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 електроних систем
(назва відділення)

Циклова комісія комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра
(освітньо-професійного ступеня)

на тему: "Розробка серверної частини інтеграційного додатка для платформи Atlassian Compass"

Виконав: студентка IV курсу, групи КН-423

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

Когут АРТЕМ
(ім'я та прізвище)

Керівник Лісовий ВОЛОДИМИР
(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 СТУДЕНТУ

Когуту Артему Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка серверної 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 студенткою роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: JavaScript; API: Compass GraphQL API; бібліотека Forge Resolver, Forge API, 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 Технічне завдання

1.2.1 Найменування та область застосування

1.2.2 Призначення розробки

1.2.3 Вимоги до програмного забезпечення

1.2.4 Вимоги до програмної документації

1.2.5 Техніко-економічні показники

1.2.6 Стадії та етапи розробки

1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту

2.1 Розробка загальної структури і варіантів використання програми

2.2 Розробка системи класів

2.3 Розробка методів

2.4 Проєктування і опис інтерфейсу користувача

2.5 Опис файлової структури програми

2.6 Опис структури бази даних програми

2.7 Тестування програми і результати її виконання

3 Спеціальний розділ

3.1 Інструкція з інсталяції програмного забезпечення

3.2 Інструкція з використання тестових наборів

3.3 Інструкція з експлуатації програмного комплексу

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань серверної частини інтеграційного додатка платформи Atlassian Compass

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості серверної частини інтеграційного додатка платформи Atlassian Compass

4.7 Розрахунок ціни серверної частини інтеграційного додатка платформи Atlassian Compass

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Створення на підприємстві служби охорони праці.

5.2. Загальні заходи та засоби нормалізації параметрів мікроклімату

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – додатку.

5. Перелік графічного матеріалу:

1. Структурна схема серверної частини додатка
2. Лістинг файлу component-page.js

3. Алгоритм очищення отримуваних 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	09.06.2026	
12	Захист кваліфікаційної роботи	24.06.2026	

7. Дата видачі завдання: 05 березня 2026 р.

Студент _____ Артем КОГУТ
(підпис)

Керівник роботи _____ Володимир ЛІСОВИЙ
(підпис)

АНОТАЦІЯ

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

Метою роботи є розробка серверної частини інтеграційного додатка для платформи Atlassian Compass, що забезпечує отримання та обробку інформації про залежності між програмними компонентами, формування дерева батьківських залежностей і таблиці зв'язків компонентів типу Capability.

Пояснювальна записка складається з п'яти розділів. У роботі виконано аналіз існуючих рішень, сформовано технічне завдання, розроблено структуру програмного забезпечення, систему класів та методи обробки даних, проведено тестування й підготовлено інструкції з інсталяції та експлуатації. Серверна частина реалізована з використанням Node.js та платформи Atlassian Forge. Дані отримуються через Compass GraphQL API без використання власної бази даних.

Обсяг пояснювальної записки – _____ сторінок.

Графічна частина містить структурну схему серверної частини додатка, лістинг модуля component-page.js, блок-схему алгоритму очищення даних та таблицю техніко-економічних показників.

Qualification work topic: Development of the server side of an integration application for the Atlassian Compass platform.

The purpose of the work is to develop the server side of an integration application for the Atlassian Compass platform that provides retrieval and processing of information about dependencies between software components, generation of a parent dependency tree and a Capability relationship table.

The explanatory note consists of five sections. The work includes analysis of existing solutions, technical specification, software structure, class system and data processing methods, testing, and preparation of installation and operation manuals. The server side is implemented using Node.js and the Atlassian Forge platform. Data is retrieved through the Compass GraphQL API without using a dedicated database.

The explanatory note contains _____ pages.

The graphical part includes the structural diagram of the server side of the application, the listing of the component-page.js module, the flowchart of the data cleaning algorithm, and the table of technical and economic indicators.

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій та постійне збільшення масштабів програмних систем призводять до зростання складності їхньої архітектури. Сучасні програмні продукти часто складаються з десятків або навіть сотень окремих компонентів, сервісів та модулів, які взаємодіють між собою та утворюють складну систему залежностей. У таких умовах особливого значення набуває ефективне управління інформацією про програмні компоненти, їх зв’язки та взаємодії, оскільки від цього залежить якість підтримки програмного забезпечення, швидкість внесення змін та стабільність роботи всієї системи.

Під час розробки та супроводу великих програмних продуктів виникає необхідність централізованого зберігання інформації про компоненти системи, їх власників, залежності та технічні характеристики. Відсутність такої інформації або складність її отримання можуть призвести до помилок під час розробки, збільшення часу на аналіз архітектури програмного забезпечення та виникнення проблем при внесенні змін до існуючих компонентів.

Одним із сучасних рішень для управління програмними компонентами є платформа Atlassian Compass. Вона забезпечує централізоване представлення компонентів програмної системи, дозволяє відстежувати їхній стан, взаємозв’язки, відповідальних осіб та інші характеристики. Використання Atlassian Compass спрощує управління складними програмними продуктами та дозволяє підтримувати актуальну інформацію про архітектуру системи. Разом із тим у процесі експлуатації платформи виникають задачі, для вирішення яких необхідно створювати додаткові додатки, що розширюють стандартний функціонал системи.

Актуальність теми кваліфікаційної роботи зумовлена необхідністю автоматизації процесів отримання та обробки інформації про програмні компоненти та їх залежності в середовищі Atlassian Compass. У великих програмних проєктах аналіз взаємозв’язків між компонентами є важливою складовою процесу підтримки архітектури та оцінювання впливу змін на інші частини системи. Використання інтеграційного додатка дозволяє спростити

отримання таких даних та підготувати їх до подальшого аналізу.

У межах кваліфікаційної роботи розробляється серверна частина інтеграційного додатка для платформи Atlassian Compass. Основним призначенням програмного продукту є отримання інформації про програмні компоненти та залежності між ними за допомогою Compass GraphQL API, обробка отриманих даних та підготовка результатів для подальшого використання. Додаток функціонує на платформі Atlassian Forge та використовує механізм Forge Resolver для реалізації серверної логіки.

Метою кваліфікаційної роботи є розробка серверної частини інтеграційного додатка для платформи Atlassian Compass, який забезпечує автоматизоване отримання, обробку та підготовку інформації про програмні компоненти та залежності між ними.

Об’єктом дослідження є процес отримання та обробки інформації про програмні компоненти в середовищі Atlassian Compass.

Предметом дослідження є методи, засоби та технології розробки серверної частини інтеграційних додатків на платформі Atlassian Forge з використанням GraphQL API.

Практичне значення отриманих результатів полягає у створенні серверної частини інтеграційного додатка, що дозволяє автоматизувати отримання інформації про програмні компоненти та їх залежності, забезпечує централізовану обробку даних і може бути використана для підвищення ефективності аналізу архітектури програмних систем.

Структура кваліфікаційної роботи складається із загального розділу, розділу розробки технічного та робочого проєкту, спеціального розділу, економічного розділу, розділу з охорони праці, техніки безпеки та екологічних вимог, висновків, переліку посилань та додатків[1]. У роботі послідовно розглянуто аналіз предметної області, проєктування серверної частини інтеграційного додатка, реалізацію основних методів обробки даних, тестування програмного забезпечення, а також питання його розгортання та експлуатації.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Сучасні програмні системи характеризуються високою складністю архітектури та значною кількістю взаємопов'язаних компонентів. Особливо це характерно для корпоративних програмних продуктів, побудованих за принципами сервісно-орієнтованої або мікросервісної архітектури. У таких системах кожен сервіс може залежати від декількох інших сервісів, бібліотек або програмних модулів, які, своєю чергою, також мають власні залежності. У результаті формується складна структура взаємозв'язків між компонентами програмного забезпечення, контроль та аналіз якої є важливою складовою процесу розробки та супроводу програмних систем.

Під час експлуатації великих програмних продуктів виникає потреба швидко визначати, які компоненти впливають на роботу конкретного сервісу, які залежності існують між різними частинами системи та які наслідки можуть виникнути після внесення змін до окремого компонента. Відсутність ефективних засобів аналізу залежностей може призводити до збільшення часу на дослідження архітектури програмного забезпечення, ускладнення процесу підтримки системи та підвищення ризику виникнення помилок.

Для розв'язання подібних задач використовуються спеціалізовані програмні платформи, призначені для централізованого управління програмними компонентами та їх взаємозв'язками. Одним із найбільш поширених рішень такого типу є платформа Atlassian Compass[2].

Atlassian Compass являє собою хмарну платформу[3] для управління компонентами програмного забезпечення. Система дозволяє формувати централізований каталог сервісів, бібліотек, застосунків та інших елементів програмної архітектури. Для кожного компонента можуть зберігатися відомості про його призначення, відповідальних осіб, документацію, репозиторії вихідного коду, а також залежності від інших компонентів системи.

Однією з основних можливостей Atlassian Compass є відображення зв'язків між компонентами. Завдяки цьому розробники отримують можливість аналізувати структуру програмного забезпечення та визначати взаємозалежності між окремими сервісами. Такий підхід дозволяє спростити підтримку складних програмних систем та покращити контроль архітектури програмного забезпечення.

Незважаючи на наявність механізму відображення залежностей, стандартний функціонал Atlassian Compass має певні обмеження. Під час перегляду інформації про компонент система відображає лише безпосередньо пов'язані елементи. Користувач може побачити тільки компоненти першого рівня залежності без можливості автоматичного перегляду повного ланцюга зв'язків.

У випадку невеликих програмних систем таке представлення є достатнім. Проте в корпоративних проєктах кількість рівнів залежностей може бути значно більшою. Наприклад, компонент може залежати від іншого сервісу, який, у свою чергу, використовує додаткові сервіси та бібліотеки. Для отримання повної інформації користувачеві необхідно послідовно переходити між компонентами та вручну аналізувати кожен рівень структури.

Такий підхід створює низку проблем. По-перше, значно збільшується час аналізу архітектури системи. По-друге, ускладнюється пошук критичних залежностей між компонентами. По-третє, користувач не отримує цілісного уявлення про місце компонента в загальній структурі програмного забезпечення. Особливо помітними ці недоліки стають у великих системах, що містять десятки або сотні взаємопов'язаних сервісів.

Ще одним недоліком стандартного функціоналу Atlassian Compass є відсутність механізму автоматичного формування повного дерева батьківських залежностей. Користувач може переглядати лише окремі зв'язки між компонентами, проте не має можливості одразу отримати повну ієрархію залежностей для вибраного елемента системи.

