
);
};

const nodeTypes = {
  custom: CustomNode,
};

const ReactFlowTree = ({ data, searchTerm }) => {
  const [nodes, setNodes, onNodesChange] = useNodesState([]);
  const [edges, setEdges, onEdgesChange] = useEdgesState([]);
  const [collapsedIds, setCollapsedIds] = useState(new Set());
  const [rfInstance, setRfInstance] = useState(null);

  const displayData = useMemo(() => {
    return data || sampleData;
  }, [data]);

```

```

    }, [data]));

    const onNodeClick = useCallback((event, node) => {
        if (!node.data.hasChildren) return;

        setCollapsedIds((prev) => {
            const next = new Set(prev);
            if (next.has(node.id)) {
                next.delete(node.id);
            } else {
                next.add(node.id);
            }
            return next;
        });
    }, []);

    useEffect(() => {
        if (!displayData) {
            setNodes([]);
            setEdges([]);
            return;
        }

        const nodesMap = new Map();
        const edgesMap = new Map();

        if (Array.isArray(displayData)) {
            displayData.forEach(item => {
                const comp = item.component || item;
                buildGraph(comp, collapsedIds, nodesMap, edgesMap,
searchTerm);
            });
        } else if (typeof displayData === 'object') {
            buildGraph(displayData, collapsedIds, nodesMap, edgesMap,

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		82

```

searchTerm);
    }

    const visibleNodeIds = new Set();
    const roots = [];

    const childIds = new Set(Array.from(edgesMap.values()).map(e =>
e.target));
    Array.from(nodesMap.keys()).forEach(id => {
        if (!childIds.has(id)) roots.push(id);
    });
    roots.forEach(rootId => traverseVisibility(rootId, true, nodesMap,
edgesMap, collapsedIds, visibleNodeIds, searchTerm));

    const visibleNodes = Array.from(nodesMap.values()).filter(n =>
!n.hidden);
    const visibleEdges = Array.from(edgesMap.values()).filter(e =>
!e.hidden);

    if (visibleNodes.length > 0) {
        const { nodes: layoutedNodes, edges: layoutedEdges } =
getLayoutedElements(
            visibleNodes,
            visibleEdges,
            DEFAULT_DIRECTION
        );
        setNodes(layoutedNodes);
        setEdges(layoutedEdges);
    } else {
        setNodes([]);
        setEdges([]);
    }
}, [displayData, collapsedIds, setNodes, setEdges, searchTerm]);

```

					2026.KBP.122.423.08.00.00 ПЗ	Арх.
Зм.	Арх.	№ докум.	Підпис	Дата		83

```

    useEffect(() => {
      if (!rfInstance || !searchTerm) return;
      const matchedNodes = nodes.filter(n => n.data &&
n.data.isSearchMatch);
      if (matchedNodes.length === 1) {
        const match = matchedNodes[0];
        const x = match.position.x + NODE_WIDTH / 2;
        const y = match.position.y + NODE_HEIGHT / 2;
        rfInstance.setCenter(x, y, { zoom: 1.2, duration: 800 });
      }
    }, [nodes, searchTerm, rfInstance]);
    return (
      <Stack space="space.100">
        <div className="tree-container">
          <ReactFlow
            nodes={nodes}
            edges={edges}
            onNodesChange={onNodesChange}
            onEdgesChange={onEdgesChange}
            onNodeClick={onNodeClick}
            onInit={setRfInstance}
            nodeTypes={nodeTypes}
            nodesDraggable={false}
            fitView
          >
            <Controls />
            <Background />
          </ReactFlow>
        </div>
      </Stack>
    );
  };

export default ReactFlowTree;

```


Додаток Г. Лістинг файлу «dependencyHelpers.jsx»

```
import React from 'react';

export const searchInComponent = (comp, lowerTerm) => {
  if (!comp) return false;
  if (comp.name && comp.name.toLowerCase().includes(lowerTerm)) return
true;
  if (comp.relationships) {
    return comp.relationships.some((rel) =>
searchInComponent(rel.startNode, lowerTerm));
  }
  return false;
};

export const processComponent = (comp, capabilitiesMap, otherComponentsMap,
dependencies) => {
  if (!comp || !comp.name) return;

  const isCapability = comp.typeId && comp.typeId.toUpperCase() ===
'CAPABILITY';
  if (isCapability) {
    capabilitiesMap.set(comp.name, comp.typeId);
  } else {
    otherComponentsMap.set(comp.name, comp.typeId);
  }

  if (comp.relationships && Array.isArray(comp.relationships)) {
    comp.relationships.forEach((rel) => {
      const startNode = rel.startNode;
      if (startNode && startNode.name) {
        const startIsCap = startNode.type &&
startNode.type.toUpperCase() === 'CAPABILITY';
```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						85
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        if (isCapability && !startIsCap) {
            dependencies.add(`${startNode.name}->${comp.name}`);
        } else if (!isCapability && startIsCap) {
            dependencies.add(`${comp.name}->${startNode.name}`);
        }