Крім того, стандартний інтерфейс платформи не містить спеціалізованої таблиці залежностей, яка б дозволяла переглядати структуровану інформацію про

всі зв'язки компонента в одному місці. За наявності великої кількості взаємозалежних сервісів це ускладнює процес аналізу та пошуку необхідної інформації.

У великих програмних системах аналіз залежностей є одним із найважливіших завдань підтримки архітектури програмного забезпечення. Будь-яка зміна окремого компонента потенційно може вплинути на роботу інших сервісів, модулів або підсистем. Тому розробники повинні мати можливість швидко визначати структуру залежностей та оцінювати можливі наслідки внесення змін до системи. Чим більшим стає програмний проєкт, тим складнішим є процес аналізу взаємозв'язків між його складовими.

Одним із найбільш поширених способів представлення залежностей є використання графових структур. У такому випадку компоненти програмної системи подаються у вигляді вузлів графа, а зв'язки між ними відображаються у вигляді ребер. Подібний підхід забезпечує наочне представлення архітектури системи та дозволяє швидко визначати взаємозв'язки між окремими елементами програмного забезпечення.

У платформі Atlassian Compass також використовується графічне представлення залежностей компонентів. Користувач має можливість переглядати зв'язки між окремими елементами системи за допомогою вбудованого графа залежностей. Проте стандартна реалізація має певні функціональні обмеження. Під час перегляду графа відображаються лише безпосередні зв'язки вибраного компонента. Система показує один рівень батьківських та один рівень дочірніх залежностей, що не дозволяє отримати повну картину взаємозв'язків у межах складної програмної архітектури.

За наявності багаторівневих залежностей користувач змушений послідовно відкривати кожен наступний компонент та окремо аналізувати його зв'язки. Такий підхід ускладнює дослідження архітектури програмного забезпечення та збільшує час пошуку необхідної інформації. Особливо помітною ця проблема стає в організаціях, де кількість компонентів може обчислюватися десятками або сотнями

одиниць.

Для вирішення подібних проблем часто використовується дерево залежностей. На відміну від звичайного графічного представлення, дерево дозволяє відображати повну ієрархію взаємозв'язків між компонентами в межах однієї структури. Користувач отримує можливість переглядати всі рівні залежностей одночасно та швидко визначати місце компонента в загальній архітектурі системи.

Важливою перевагою деревоподібного представлення є можливість рекурсивного аналізу зв'язків між компонентами. У цьому випадку система автоматично виконує пошук батьківських елементів до визначеного рівня вкладеності та формує єдину структуру даних. Такий підхід значно скорочує кількість ручних операцій, необхідних для аналізу архітектури програмного забезпечення.

Крім графічного відображення залежностей, важливим інструментом аналізу є табличне представлення інформації. Таблиця дозволяє отримати структурований перелік компонентів та швидко визначити наявність або відсутність зв'язків між ними. Особливо корисним такий підхід є під час аналізу компонентів типу Capability, які використовуються для опису бізнес-можливостей програмної системи.

У розроблюваному програмному продукті реалізовано механізм формування таблиці залежностей, яка відображає всі компоненти типу Capability та дозволяє визначати наявність зв'язків між ними та іншими компонентами системи. Додатково виконується розрахунок відсоткового співвідношення зв'язаних компонентів, що дозволяє оцінювати ступінь наповнення архітектури залежностями та спрощує подальший аналіз структури програмного забезпечення.

Під час проєктування системи також було враховано можливість виникнення циклічних залежностей між компонентами. Для запобігання нескінченному обходу структури зв'язків реалізовано механізм контролю вже оброблених компонентів. Крім того, пошук залежностей обмежується десятьма

рівнями вкладеності, що є достатнім для більшості практичних сценаріїв використання та дозволяє уникнути надмірного навантаження на систему.

Для усунення зазначених недоліків у сучасних програмних системах застосовуються різноманітні методи візуалізації залежностей. Найбільш поширеними серед них є графові структури, дерева залежностей та табличні представлення взаємозв'язків між компонентами. Використання таких підходів дозволяє значно спростити аналіз архітектури та забезпечити швидке отримання необхідної інформації.

Особливо ефективним підходом є побудова повного дерева залежностей, яке формується шляхом рекурсивного обходу всіх пов'язаних компонентів. На відміну від звичайного відображення окремих зв'язків, дерево дозволяє користувачеві бачити всю структуру залежностей одночасно. Це спрощує аналіз програмної системи та дозволяє швидше оцінювати вплив змін на інші компоненти.

На основі проведеного аналізу було встановлено, що стандартні можливості Atlassian Compass не забезпечують достатнього рівня деталізації під час роботи зі складними багаторівневими залежностями. Для вирішення цієї проблеми доцільним є створення інтеграційного додатка, який розширюватиме функціональні можливості платформи.

У межах кваліфікаційної роботи розробляється серверна частина інтеграційного додатка для Atlassian Compass, основним завданням якого є автоматизоване отримання та обробка інформації про залежності компонентів. На відміну від стандартного механізму платформи, розроблюване рішення забезпечує рекурсивний пошук усіх батьківських залежностей компонента незалежно від рівня вкладеності.

Серверна частина додатка отримує дані через Compass GraphQL API [4], виконує їх обробку та формує структуру повного дерева залежностей. Додатково реалізується механізм формування таблиці залежностей, який відсутній у стандартному функціоналі Atlassian Compass.

Порівняння стандартного функціоналу платформи та розроблюваного

програмного продукту наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння можливостей Atlassian Compass та розроблюваного додатка

Функціональна можливість	Atlassian Compass	Розроблюваний додаток
Відображення безпосередніх залежностей	Так	Так
Відображення багаторівневих залежностей	Ні	Так
Побудова повного дерева батьківських залежностей	Ні	Так
Рекурсивний аналіз залежностей	Ні	Так
Табличне представлення залежностей	Ні	Так
Отримання даних через GraphQL API	Так	Так
Автоматична обробка структури залежностей	Обмежено	Так

Таким чином, проведений аналіз показав, що існуючий механізм відображення залежностей у платформі Atlassian Compass має ряд функціональних обмежень під час роботи зі складними програмними системами. Розробка інтеграційного додатка дозволяє усунути виявлені недоліки шляхом реалізації рекурсивного аналізу залежностей, побудови повного дерева батьківських компонентів та формування структурованої таблиці залежностей. Це забезпечує більш зручний та ефективний аналіз архітектури програмного забезпечення.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Найменування програмного продукту – серверна частина інтеграційного додатка для платформи Atlassian Compass.

Програмний продукт призначений для отримання, обробки та підготовки інформації про залежності між програмними компонентами, що зберігаються в середовищі Atlassian Compass. Робота додатка здійснюється в межах хмарної платформи Atlassian Forge [5], яка забезпечує виконання серверної логіки та взаємодію з програмними інтерфейсами Atlassian.

Областю застосування програмного продукту є інформаційні системи, що використовують платформу Atlassian Compass для управління програмними компонентами та контролю архітектури програмного забезпечення. Додаток може використовуватися в організаціях, які розробляють складні програмні продукти з великою кількістю взаємопов'язаних сервісів, модулів та бібліотек.

Основним призначенням системи є автоматизація процесу отримання інформації про залежності між компонентами програмного забезпечення та формування повної структури батьківських залежностей для вибраного компонента. Отримані дані можуть використовуватися для аналізу архітектури програмної системи, оцінки впливу змін на інші компоненти та виявлення критичних залежностей між сервісами.

Програмний продукт функціонує як інтеграційний додаток усередині Atlassian Compass та використовує Compass GraphQL API для отримання інформації про компоненти. Власна база даних у системі не використовується, а всі необхідні дані отримуються безпосередньо з платформи Atlassian Compass у режимі реального часу.

Основними користувачами програмного продукту є:

– архітектори програмного забезпечення, які виконують аналіз структури програмних систем;

– розробники програмного забезпечення, що працюють із компонентами та сервісами проєкту;

– технічні керівники команд, які контролюють взаємозв'язки між компонентами програмної системи;

– DevOps-інженери та адміністратори, відповідальні за підтримку архітектури програмних продуктів.

Використання розробленого програмного продукту дозволяє спростити аналіз залежностей між компонентами, зменшити час пошуку інформації про структуру системи та підвищити ефективність роботи з архітектурою програмного забезпечення.

1.2.2 Призначення розробки

Експлуатаційним призначенням розроблюваного програмного продукту є забезпечення зручного аналізу залежностей між програмними компонентами, що зберігаються в середовищі Atlassian Compass. Програмний продукт призначений для використання розробниками програмного забезпечення, архітекторами програмних систем, технічними керівниками команд та іншими спеціалістами, які виконують аналіз структури програмних проєктів.

У процесі розробки та супроводу програмного забезпечення важливим завданням є визначення взаємозв'язків між компонентами системи. Наявність великої кількості сервісів та модулів ускладнює процес аналізу архітектури програмного забезпечення та оцінювання наслідків внесення змін до окремих компонентів. Стандартні можливості Atlassian Compass дозволяють переглядати лише безпосередні залежності компонентів, що є недостатнім для повноцінного аналізу складних багаторівневих структур.

Розроблюваний програмний продукт призначений для усунення зазначеного недоліку шляхом автоматизованого отримання та обробки інформації про всі батьківські залежності вибраного компонента. Використання додатка дозволяє користувачеві отримувати повну структуру взаємозв'язків без

необхідності виконання багаторазових переходів між окремими компонентами системи.

Функціональним призначенням програмного продукту є реалізація механізмів отримання, аналізу та обробки інформації про залежності програмних компонентів за допомогою програмного інтерфейсу Atlassian Compass GraphQL API.

Для досягнення поставленої мети програмний продукт повинен забезпечувати виконання таких функцій:

– отримання інформації про вибраний компонент із середовища Atlassian Compass;

– виконання запитів до Compass GraphQL API;

– отримання інформації про залежності між компонентами;

– рекурсивний пошук усіх батьківських компонентів незалежно від рівня вкладеності;

– обробку та структурування отриманих даних;

– виявлення та усунення дублювання компонентів у структурі залежностей;

– формування повного дерева батьківських залежностей;

– формування структурованої таблиці залежностей;

– передачу результатів обробки клієнтській частині додатка.

Робота програмного продукту базується на використанні платформи Atlassian Forge, яка забезпечує виконання серверної логіки без необхідності використання окремого сервера або додаткової серверної інфраструктури. Усі операції з отримання та обробки даних виконуються безпосередньо в середовищі Forge.

Власна база даних у програмному продукті не використовується. Необхідна інформація отримується в режимі реального часу з Atlassian Compass за допомогою GraphQL API. Такий підхід дозволяє працювати лише з актуальними даними та уникнути необхідності дублювання інформації в окремому сховищі.

Таким чином, експлуатаційним призначенням програмного продукту є підвищення ефективності аналізу архітектури програмних систем, а функціональним призначенням – автоматизоване отримання, обробка та представлення інформації про повну структуру батьківських залежностей програмних компонентів у середовищі Atlassian Compass.

1.2.3 Вимоги до програмного забезпечення

Для роботи програмного забезпечення використовуються дані, що надходять із платформи Atlassian Compass через GraphQL API. Отримана інформація використовується для аналізу залежностей між програмними компонентами та подальшого формування структури батьківських залежностей.

Основні вхідні дані наведено в таблиці 1.2.

Таблиця 1.2 – Перелік та опис вхідних даних

Назва вхідного повідомлення	Форма подання	Періодичність надходження	Джерело даних
Дані про компонент	GraphQL-відповідь	За запитом користувача	Atlassian Compass
Дані про залежності компонента	GraphQL-відповідь	За запитом користувача	Atlassian Compass
Дані про батьківські компоненти	GraphQL-відповідь	За запитом користувача	Atlassian Compass
Метадані компонентів	GraphQL-відповідь	За запитом користувача	Atlassian Compass

Результатом роботи програмного забезпечення є структурована інформація про залежності компонентів, підготовлена для подальшого відображення користувачеві.

Основні види вихідної інформації наведено в таблиці 1.3.

Таблиця 1.3 – Перелік та опис вихідних даних

Назва вихідного повідомлення	Форма подання	Періодичність формування	Користувач
Дерево залежностей	JSON	За запитом користувача	Клієнтська частина
Таблиця залежностей	JSON	За запитом користувача	Клієнтська частина
Список батьківських компонентів	JSON	За запитом користувача	Клієнтська частина
Повідомлення про помилки	JSON	У разі виникнення помилки	Клієнтська частина

Програмне забезпечення повинно забезпечувати отримання інформації про вибраний компонент із середовища Atlassian Compass та виконувати аналіз його залежностей. Для цього серверна частина взаємодіє з Compass GraphQL API та отримує необхідні дані про компоненти програмної системи.

Після отримання початкових даних виконується аналіз зв'язків між компонентами. Серверна частина здійснює рекурсивний пошук усіх батьківських залежностей вибраного компонента та формує повну структуру взаємозв'язків незалежно від кількості рівнів вкладеності. У процесі обробки інформації виконується перевірка вже опрацьованих компонентів для запобігання дублюванню даних у структурі залежностей.

Отримані результати використовуються для формування дерева залежностей та структурованої таблиці залежностей. Підготовлені дані передаються клієнтській частині додатка для подальшого відображення користувачеві.

Програмне забезпечення працює виключно в режимі читання та не виконує створення, зміну або видалення компонентів у середовищі Atlassian Compass. Усі операції обмежуються отриманням, обробкою та підготовкою інформації про

залежності компонентів.

До основних обмежень програмного забезпечення належать:

– функціонування лише в середовищі Atlassian Compass;

– залежність від доступності Compass GraphQL API;

– можливість роботи лише з компонентами, доступ до яких дозволено поточному користувачеві;

– відсутність можливості редагування даних компонентів;

– використання виключно офіційного програмного інтерфейсу платформи.

Для забезпечення надійної роботи програмного забезпечення повинна виконуватися перевірка коректності отриманих даних та обробка можливих помилок під час взаємодії з Atlassian Compass. У разі виникнення помилок система повинна формувати зрозумілі повідомлення про причини їх виникнення та не допускати аварійного завершення роботи. Під час рекурсивного аналізу залежностей необхідно здійснювати контроль уже оброблених компонентів, що дозволяє уникнути циклічних переходів та повторного опрацювання однакових елементів.

Експлуатація програмного забезпечення передбачається в середовищі Atlassian Compass із використанням платформи Atlassian Forge. Для коректної роботи необхідна наявність облікового запису Atlassian з відповідними правами доступу до компонентів системи, а також доступ до мережі Інтернет. Програмний продукт не потребує встановлення окремої бази даних або додаткового серверного програмного забезпечення, оскільки всі операції виконуються засобами платформи Atlassian Forge.

1.2.4 Вимоги до програмної документації

Розробка програмного забезпечення повинна супроводжуватися створенням комплекту документації, необхідної для впровадження, використання

та подальшого супроводу програмного продукту. Документація повинна забезпечувати можливість ознайомлення з принципами роботи системи, особливостями її структури та порядком експлуатації.

До складу документації повинні входити пояснювальна записка до кваліфікаційної роботи, опис структури програмного забезпечення, опис алгоритмів роботи основних програмних модулів, інструкція з інсталяції програмного забезпечення та інструкція з експлуатації програмного комплексу.

Пояснювальна записка повинна містити відомості про призначення програмного продукту, архітектуру системи, використані технології, алгоритми роботи програмних модулів та результати тестування.

Технічна документація повинна містити опис структури серверної частини додатка, порядок взаємодії з Atlassian Compass API, призначення окремих програмних модулів та особливості обробки інформації про залежності компонентів.

Кожний програмний модуль повинен супроводжуватися коментарями, які містять відомості про його призначення, вхідні та вихідні параметри, а також особливості використання. Коментарі повинні бути стислими, зрозумілими та відображати логіку роботи відповідних програмних елементів.

Інструкція з інсталяції повинна містити опис програмного середовища, необхідного для розгортання додатка, порядок встановлення залежностей та послідовність дій для розміщення програмного продукту в середовищі Atlassian Forge.

Інструкція з експлуатації повинна містити відомості про порядок використання програмного забезпечення, особливості взаємодії з платформою Atlassian Compass та можливі обмеження під час роботи системи.

Уся документація повинна бути оформлена відповідно до вимог чинних стандартів оформлення програмної та технічної документації, мати логічну структуру та забезпечувати зручність використання програмного продукту кінцевими користувачами та розробниками.

1.2.5 Техніко-економічні показники

Розробка серверної частини інтеграційного додатка для платформи Atlassian Compass виконується з метою підвищення ефективності аналізу залежностей між програмними компонентами та автоматизації процесу отримання інформації про структуру програмної системи.

У процесі експлуатації програмного забезпечення користувачі отримують можливість швидкого доступу до інформації про всі батьківські залежності компонента без необхідності виконання багаторазових переходів між окремими елементами системи. Це дозволяє скоротити час аналізу архітектури програмного забезпечення та спростити пошук взаємозв'язків між компонентами.

Розроблюване програмне забезпечення функціонує в середовищі Atlassian Forge та використовує наявну інфраструктуру платформи Atlassian. Завдяки цьому відсутня необхідність придбання додаткового серверного обладнання, встановлення окремих серверів або використання спеціалізованих систем керування базами даних. Такий підхід дозволяє зменшити витрати на впровадження та подальший супровід програмного продукту.

Для виконання робіт з розробки програмного забезпечення використовуються трудові та машинні ресурси. Основні ресурси, необхідні для створення програмного продукту, наведено в таблиці 1.4.

Таблиця 1.4 – Ресурси, необхідні для розробки програмного забезпечення

Найменування ресурсу	Значення
Кількість розробників	1
Трудомісткість розробки	3 людино-місяці
Машинний час	250 годин

Очікуваний технічний ефект від використання програмного продукту полягає у підвищенні зручності роботи з архітектурою програмного забезпечення, автоматизації аналізу залежностей між компонентами та зменшенні часу,

необхідного для пошуку інформації про структуру програмної системи.

Економічна доцільність розробки визначається скороченням трудових витрат на виконання операцій з аналізу залежностей компонентів, підвищенням продуктивності праці розробників та спрощенням процесу підтримки програмного забезпечення.

Таким чином, розробка серверної частини інтеграційного додатка для платформи Atlassian Compass є технічно та економічно доцільною, оскільки дозволяє підвищити ефективність роботи з архітектурою програмних систем без необхідності значних додаткових витрат на інфраструктуру та програмні засоби.

1.2.6 Стадії та етапи розробки

Розробка серверної частини інтеграційного додатка для платформи Atlassian Compass здійснювалася поетапно відповідно до загальноприйнятого процесу створення програмного забезпечення. Послідовне виконання окремих етапів дозволило забезпечити якісне проєктування системи, реалізацію необхідного функціоналу та перевірку працездатності програмного продукту.

На початковому етапі було виконано аналіз предметної області та дослідження можливостей платформи Atlassian Compass. У процесі аналізу було вивчено існуючі механізми роботи із залежностями компонентів, визначено особливості використання Compass GraphQL API та виявлено недоліки стандартного функціоналу платформи щодо відображення багаторівневих залежностей.

Наступним етапом стало формування технічного завдання. Було визначено функціональні вимоги до програмного забезпечення, сформовано перелік основних задач, які повинна вирішувати система, а також встановлено вимоги до архітектури майбутнього програмного продукту.

Під час проєктування було розроблено загальну структуру серверної частини додатка, визначено склад програмних модулів та порядок їх взаємодії. Окрему увагу приділено проєктуванню механізму взаємодії з Compass GraphQL

API та алгоритмам рекурсивного пошуку батьківських залежностей компонентів.

Етап програмної реалізації передбачав створення серверної частини додатка засобами платформи Atlassian Forge. Було реалізовано механізми виконання GraphQL-запитів [6], отримання інформації про компоненти, обробки даних та формування структури залежностей для подальшого використання клієнтською частиною системи.

Після завершення програмної реалізації виконувалося тестування програмного забезпечення. Під час тестування перевірялася коректність отримання інформації з Atlassian Compass, правильність формування дерева залежностей, робота алгоритмів рекурсивного обходу та обробка можливих помилкових ситуацій.

Завершальним етапом стало оформлення програмної та експлуатаційної документації. Було підготовлено пояснювальну записку, опис структури програмного забезпечення, опис алгоритмів роботи системи, а також інструкції щодо встановлення та використання програмного продукту.

Основні етапи виконання робіт наведено в таблиці 1.5.

Таблиця 1.5 – Стадії та етапи розробки програмного забезпечення

Стадія розробки	Зміст робіт
Аналіз предметної області	Дослідження Atlassian Compass та існуючих підходів до аналізу залежностей
Формування технічного завдання	Визначення вимог до програмного продукту
Проєктування	Розробка структури системи та алгоритмів роботи
Програмна реалізація	Розробка серверної частини додатка
Тестування	Перевірка працездатності та виправлення помилок
Документування	Підготовка технічної та експлуатаційної документації

У результаті виконання зазначених етапів було створено серверну частину інтеграційного додатка для платформи Atlassian Compass, яка забезпечує отримання, обробку та підготовку інформації про залежності між програмними компонентами.