        processComponent(startNode, capabilitiesMap,
otherComponentsMap, dependencies);
    }
    });
}
};

export const processAllComponents = (filteredData) => {
    const capabilitiesMap = new Map();
    const otherComponentsMap = new Map();
    const dependencies = new Set();

    if (Array.isArray(filteredData)) {
        filteredData.forEach((item) => {
            const comp = item.component || item;
            processComponent(comp, capabilitiesMap, otherComponentsMap,
dependencies);
        });
    } else if (typeof filteredData === 'object' && filteredData !== null) {
        processComponent(filteredData, capabilitiesMap, otherComponentsMap,
dependencies);
    }

    return { capabilitiesMap, otherComponentsMap, dependencies };
};

export const getHeader = (sortedCapabilities) => {
    return {

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		86

```

    cells: [
      {
        key: 'header-corner',
        content: 'Модулі \\ Capabilities',
      },
      ...sortedCapabilities.map((cap) => ({
        key: `col-${cap}`,
        content: (
          <div title={cap}>
            {cap}
          </div>
        ),
      })),
      {
        key: 'header-stats',
        content: 'Використання в Cap (%)',
      }
    ],
  };
};

```

```

export const getRows = (sortedOthers, sortedCapabilities, dependencies,
moduleUsage, capabilityUsage, overallPercentage) => {
  const rows = sortedOthers.map((mod) => {
    const usageCount = moduleUsage[mod] || 0;
    const usagePercent = sortedCapabilities.length > 0
      ? ((usageCount / sortedCapabilities.length) * 100).toFixed(1)
      : 0;

    return {
      key: `row-${mod}`,
      cells: [
        {
          key: `cell-${mod}-header`,

```

```

        content: <strong>{mod}</strong>,
    },
    ...sortedCapabilities.map((cap) => {
        const hasDependency = dependencies.has(`${mod}-${
>${cap}`);

        return {
            key: `cell-${mod}-${cap}`,
            content: (
                <div className="text-center">
                    {hasDependency ? '✓' : ''}
                </div>
            ),
        };
    }),
    {
        key: `cell-${mod}-stats`,
        content: <div className="text-center">{usagePercent}%</div>,
    },
];

});

if (sortedCapabilities.length > 0 && sortedOthers.length > 0) {
    rows.push({
        key: 'row-summary-stats',
        cells: [
            {
                key: 'cell-summary-header',
                content: <strong>Покриття модулями (%)</strong>,
            },
            ...sortedCapabilities.map((cap) => {
                const usageCount = capabilityUsage[cap] || 0;
                const usagePercent = ((usageCount / sortedOthers.length)

```

```

* 100).toFixed(1);

        return {
            key: `cell-summary-${cap}`,
            content: <div className="text-center"><strong>{usagePercent}%</strong></div>,
        };
    })),
    {
        key: 'cell-summary-overall',
        content: <div className="text-center"><strong>{overallPercentage}%</strong></div>,
    }
    ],
    });
}

return rows;
};

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		89

Додаток Д. Лістинг файлу «DependencyTable.jsx»