1.2.7 Порядок контролю та прийому

Контроль якості розробленого програмного забезпечення повинен здійснюватися на всіх етапах життєвого циклу програмного продукту, починаючи від аналізу вимог і закінчуючи завершальним тестуванням готової системи. Основною метою контролю є підтвердження відповідності реалізованого функціоналу вимогам технічного завдання та забезпечення стабільної роботи програмного продукту в середовищі Atlassian Compass.

Перевірка правильності роботи програмного забезпечення повинна виконуватися шляхом тестування окремих функціональних модулів та комплексного тестування системи в цілому. Особлива увага приділяється коректності взаємодії серверної частини додатка з Compass GraphQL API, правильності обробки отриманих даних та формуванню структури залежностей компонентів.

У процесі контролю повинні перевірятися такі характеристики програмного забезпечення:

- коректність виконання GraphQL-запитів до Atlassian Compass;
- правильність отримання інформації про компоненти системи;
- правильність формування структури батьківських залежностей;
- коректність роботи алгоритмів рекурсивного обходу залежностей;
- правильність обробки компонентів із багаторівневою структурою зв'язків;
- відсутність дублювання даних у сформованих результатах;
- коректність передачі інформації між серверною та клієнтською

частинами додатка;

– стабільність роботи програмного забезпечення в середовищі Atlassian Forge.

Приймання програмного продукту здійснюється після завершення всіх етапів розробки та успішного проходження тестування. Підставою для прийняття програмного забезпечення є відповідність реалізованого функціоналу вимогам технічного завдання та відсутність критичних помилок, які можуть впливати на працездатність системи.

Критеріями успішного прийняття програмного забезпечення є:

– успішне розгортання додатка в середовищі Atlassian Forge;

– коректне отримання інформації з Atlassian Compass;

– правильне формування повної структури батьківських залежностей компонентів;

– коректна робота всіх реалізованих функцій;

– відповідність результатів роботи очікуваним значенням;

– відсутність критичних помилок під час експлуатації.

Результати перевірки повинні підтвердити готовність програмного продукту до використання в середовищі Atlassian Compass та його відповідність встановленим функціональним і технічним вимогам.

Таким чином, порядок контролю та прийому програмного забезпечення забезпечує перевірку працездатності розробленої системи, підтверджує її відповідність технічному завданню та гарантує можливість подальшого використання програмного продукту за призначенням.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка загальної структури і варіантів використання програми

На основі сформованого технічного завдання було виконано проєктування загальної структури програмного забезпечення та визначено основні варіанти його використання. Розроблюваний програмний продукт являє собою серверну частину інтеграційного додатка для платформи Atlassian Compass, основним призначенням якого є автоматизоване отримання та обробка інформації про залежності між програмними компонентами.

Особливістю розроблюваного рішення є його інтеграція з екосистемою Atlassian. Програмний продукт функціонує в середовищі Atlassian Forge та використовує Compass GraphQL API як основне джерело даних. Власна база даних у системі відсутня, а всі необхідні відомості про компоненти та їх залежності отримуються безпосередньо з Atlassian Compass у режимі реального часу.

Під час проєктування програмного забезпечення було прийнято рішення використовувати клієнт-серверну архітектуру. Такий підхід дозволяє розділити відповідальність між окремими частинами системи та забезпечити централізовану обробку даних на серверній стороні.

До складу програмного забезпечення входять такі основні компоненти:

- користувач Atlassian Compass;
- клієнтська частина додатка;
- серверна частина додатка;
- платформа Atlassian Forge;
- Compass GraphQL API;
- Atlassian Compass.

Структурну схему серверної частини інтеграційного додатка для платформи Atlassian Compass наведено у графічній частині кваліфікаційної роботи на 1 аркуші 2026.КВР.122.423.06.00.00 СС. На схемі відображено основні компоненти системи

та взаємозв'язки між клієнтською частиною, серверною частиною, платформою Atlassian Forge і Compass GraphQL API.

Користувач взаємодіє з клієнтською частиною додатка через інтерфейс Atlassian Compass. Клієнтська частина відповідає лише за відображення отриманої інформації та передачу запитів до серверної частини. Уся бізнес-логіка системи реалізована на серверній стороні.

Серверна частина є основним елементом програмного забезпечення. Саме вона виконує отримання інформації з Compass GraphQL API, аналіз залежностей між компонентами, формування дерева залежностей та підготовку даних для відображення у вигляді таблиці. Такий підхід дозволяє мінімізувати навантаження на клієнтську частину та забезпечити централізоване виконання всіх операцій з обробки даних.

Взаємодія між компонентами системи здійснюється за таким принципом. Після відкриття сторінки компонента в Atlassian Compass клієнтська частина автоматично передає його ідентифікатор серверній частині додатка. Серверна частина виконує необхідні GraphQL-запити до Compass API та отримує інформацію про компонент і його залежності. Після завершення обробки сформовані результати повертаються клієнтській частині для подальшого відображення користувачеві.

Під час аналізу предметної області було визначено основних суб'єктів, які беруть участь у роботі системи.

Основним суб'єктом є користувач Atlassian Compass. Саме він ініціює виконання всіх операцій та взаємодіє з результатами роботи програмного забезпечення.

Другим суб'єктом виступає платформа Atlassian Compass, яка забезпечує зберігання інформації про програмні компоненти та їх взаємозв'язки.

Третім суб'єктом є Compass GraphQL API, який використовується серверною частиною додатка для отримання необхідних даних.

На основі аналізу вимог до програмного забезпечення було визначено

основні варіанти використання системи. Варіантом використання називається окремий сценарій взаємодії користувача з програмним забезпеченням, який дозволяє досягти певного результату.

До основних варіантів використання розроблюваного програмного продукту належать:

- отримання інформації про компонент;
- отримання батьківських залежностей компонента;
- формування дерева залежностей;
- формування таблиці залежностей Capability;
- виконання пошуку в таблиці залежностей;
- обробка помилок отримання даних.

Загальна UML-діаграма[7] варіантів використання системи наведена на рисунку 2.1.

Рисунок 2.1 – UML-діаграма варіантів використання системи

Першим варіантом використання системи є отримання інформації про компонент. Даний сценарій виконується автоматично після відкриття сторінки компонента в Atlassian Compass. Клієнтська частина передає серверній частині ідентифікатор вибраного компонента. Після отримання запиту серверна частина виконує звернення до Compass GraphQL API та отримує інформацію про компонент, необхідну для подальшої обробки. Результатом виконання даного сценарію є отримання повного набору даних про вибраний компонент та його доступність для подальшого аналізу.

Основний сценарій виконання варіанту використання наведено нижче:

1) Користувач відкриває сторінку компонента.
2) Клієнтська частина формує запит до серверної частини.
3) Серверна частина отримує ідентифікатор компонента.
4) Виконується запит до Compass GraphQL API.
5) Отримується інформація про компонент.
6) Дані передаються для подальшої обробки.

Після отримання відповіді від Compass GraphQL API серверна частина перевіряє наявність необхідних даних та виконує їх попередню обробку. З отриманої структури виділяються ідентифікатор, назва, тип, піктограма компонента та відомості про його зв’язки. Зайві службові поля вилучаються, а результат перетворюється у формат, придатний для подальшого формування дерева і таблиці залежностей. Якщо компонент не знайдено або відповідь API не містить необхідних відомостей, система повертає порожній результат або повідомлення про помилку.

Другим варіантом використання є отримання батьківських залежностей компонента. Після отримання інформації про вибраний компонент серверна частина здійснює аналіз його зв'язків з іншими елементами системи. Для цього використовується механізм рекурсивного обходу залежностей. Пошук виконується до десятого рівня вкладеності або до моменту відсутності нових батьківських компонентів.

Під час виконання цього сценарію додатково здійснюється контроль уже оброблених компонентів. Це дозволяє уникнути повторної обробки однакових елементів та запобігти виникненню циклічних переходів.

Основний сценарій виконання:

1) Серверна частина отримує інформацію про компонент.
2) Виконується пошук батьківських залежностей.
3) Для кожного знайденого компонента аналізуються вкладені дані про його батьківські залежності.
4) Здійснюється перевірка вже оброблених елементів.
5) Формується повний перелік знайдених залежностей.
6) Результат передається наступному етапу обробки.

Третім варіантом використання є формування дерева залежностей. Після завершення рекурсивного пошуку серверна частина отримує набір компонентів та зв'язків між ними. Отримані дані перетворюються у структуру дерева, придатну для подальшого відображення засобами клієнтської частини.

Формування дерева залежностей є однією з основних функцій програмного продукту, оскільки саме вона забезпечує відображення повної структури батьківських компонентів. На відміну від стандартного механізму Atlassian Compass, який відображає лише один рівень залежностей, розроблена система дозволяє переглядати всю ієрархію зв'язків в межах єдиного дерева.

Основний сценарій виконання:

1) Завершується рекурсивний пошук залежностей.
2) Отримані дані групуються відповідно до структури зв'язків.
3) Формуються вузли дерева.
4) Формуються зв'язки між вузлами.
5) Створюється підсумкова структура дерева.
6) Результат передається клієнтській частині.

Також було розроблено UML-діаграму діяльності формування дерева, яку зображено на рисунку 2.2.

Рисунок 2.2 – Діаграма діяльності формування дерева залежностей

Четвертим варіантом використання є формування таблиці залежностей компонентів типу Capability. Для реалізації даного сценарію серверна частина отримує одним запитом інформацію про всі компоненти типу Capability та інші доступні компоненти системи. Після цього виконується аналіз наявності зв'язків між ними.

У результаті формується структурована таблиця, яка дозволяє оцінити ступінь зв'язаності компонентів програмної системи.

Основний сценарій виконання:

1) Серверна частина виконує запит до Compass GraphQL API.
2) Отримуються всі компоненти типу Capability.
3) Отримуються всі доступні компоненти системи.
4) Виконується аналіз зв'язків між компонентами.
5) Формується таблиця залежностей.
6) Результат повертається клієнтській частині.

Останнім варіантом використання є обробка помилок отримання даних. Даний сценарій виконується у випадках недоступності Compass GraphQL API, відсутності необхідних даних або виникнення інших виняткових ситуацій.

Під час виникнення помилки серверна частина припиняє виконання поточного сценарію, формує повідомлення про причину помилки та повертає його клієнтській частині. Такий підхід забезпечує стабільність роботи програмного забезпечення та запобігає аварійному завершенню роботи системи.

Основний сценарій виконання:

1) Виникає помилка під час виконання операції.
2) Серверна частина перехоплює виняткову ситуацію.
3) Формується повідомлення про помилку.
4) Інформація передається клієнтській частині.
5) Користувач отримує повідомлення про проблему.

У результаті розробки загальної структури програмного забезпечення було визначено основні компоненти системи, суб'єктів взаємодії та варіанти

використання серверної частини інтеграційного додатка. Запропонована структура забезпечує розділення відповідальності між клієнтською частиною, серверною частиною, платформою Atlassian Forge та Compass GraphQL API. Основна логіка програмного продукту зосереджена на серверній стороні, що дозволяє централізовано виконувати отримання, обробку та підготовку даних про залежності компонентів. Розглянуті варіанти використання охоплюють основні функції системи: отримання інформації про компонент, пошук батьківських залежностей, формування дерева залежностей, формування таблиці Capability, виконання пошуку та обробку помилкових ситуацій. Побудовані UML-діаграми дозволяють наочно представити загальну структуру системи, взаємодію користувача з програмним продуктом та послідовність виконання основних сценаріїв роботи.

2.2 Розробка системи класів

Після визначення структури програмного забезпечення та варіантів використання було виконано проєктування системи класів серверної частини інтеграційного додатка для платформи Atlassian Compass. Основною метою даного етапу є виділення структурних елементів програмного забезпечення, визначення їх функціонального призначення та встановлення взаємозв'язків між ними.

Особливістю розробленого програмного забезпечення є використання платформи Atlassian Forge, яка забезпечує виконання серверної логіки без необхідності створення окремого сервера або власної системи зберігання даних. Усі дані отримуються безпосередньо із Compass GraphQL API, а обробка запитів виконується за допомогою Forge Resolver.

З огляду на особливості архітектури Atlassian Forge під час проєктування системи класів було використано модульний підхід, а серверна логіка реалізована у вигляді окремих класів мови JavaScript, кожен з яких інкапсулює власний стан та поведінку. Кожен клас реалізує окрему частину функціональності серверної частини та взаємодіє з іншими класами через визначені методи-інтерфейси. Такий

підхід забезпечує логічне розділення відповідальності між окремими компонентами програмного забезпечення, спрощує процес його супроводу та тестування, а також дозволяє локалізувати зміни в межах окремих класів без впливу на інші програмні компоненти.

У традиційних вебзастосунках система класів часто містить сутності бази даних, репозиторії, сервіси та контролери. У розробленому програмному забезпеченні власна база даних відсутня, тому структура серверної частини складається переважно з класу-резолвера, класу взаємодії з GraphQL API та допоміжного класу обробки даних.

На основі аналізу структури проєкту було виділено такі основні логічні класи:

- ComponentPageResolver;
- GraphApiClient;
- ComponentCleaner;
- GraphQueries;
- ForgeManifest;
- CompassComponent.

Клас ComponentPageResolver є центральним елементом серверної частини додатка. Його призначенням є реєстрація серверних функцій Forge Resolver та координація взаємодії між клієнтською частиною й іншими класами системи. На відміну від попередньої функціональної реалізації, цей клас інкапсулює екземпляри допоміжних класів GraphApiClient та ComponentCleaner як власні поля, що ін'єктуються у конструкторі. Клас містить такі атрибути та методи:

Атрибути:

- resolver: Resolver;
- cleaner: ComponentCleaner;
- graphApi: GraphApiClient.

Методи:

- registerHandlers(): void;

– getComponent(context): Object;
– getAllComponents(context): Object[];
– getDefinitions(): Object.

Метод registerHandlers виконує реєстрацію обробників getComponent та getAllComponents у внутрішньому екземплярі Resolver. Метод getComponent призначений для отримання інформації про поточний компонент Atlassian Compass. Метод getAllComponents використовується для отримання списку компонентів, необхідних для формування таблиці залежностей Capability. Метод getDefinitions повертає об'єкт зареєстрованих обробників, що експортується як точка входу run для середовища Atlassian Forge.

Клас GraphApiClient призначений для взаємодії з Compass GraphQL API та інкапсулює механізми виконання GraphQL-запитів. До його складу входять такі методи:

– requestGraph(query: String, variables: Object): Object;
– fetchAllComponents(query: String, cloudId: String): Object[].

Метод requestGraph забезпечує виконання окремого GraphQL-запиту та отримання відповіді від сервера Compass, а також обробку помилок GraphQL-відповіді. Метод fetchAllComponents використовується для отримання повного набору компонентів, доступних у поточному середовищі Atlassian Compass, шляхом послідовного виконання пагінованих запитів через виклик власного методу requestGraph.

Клас ComponentCleaner виконує допоміжні функції обробки та підготовки даних перед їх передаванням клієнтській частині. До його складу входять такі методи:

– cleanComponent(data: Object, seenIds: Set): Object;
– processRelationships(rel: Object, seenIds: Set): Object[];
– cleanAllComponents(data: Object): Object[].

Методи cleanComponent та processRelationships рекурсивно взаємодіють між собою в межах одного екземпляра класу для очищення вкладеної структури

компонента від службових полів (зокрема typeMetadata) та для усунення дублікатів за допомогою множини ідентифікаторів seenIds. Метод cleanAllComponents є точкою входу для обробки повного списку компонентів і використовує cleanComponent із новоствореною множиною seenIds. Результати методів даного класу використовуються у функціях відображення дерева залежностей та таблиці Capability.

Клас GraphQueries використовується для централізованого зберігання GraphQL-запитів у вигляді статичних полів класу. До складу класу входять такі атрибути:

- GET_COMPONENT: String (static);
- SEARCH_ALL_COMPONENTS: String (static).

Виділення GraphQL-запитів в окремий клас зі статичними полями дозволяє відокремити логіку отримання даних від логіки їх обробки, уникнути створення зайвих екземплярів класу та спрощує супровід програмного забезпечення.

Клас ForgeManifest представляє конфігурацію Forge-додатка та містить параметри розгортання програмного забезпечення на платформі Atlassian Forge. До складу даного класу належать атрибути:

- module;
- functionHandler;
- resourcePath;
- runtime;
- scopes.

Зазначені параметри визначають структуру додатка, перелік дозволів та точки входу серверної частини.

Окремо доцільно виділити логічний клас CompassComponent, який використовується як модель даних компонента Atlassian Compass. Незважаючи на відсутність окремої реалізації у вигляді класу мови програмування JavaScript [8], дана структура використовується всіма класами серверної частини під час передачі інформації між компонентами системи.

До складу класу CompassComponent входять такі атрибути:

– id: String;
– name: String;
– typeId: String;
– iconUrl: String;
– relationships: Object[].

Об'єкти даного класу використовуються під час побудови дерева залежностей та формування таблиці залежностей Capability.

Взаємодія між класами серверної частини здійснюється відповідно до принципу розподілу відповідальності та з використанням ін'єкції залежностей через конструктор. Клас ComponentPageResolver під час ініціалізації створює власні екземпляри класів GraphApiClient та ComponentCleaner, координує виконання серверних функцій та делегує їм відповідні підзадачі. Клас GraphApiClient використовує статичні GraphQL-запити, що зберігаються в класі GraphQueries, та забезпечує обмін даними з Compass GraphQL API. Клас ComponentCleaner виконує підготовку отриманих даних до подальшого використання. Конфігурація та зв'язок серверної частини з платформою Atlassian Forge визначаються класом ForgeManifest.

Структура взаємозв'язків між розглянутими класами наведена на UML-діаграмі класів, зображеній на рисунку 2.3.

У результаті проєктування системи класів було визначено основні структурні елементи серверної частини програмного забезпечення, їх призначення, набір атрибутів і методів, а також взаємозв'язки між ними. Побудована UML-діаграма класів відображає архітектурну структуру серверної частини інтеграційного додатка та створює основу для подальшого опису алгоритмів роботи методів у наступному підрозділі.

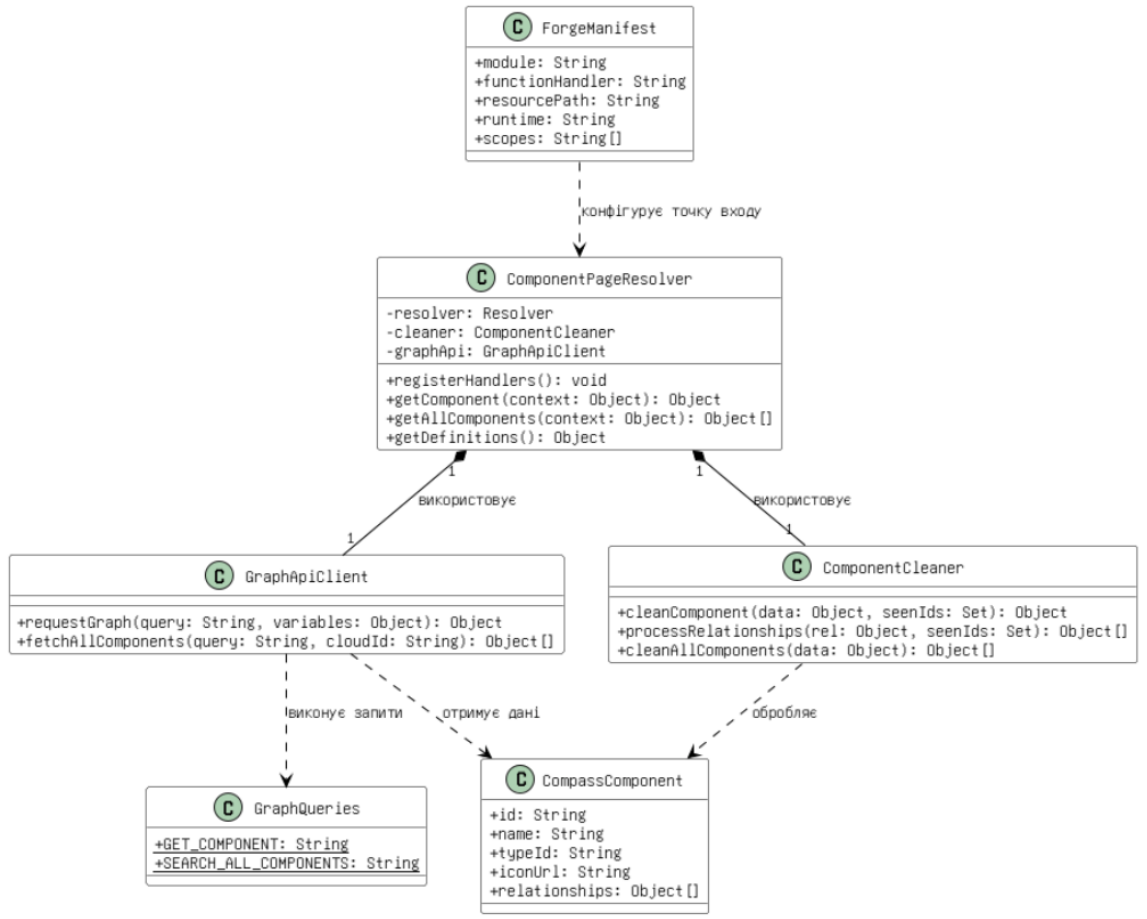


Рисунок 2.3 – UML-діаграма класів серверної частини інтеграційного додатка для платформи Atlassian Compass.