```
import React, { useState, useMemo } from 'react';
import DynamicTable from '@atlaskit/dynamic-table';
import { Stack, Box, Inline, Text } from '@atlaskit/primitives';
import {
  searchInComponent,
  processAllComponents,
  getHeader,
  getRows
} from '../helpers/dependencyHelpers';
const sampleData = {
  id: 'loading',
  name: 'Завантаження...',
  type: 'SERVICE',
  iconUrl: '',
  relationships: [],
};
const DependencyTableComponent = ({ data, searchTerm }) => {
  const { capabilitiesMap, otherComponentsMap, dependencies } = useMemo(()
=> {
    const allComponents = data || [];
    const dataToProcess = allComponents.length ? allComponents : [{
component: sampleData }];
    return processAllComponents(dataToProcess);
  }, [data]);
  const sortedCapabilities = useMemo(() =>
Array.from(capabilitiesMap.keys()).sort(), [capabilitiesMap]);
  const sortedOthers = useMemo(() => {
    let others = Array.from(otherComponentsMap.keys());
    if (searchTerm) {
      const lowerTerm = searchTerm.toLowerCase();
      others = others.filter(mod
mod.toLowerCase().includes(lowerTerm));
    }
  }, [searchTerm, otherComponentsMap]);
```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						90
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    } return others.sort();
  }, [otherComponentsMap, searchTerm]);
  const { capabilityUsage, moduleUsage, totalConnections } = useMemo(() =>
  {
    const capUsage = {};
    const modUsage = {};
    let connections = 0;
    const visibleOthers = new Set(sortedOthers);
    dependencies.forEach(dep => {
      const [mod, cap] = dep.split('->');
      if (capabilitiesMap.has(cap) && visibleOthers.has(mod)) {
        capUsage[cap] = (capUsage[cap] || 0) + 1;
        modUsage[mod] = (modUsage[mod] || 0) + 1;
        connections++;
      } });
    return { capabilityUsage: capUsage, moduleUsage: modUsage,
totalConnections: connections };
  }, [dependencies, capabilitiesMap, sortedOthers]);
  const totalPossibleConnections = sortedCapabilities.length *
sortedOthers.length;
  const overallPercentage = totalPossibleConnections > 0
    ? ((totalConnections / totalPossibleConnections) * 100).toFixed(1)
    : 0;
  const head = useMemo(() => getHeader(sortedCapabilities),
[sortedCapabilities]);
  const rows = useMemo(() =>
    getRows(sortedOthers, sortedCapabilities, dependencies,
moduleUsage, capabilityUsage, overallPercentage),
    [sortedOthers, sortedCapabilities, dependencies, moduleUsage,
capabilityUsage, overallPercentage]
  );
  const hasData = sortedCapabilities.length > 0 || sortedOthers.length >
0;
  return (

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						91
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    <Stack space="space.100">
      <Box
        padding="space.100"
        backgroundColor="color.background.neutral.subtle" borderRadius="4px">
        <Inline space="space.400">
          <Box>
            <Text weight="bold">Загальний відсоток залежностей:
          </Text>

          <Text
            appearance="success">{overallPercentage}%</Text>
          </Box>
          <Box>
            <Text weight="bold">Всього Capability: </Text>
            <Text>{sortedCapabilities.length}</Text>
          </Box>
          <Box>
            <Text weight="bold">Всього Модулів: </Text>
            <Text>{sortedOthers.length}</Text>
          </Box>
        </Inline>
      </Box>
    <div className="table-wrapper">
      <DynamicTable
        head={hasData ? head : undefined}
        rows={hasData ? rows : []}
        emptyView={
          <div className="empty-view">
            Немає даних для таблиці
          </div>
        }
      />
    </div>
  </Stack>

  );};
export default DependencyTableComponent;

```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						92
Зм.	Арк.	№ докум.	Підпис	Дата		