2.3 Розробка методів

Після визначення структури класів серверної частини програмного забезпечення було виконано розроблення методів, які реалізують основну функціональність інтеграційного додатка для платформи Atlassian Compass. Розроблені методи забезпечують отримання інформації про компоненти, формування структури залежностей, отримання повного переліку компонентів середовища Atlassian Compass, а також попередню обробку отриманих даних перед їх передаванням клієнтській частині.

Основна серверна логіка реалізована у вигляді методів класів ComponentPageResolver, GraphApiHelper та ResolverHelper. Взаємодія між зазначеними методами забезпечує повний цикл обробки запиту від моменту його

надходження з клієнтської частини до отримання готового результату.

Одним із ключових методів серверної частини є метод getComponent(). Його призначенням є отримання інформації про поточний компонент Atlassian Compass та його батьківські залежності. Після виклику методу із контексту Forge отримується ідентифікатор поточного компонента. Далі формується GraphQL-запит до Compass API та виконується отримання необхідних даних. Після завершення запиту результат передається до модуля обробки даних, де виконується очищення структури відповіді та видалення службової інформації. Підготовлений результат повертається клієнтській частині для подальшого формування дерева залежностей.

Метод getAllComponents() використовується під час формування таблиці залежностей Capability. На відміну від попереднього методу, він отримує не один компонент, а повний перелік компонентів середовища Atlassian Compass. Для цього використовується cloudId поточного середовища. Після отримання даних виконується їх очищення та усунення дубльованих записів. Підготовлений список компонентів передається клієнтській частині для побудови таблиці залежностей та виконання пошуку.

Для взаємодії з Compass GraphQL API використовується метод requestGraph(). Даний метод є універсальним засобом виконання GraphQL-запитів. Метод приймає текст запиту та набір параметрів, після чого виконує звернення до Compass GraphQL API від імені Forge-додатка. Після отримання відповіді виконується перевірка на наявність помилок. Якщо API повертає повідомлення про помилку, формується виняткова ситуація. У випадку успішного виконання запиту метод повертає отримані дані для подальшої обробки.

Для отримання повного списку компонентів використовується метод fetchAllComponents(). Особливістю його роботи є підтримка пагінації Compass GraphQL API. Оскільки API повертає компоненти частинами, метод виконує послідовне отримання всіх сторінок результатів до моменту, коли буде досягнуто останню сторінку даних. Після завершення отримання результатів виконується

формування єдиного списку компонентів та усунення дубльованих записів.

Алгоритм роботи методу fetchAllComponents() наведено на рисунку 2.4.

Рисунок 2.4 – Діаграма алгоритму роботи методу fetchAllComponents()

Важливу роль у роботі серверної частини відіграє метод cleanComponent(). Його призначенням є підготовка структури даних до подальшого використання у клієнтській частині. Compass GraphQL API повертає велику кількість службових полів та вкладених структур, які не використовуються під час побудови дерева залежностей. Тому перед поверненням результатів виконується їх спрощення та приведення до єдиного формату.

Особливістю методу cleanComponent() є використання рекурсивного обходу структури залежностей. Під час обробки компонента виконується перевірка його ідентифікатора. Якщо компонент уже був оброблений раніше, його повторна обробка не виконується. Такий підхід дозволяє запобігти виникненню циклічних залежностей та дублюванню вузлів у дереві.

Алгоритм очищення структури компонента та обробки його залежностей наведено у графічній частині кваліфікаційної роботи на 3 аркуші 2026.КВР.122.423.06.00.00 БС.

Подана діаграма відображає загальний принцип роботи методу cleanComponent(). Спочатку перевіряється коректність отриманих даних, після чого визначається тип структури, яка обробляється. Якщо вхідні дані є масивом, кожен його елемент проходить окрему обробку. Якщо обробляється об’єкт компонента, виконується перевірка його ідентифікатора для запобігання повторній обробці. Після цього аналізуються властивості об’єкта, а службові вкладені структури перетворюються у спрощений формат.

Для обробки залежностей використовується допоміжний метод processRelationships(). Його призначенням є формування списку батьківських компонентів та передавання їх на подальшу рекурсивну обробку. У результаті формується ієрархічна структура даних, яка використовується клієнтською частиною для побудови дерева залежностей.

Алгоритм обробки зв’язків між компонентами наведено на рисунку 2.5.

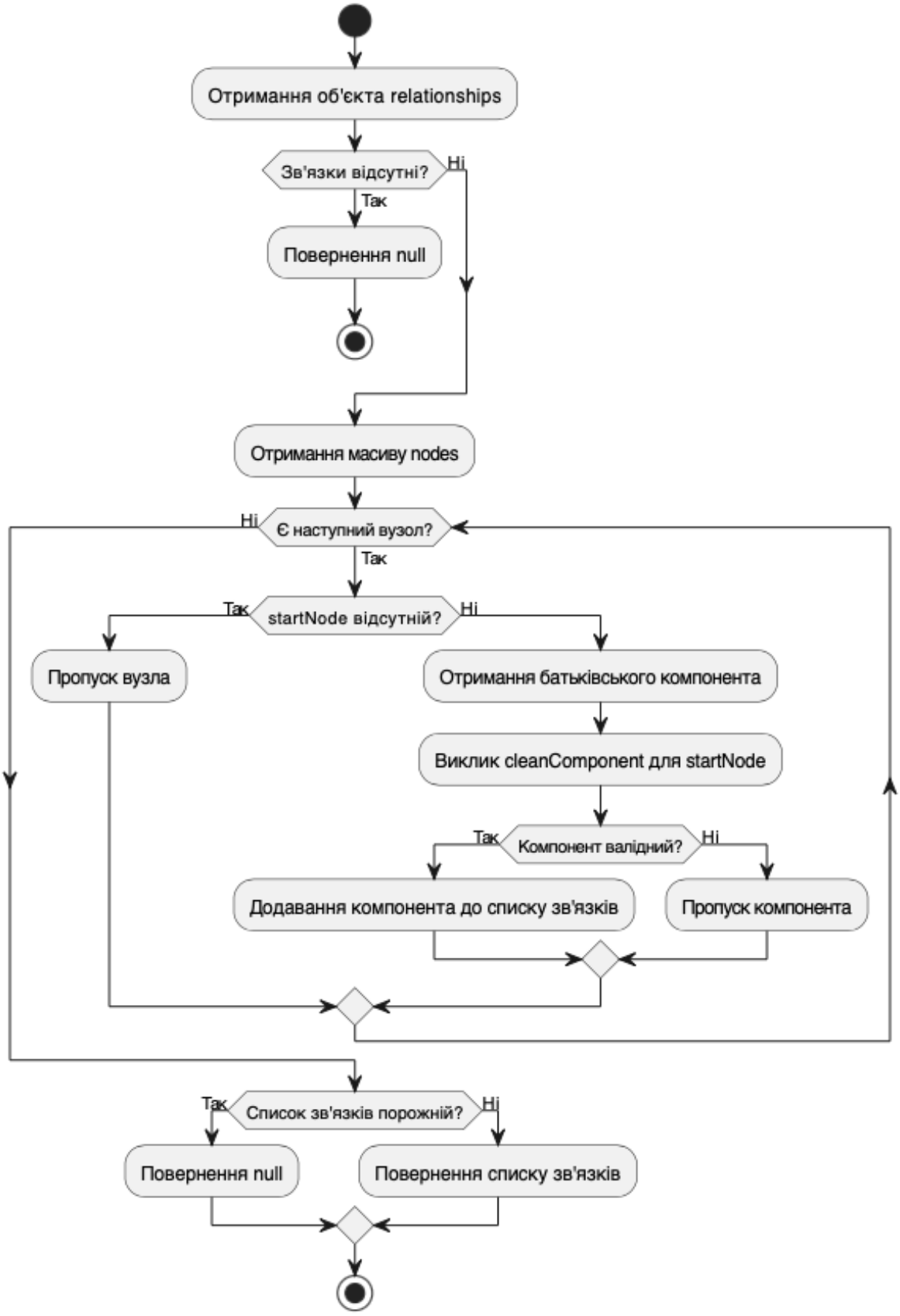


Рисунок 2.5 – UML-діаграма діяльності обробки зв’язків.

Діаграма діяльності обробки зв’язків деталізує роботу допоміжного методу processRelationships(). Метод приймає структуру зв’язків, перевіряє її наявність та послідовно обробляє кожен вузол. Основним елементом вузла є startNode, який відповідає батьківському компоненту. Якщо такий компонент наявний, він передається на подальшу обробку. У результаті формується список коректних зв’язків, який використовується під час побудови дерева залежностей.

Для обробки набору компонентів використовується метод cleanAllComponents(). Він застосовується під час формування таблиці залежностей Capability та забезпечує єдині правила очищення даних для всіх компонентів системи.

Під час розроблення дерева залежностей було реалізовано механізм рекурсивного пошуку батьківських компонентів. Пошук виконується шляхом послідовного переходу між зв’язками типу INWARD, що дозволяє визначити всі залежності поточного компонента. Для забезпечення стабільної роботи програмного забезпечення реалізовано контроль уже оброблених компонентів та захист від циклічних зв’язків між ними.

Окремою особливістю реалізації є обмеження максимальної глибини дерева залежностей десятьма рівнями. Таке обмеження встановлено безпосередньо в GraphQL-запиті та дозволяє уникнути надмірного збільшення обсягу отримуваних даних, а також забезпечує прийнятну швидкодію під час побудови дерева залежностей.

Під час формування таблиці залежностей Capability використовується повний перелік компонентів середовища Atlassian Compass. Після отримання даних клієнтська частина визначає компоненти типу Capability та аналізує наявність зв’язків між ними й іншими компонентами системи. На основі отриманих результатів виконується обчислення відсотка зв’язаних компонентів та формується підсумкова таблиця.