Додаток Е. Лістинг файлу «component-page.js»

```
import Resolver from '@forge/resolver';
import { cleanComponent, cleanAllComponents } from
'../helpers/resolverHelpers.js';
import { GET_COMPONENT, SEARCH_ALL_COMPONENTS } from
'../constants/queries.js';
import { requestGraph, fetchAllComponents } from '../helpers/graphApi.js';

const resolver = new Resolver();

resolver.define('getComponent', async ({ payload, context }) => {
    const componentId = payload?.componentId ||
context.extension.componentId;

    const data = await requestGraph(GET_COMPONENT, { componentId });
    const componentResult = data?.compass?.component;

    return cleanComponent(componentResult);
});

resolver.define('getAllComponents', async ({ context }) => {
    const { cloudId } = context;

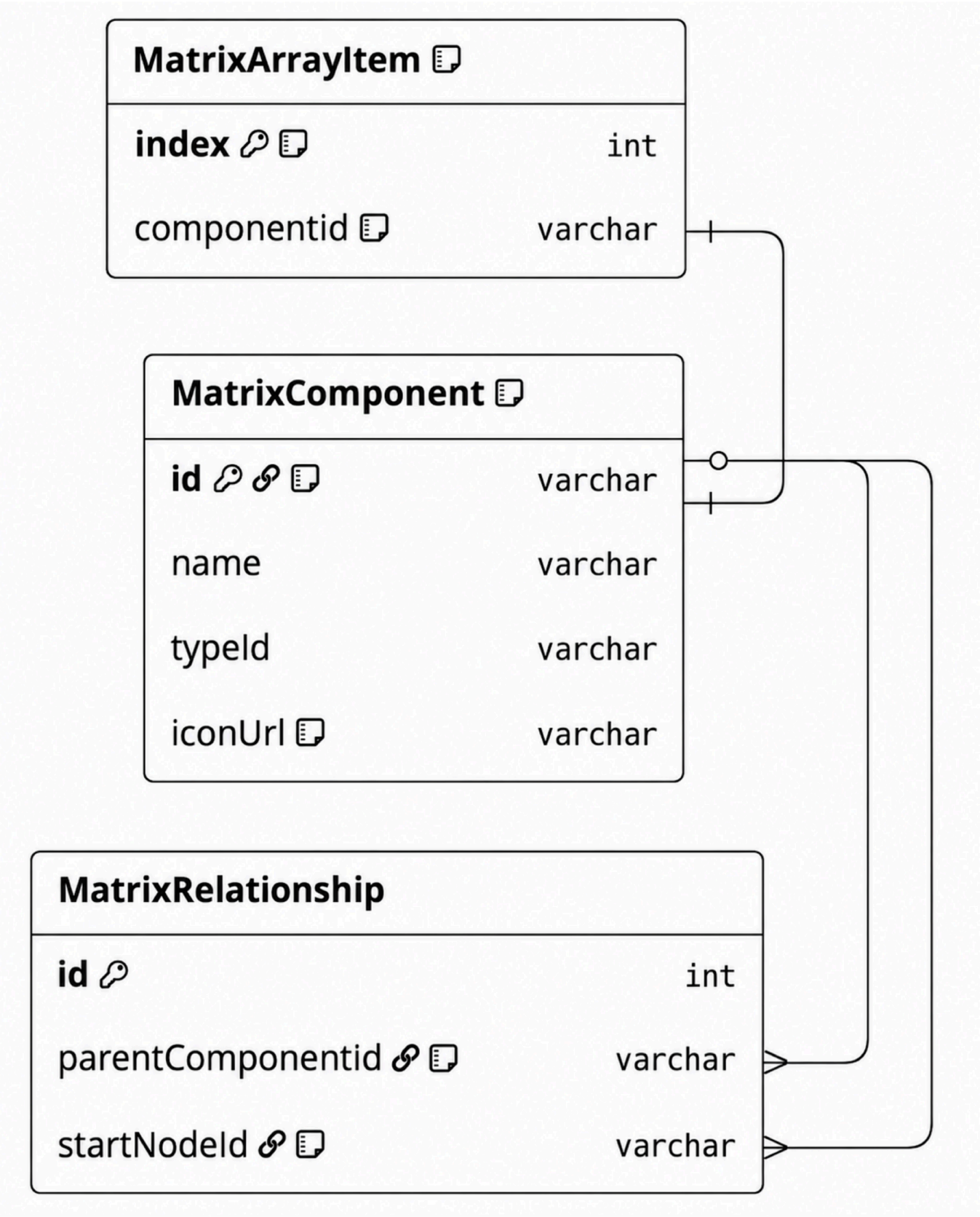
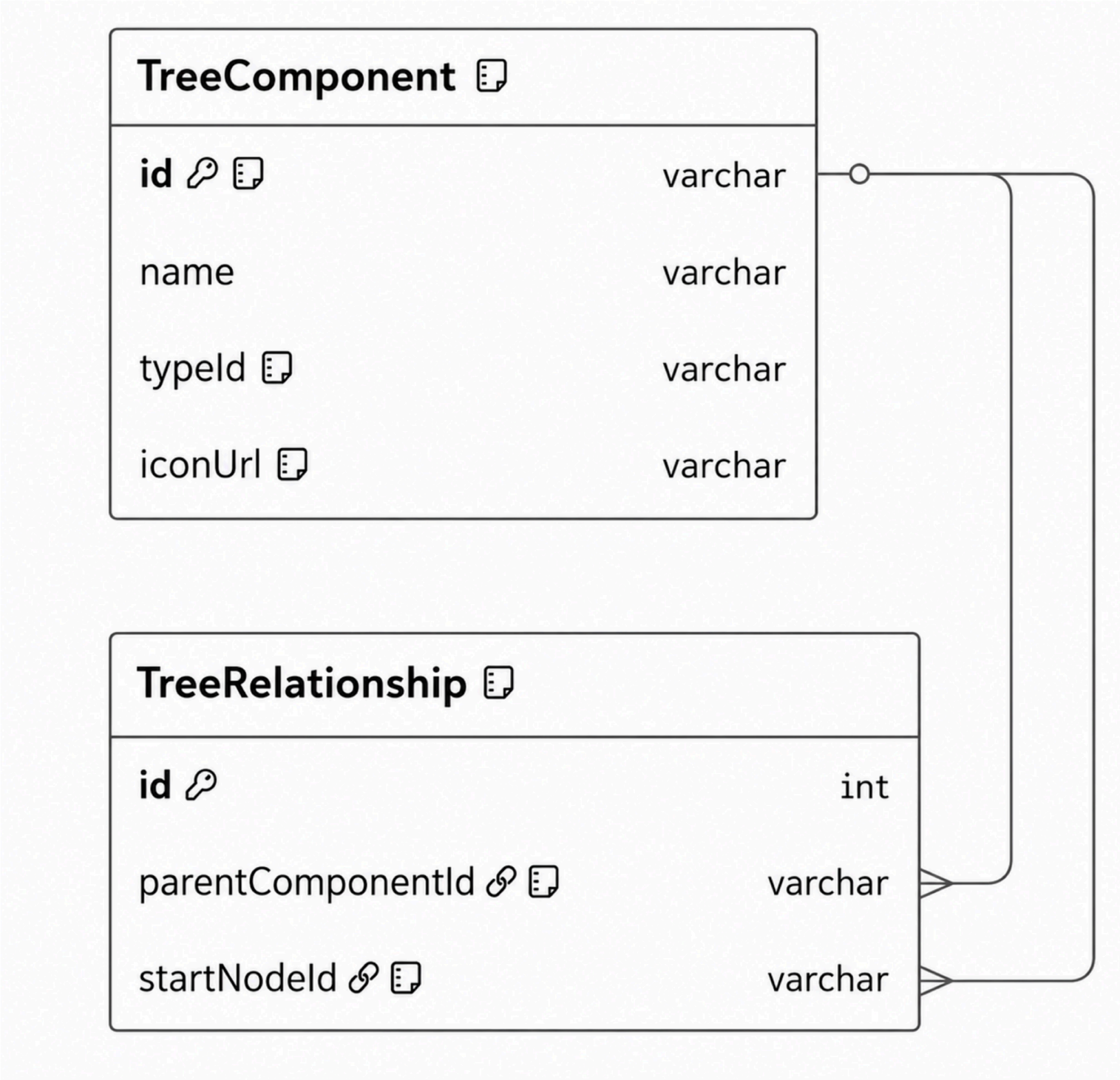
    const allComponents = await fetchAllComponents(SEARCH_ALL_COMPONENTS,
cloudId);

    return cleanAllComponents(allComponents);
});

export const run = resolver.getDefinitions();
```

					2026.KBP.122.423.08.00.00 ПЗ	Арк.
						93
Зм.	Арк.	№ докум.	Підпис	Дата		

ER-діаграма формату даних



Таблиця техніко-економічних показників

№	Показник	Одиниця вимірювання	Значення
1	Мова програмування	–	JavaScript
2	Фреймворк	–	React 18.2.0
3	Інструмент збірки	–	Vite 5.0.0
4	Механізм взаємодії з сервером	–	Atlassian Forge Bridge (API invoke)
5	Основна бібліотека UI	–	Atlassian Atlaskit
6	Бібліотека візуалізації графа	–	React Flow 11.11.4
7	Архітектурний тип застосунку	–	SPA (Single Page Application)
8	Содіовартість	грн	91 858
9	Плановий грошовий потік	грн	51440
10	Ціна	грн	143 298
11	Чиста теперішня вартість	грн	36094
12	Термін окупності	рік	2,05

Архитектура додатку