Розроблені методи забезпечують реалізацію всіх функціональних можливостей серверної частини інтеграційного додатка. Використання окремих

методів для отримання даних, їх обробки та формування результатів дозволило розподілити функціональність між модулями системи та підвищити зручність супроводу програмного забезпечення. Поліморфні методи та механізми наслідування в даному проєкті не використовуються, оскільки архітектура Forge-додатка побудована на основі незалежних модулів і функцій, кожна з яких виконує чітко визначене завдання.

2.4 Проєктування і опис інтерфейсу користувача

Оскільки темою дипломної роботи є розробка серверної частини інтеграційного додатка для платформи Atlassian Compass, основна увага в даному підрозділі приділяється опису програмного інтерфейсу взаємодії між клієнтською та серверною частинами системи. Користувацький інтерфейс реалізується засобами React та інтегрується безпосередньо в середовище Atlassian Compass, тому його детальна реалізація не є основним об’єктом дослідження даної роботи.

Інтеграційний додаток працює у вигляді сторінки компонента Atlassian Compass. Після відкриття сторінки клієнтська частина автоматично викликає серверні методи Forge Resolver та отримує дані, необхідні для побудови інтерфейсу користувача.

Для взаємодії клієнтської та серверної частин використовуються два основні серверні методи:

– getComponent() – отримання інформації про поточний компонент та його батьківські залежності;

– getAllComponents() – отримання списку всіх компонентів середовища Atlassian Compass.

Метод getComponent() використовується для побудови дерева залежностей. Він отримує ідентифікатор поточного компонента із контексту Atlassian Forge, виконує GraphQL-запит до Compass API та повертає очищену структуру даних, пркилад якої наведено в лістингу 2.1.

```
{
  "id": "component-001",
  "name": "Main Capability",
  "typeId": "CAPABILITY",
  "iconUrl": "https://.../capability.svg",
  "relationships": [
    {
      "startNode": {
        "id": "component-002",
        "name": "Parent Service",
        "typeId": "SERVICE",
        "iconUrl": "https://.../service.svg",
        "relationship": […]
      }
    }
  ]
}
```

Лістинг 2.1 – Структура даних для getComponent().

Основними полями даної структури є:

– id – унікальний ідентифікатор компонента;
– name – назва компонента;
– typeId – тип компонента;
– iconUrl – адреса іконки типу компонента;
– relationships – список батьківських залежностей.

На основі отриманих даних клієнтська частина формує дерево залежностей та відображає його користувачу.

Метод getAllComponents() використовується для формування таблиці залежностей Capability. Даний метод повертає список усіх компонентів, доступних

у поточному середовищі Atlassian Compass, приклад такого списку наведено в лістингу 2.2.

```
[
  {
    "component": {
      "id": "component-001",
      "name": "Main Capability",
      "typeId": "CAPABILITY",
      "iconUrl": "https://.../capability.svg",
      "relationship": […]
    }
  },
  {
    "component": {
      "id": "component-002",
      "name": "User Service",
      "typeId": "SERVICE",
      "iconUrl": "https://.../service.svg",
      "relationship": […]
    }
  }
]
```

Лістинг 2.2 – Структура даних для getAllComponents().

На основі отриманого списку компонентів клієнтська частина виконує поділ компонентів на дві групи: компоненти типу Capability та інші компоненти системи. Після цього визначається наявність або відсутність зв’язків між ними та обчислюється відсоток зв’язаних компонентів.

Інтерфейс взаємодії між клієнтською та серверною частинами побудований за принципом мінімальної залежності. Серверна частина відповідає за отримання та підготовку даних, а клієнтська частина виконує лише їх візуалізацію. Такий

підхід дозволяє відокремити логіку обробки даних від логіки відображення та спрощує подальший супровід програмного забезпечення.

Таким чином, інтерфейс користувача інтеграційного додатка базується на взаємодії клієнтської частини із серверними методами Forge Resolver. Серверна частина забезпечує отримання, обробку та передачу даних у форматі JSON, які використовуються для побудови дерева залежностей та таблиці залежностей Capability у середовищі Atlassian Compass. Перед передачею результатів клієнтській частині серверні методи перевіряють структуру відповіді Compass GraphQL API, вилучають зайві службові поля та формують уніфіковані об’єкти компонентів. До складу підготовлених даних входять ідентифікатор, назва, тип, піктограма та відомості про зв’язки компонента. Завдяки цьому клієнтська частина отримує готову структуру даних і не залежить від внутрішніх особливостей відповіді зовнішнього API.

2.5 Опис файлової структури програми

Розроблене програмне забезпечення не використовує зовнішні файли як джерело вхідних даних та не здійснює збереження результатів роботи у файли. Уся інформація отримується безпосередньо з Compass GraphQL API та обробляється під час виконання Forge-додатка. У зв’язку з цим у даному підрозділі наведено файлову структуру вихідного коду серверної частини програмного забезпечення та опис призначення основних файлів проєкту.

Серверна частина додатка побудована за модульним принципом. Такий підхід забезпечує розподіл функціональності між окремими файлами та спрощує супровід програмного забезпечення. Кожний програмний модуль виконує окрему групу завдань: реєстрацію серверних методів, обробку запитів клієнтської частини, взаємодію з Compass GraphQL API, зберігання текстів GraphQL-запитів або очищення отриманих даних. Такий розподіл зменшує взаємозалежність файлів і полегшує тестування та внесення змін до програмного забезпечення. Загальна структура файлів серверної частини наведена на рисунку 2.6.

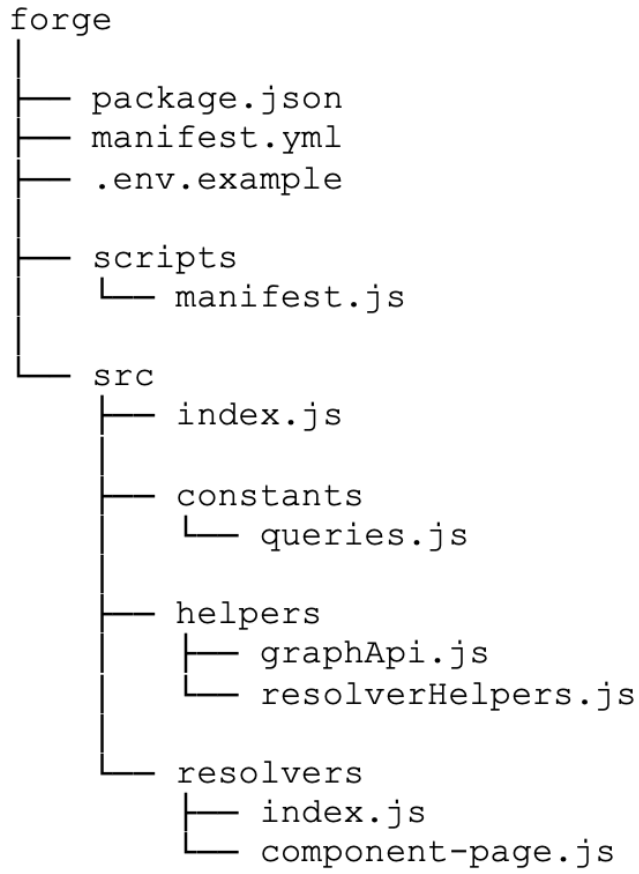


Рисунок 2.6 – Файлова структура серверної частини інтеграційного додатка для платформи Atlassian Compass

Основним конфігураційним файлом проєкту є файл manifest.yml, лістинг якого, поданий в додатку А пояснювальної записки. Він використовується платформою Atlassian Forge для визначення структури додатка, точок входу, необхідних дозволів та параметрів виконання. У даному файлі описано модуль сторінки компонента Compass, функцію-обробник запитів, налаштування середовища виконання Node.js [9] та перелік дозволів, необхідних для роботи з Compass API.

Файл package.json містить інформацію про проєкт, перелік залежностей та службові параметри, необхідні для збирання й розгортання Forge-додатка. За допомогою цього файлу виконується керування бібліотеками, що використовуються під час роботи серверної частини.

Файл .env.example використовується як приклад конфігураційного файлу

середовища. У ньому містяться шаблони параметрів, які можуть бути використані під час налаштування або розгортання програмного забезпечення в іншому середовищі.

У каталозі scripts розташований файл manifest.js. Його призначення полягає у формуванні або модифікації конфігурації Forge-додатка під час процесу розробки та збирання проєкту. Виділення допоміжних скриптів в окремий каталог дозволяє відокремити службову логіку від основного програмного коду.

Основний програмний код серверної частини розміщується в каталозі src. У даному каталозі зосереджено всі модулі, які реалізують функціональні можливості інтеграційного додатка.

Файл index.js є початковою точкою входу серверної частини. Через нього виконується підключення основних модулів та ініціалізація логіки додатка.

Каталог resolvers містить модулі, що забезпечують взаємодію між клієнтською та серверною частинами додатка.

Файл component-page.js є центральним модулем серверної частини. Саме в ньому реалізовано Forge Resolver та визначено серверні методи, які можуть викликатися клієнтською частиною. Даний файл забезпечує отримання інформації про компоненти, звернення до Compass GraphQL API та передачу підготовлених результатів користувацькому інтерфейсу. Лістинг цього файлу наведено у графічній частині кваліфікаційної роботи на 2 аркуші 2026.КВР.122.423.06.00.00 ТП та в додатку Б пояснювальної записки.

Файл index.js, розташований у каталозі resolvers, використовується для організації експорту та підключення resolver-модулів. Його наявність спрощує структуру імпорту та підвищує зручність подальшого розширення серверної частини.

До складу каталогу helpers входять файли graphApi.js та resolverHelpers.js, лістинги яких, подано в додатках В та Г пояснювальної записки.

Файл graphApi.js реалізує функції взаємодії з Compass GraphQL API. У ньому зосереджено логіку виконання GraphQL-запитів, обробки відповідей сервера

та отримання повного списку компонентів із використанням механізму пагінації.

Файл resolverHelpers.js містить допоміжні функції для підготовки та обробки даних. Основним призначенням даного модуля є очищення структур даних, обробка залежностей між компонентами та формування результатів у вигляді, придатному для подальшого використання клієнтською частиною.

У каталозі constants розташовано файл queries.js, лістинг якого подано в додатку Д пояснювальної записки. Даний файл містить GraphQL-запити, які використовуються для отримання інформації з Compass GraphQL API. Виділення запитів в окремий модуль дозволяє централізовано керувати структурами запитів та спрощує внесення змін у випадку модифікації API.

Для зручності супроводу програмного забезпечення файли проєкту згруповані за функціональним призначенням. Конфігураційні файли винесено окремо від програмного коду, GraphQL-запити розміщено в окремому модулі констант, допоміжні функції зосереджено в каталозі helpers, а серверні обробники запитів — у каталозі resolvers. Така структура відповідає принципам модульного проєктування[10] та забезпечує зручність подальшого розвитку програмного забезпечення.

Таким чином, файлова структура серверної частини інтеграційного додатка побудована за модульним принципом та забезпечує чіткий розподіл відповідальності між окремими файлами. Це спрощує навігацію по проєкту, підвищує зрозумілість програмного коду та створює умови для подальшого розширення функціональних можливостей програмного забезпечення.

2.6 Опис структури бази даних програми

Особливістю розробленого програмного забезпечення є відсутність власної локальної або віддаленої бази даних. Серверна частина інтеграційного додатка не виконує постійного зберігання інформації та не використовує системи керування базами даних для накопичення або обробки даних. Усі необхідні відомості отримуються безпосередньо із Compass GraphQL API під час виконання запитів.

Замість власної бази даних програмне забезпечення використовує інформаційну модель, сформовану на основі об’єктів, які повертаються Compass GraphQL API. Отримані дані проходять попередню обробку та використовуються для побудови дерева залежностей і таблиці залежностей Capability.

Основним об’єктом, з яким працює серверна частина, є компонент Atlassian Compass, спрощена структура якого відображена в лістингу 2.3. Після виконання GraphQL-запиту формується структура даних, що містить інформацію про компонент та його залежності.

```
{
  "data": {
    "compass": {
      "component": {
        "id": "ari:cloud:compass:.../component-001",
        "name": "Main Capability",
        "typeId": "CAPABILITY",
        "typeMetadata": {
          "iconUrl": "https://.../capability.svg"
        },
        "relationships": {
          "nodes": [ {
              "startNode": {
                "id": "ari:cloud:compass:.../component-002",
                "name": "Parent Service A",
                "typeId": "SERVICE",
                "typeMetadata": {
                  "iconUrl": "https://.../service.svg"
                },
                "relationships": { … }
              }
}}]}}}}
```

Лістинг 2.3 – Приклад спрощеної структури компонента.

Основною сутністю інформаційної моделі є компонент Compass структура даних якого наведена в таблиці 2.1.

Таблиця 2.1 – Структура даних компонента Compass

№	Поле	Тип даних	Опис
1	id	String	Унікальний ідентифікатор компонента
2	name	String	Назва компонента
3	typeId	String	Тип компонента
4	iconUrl	String	Адреса іконки типу компонента
5	relationships	Array	Масив залежностей компонента

Для представлення зв’язків між компонентами використовується структура залежності. Вона описує зв’язок між поточним компонентом та його батьківським компонентом.

Структура об’єкта startNode фактично повторює структуру компонента Compass та містить основні характеристики пов’язаного компонента.

Для формування таблиці залежностей Capability використовується список усіх компонентів, отриманих через GraphQL API. Після виконання запиту SEARCH_ALL_COMPONENTS формується набір компонентів, що містить інформацію про всі доступні елементи поточного середовища Atlassian Compass.

Після отримання даних серверна частина виконує їх нормалізацію та очищення. Зокрема зі структури GraphQL-відповіді видаляються службові поля, а вкладені метадані типів компонентів перетворюються на спрощене поле iconUrl. У результаті формується компактна структура даних, яка використовується клієнтською частиною для побудови дерева залежностей та таблиці Capability.

Центральним елементом моделі є компонент Compass, який може мати довільну кількість батьківських залежностей. Кожна залежність посилається на інший компонент системи, що дозволяє формувати ієрархічне дерево взаємозв’язків між компонентами.

Таким чином, незважаючи на відсутність власної бази даних, програмне забезпечення використовує чітко визначену структуру даних, яка надходить із Compass GraphQL API. Описані об’єкти та їх взаємозв’язки утворюють логічну інформаційну модель додатка і забезпечують реалізацію всіх функцій аналізу залежностей компонентів Atlassian Compass.

2.7 Тестування програми і результати її виконання

Тестування є одним із завершальних етапів розроблення програмного забезпечення та виконується з метою перевірки правильності реалізації функціональних можливостей, виявлення помилок та підтвердження працездатності системи. Під час розроблення серверної частини інтеграційного додатка для платформи Atlassian Compass тестування проводилося відповідно до загальноприйнятих принципів та підходів, визначених стандартом IEEE Std 29119-1-2022

Основною метою тестування було підтвердження коректності роботи серверних методів, перевірка правильності взаємодії з Compass GraphQL API, перевірка алгоритмів побудови дерева залежностей та формування таблиці залежностей Capability, а також оцінювання стійкості системи до помилкових або неповних даних.

Під час тестування використовувалися такі види тестування:

- функціональне тестування;
- інтеграційне тестування;
- тестування інтерфейсу програмування застосунків;
- тестування граничних випадків;
- тестування обробки помилок.

Функціональне тестування використовувалося для перевірки відповідності реалізованого функціоналу технічному завданню. Під час його проведення перевірялися можливість отримання інформації про поточний компонент,

побудова дерева залежностей, формування таблиці залежностей Capability та виконання пошуку.

Інтеграційне тестування застосовувалося для перевірки взаємодії між серверною частиною додатка та Compass GraphQL API. Під час тестування перевірялося коректне формування GraphQL-запитів, отримання відповідей від API та правильність обробки отриманих даних.

Окрему увагу було приділено тестуванню алгоритму побудови дерева залежностей. Під час тестування перевірялося:

- отримання інформації про поточний компонент;
- отримання батьківських залежностей;
- рекурсивна побудова дерева;
- коректне відображення вкладених залежностей;
- захист від циклічних залежностей;
- захист від повторної обробки компонентів;
- обмеження максимальної глибини дерева десятьма рівнями.

Для перевірки роботи таблиці залежностей Capability було проведено тестування процесу отримання всіх компонентів середовища Atlassian Compass, визначення компонентів типу Capability та аналізу їх взаємозв'язків з іншими компонентами системи. Додатково перевірялася правильність роботи пошуку та підрахунку відсотка пов'язаних компонентів.

Під час тестування серверної частини також перевірялася робота механізму обробки помилок. Було розглянуто такі сценарії:

- відсутність відповіді від Compass GraphQL API;
- отримання порожнього списку компонентів;
- отримання помилки GraphQL;
- наявність циклічних залежностей;
- повторна поява одного й того самого компонента в дереві залежностей.

У всіх наведених випадках серверна частина коректно обробляла помилки та запобігала аварійному завершенню роботи програмного забезпечення.

Для систематизації результатів тестування було сформовано контрольний список перевіреного функціоналу, наведений у таблиці 2.2.

Таблиця 2.2 – Контрольний список протестованого функціоналу

№	Функціональна можливість	Очікуваний результат	Результат тестування
1	Отримання інформації про компонент	Повернення інформації про компонент	Виконано успішно
2	Отримання батьківських залежностей	Повернення списку залежностей	Виконано успішно
3	Побудова дерева залежностей	Формування дерева всіх доступних рівнів	Виконано успішно
4	Захист від циклічних залежностей	Відсутність нескінченної рекурсії	Виконано успішно
5	Захист від повторної обробки	Відсутність дублікатів вузлів	Виконано успішно
6	Обмеження глибини дерева	Максимум 10 рівнів	Виконано успішно
7	Отримання всіх компонентів	Повернення списку компонентів	Виконано успішно
8	Формування таблиці Capability	Коректне формування таблиці	Виконано успішно
9	Обробка помилок GraphQL API	Коректне повідомлення про помилку	Виконано успішно

Під час тестування використовувалися реальні дані середовища Atlassian Compass. Це дозволило перевірити роботу програмного забезпечення в умовах, максимально наближених до реальної експлуатації. У процесі тестування було підтверджено коректність формування GraphQL-запитів, правильність обробки

відповідей Compass GraphQL API та стабільність роботи алгоритмів обробки залежностей.

У результаті проведеного тестування було підтверджено працездатність серверної частини інтеграційного додатка для платформи Atlassian Compass. Усі функціональні можливості, передбачені технічним завданням, реалізовано коректно та працюють відповідно до встановлених вимог. Серверна частина забезпечує стабільну взаємодію з Compass GraphQL API, коректно обробляє помилки та формує результати, необхідні для побудови дерева залежностей і таблиці залежностей Capability.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції програмного забезпечення

Розроблене програмне забезпечення являє собою серверну частину інтеграційного додатка для платформи Atlassian Compass, створену з використанням платформи Atlassian Forge. Додаток призначений для роботи всередині середовища Atlassian Compass та не потребує встановлення окремого вебсервера або системи керування базами даних. Усі серверні функції виконуються в хмарному середовищі Atlassian Forge, а дані отримуються безпосередньо через Compass GraphQL API.

Для встановлення та запуску програмного забезпечення необхідно забезпечити наявність комплексу технічних та програмних засобів, що відповідають мінімальним вимогам, які наведено в таблиці 3.1.

Таблиця 3.1 – Мінімальні апаратні вимоги

Параметр	Мінімальне значення
Процесор	Intel Core i3 або аналогічний
Оперативна пам'ять	4 ГБ
Вільне місце на диску	1 ГБ
Підключення до мережі Інтернет	Обов'язкове
Роздільна здатність екрана	1366×768 пікселів

Для комфортної роботи рекомендується використовувати персональний комп'ютер із процесором Intel Core i5 або новішим та оперативною пам'яттю не менше 8 ГБ.

Програмне забезпечення може працювати в таких операційних системах:

- Microsoft Windows 10 або новіша;
- macOS 13 Ventura або новіша;

– Linux Ubuntu 22.04 або новіша.

Під час розроблення програмного забезпечення використовувалися такі програмні засоби:

– мова програмування JavaScript;
– платформа Atlassian Forge;
– середовище виконання Node.js;
– бібліотеки Forge API[11] та Forge Resolver;

Для розгортання програмного забезпечення необхідно встановити Node.js та інструментарій Atlassian Forge CLI. Після встановлення Forge CLI виконується авторизація користувача в Atlassian за допомогою команди: forge login.

Після успішної авторизації необхідно встановити залежності проєкту: npm install[12].

Команда встановлює всі необхідні бібліотеки та модулі, перелік яких наведений у файлі package.json.

Для запуску локального середовища розробки використовується команда: forge tunnel.

Команда забезпечує зв'язок локального середовища розробки з хмарною платформою Atlassian Forge та дозволяє виконувати налагодження серверної частини в режимі реального часу.

Після завершення розроблення програмне забезпечення розгортається в середовищі Atlassian Forge за допомогою команди: forge deploy.

Для встановлення додатка в конкретне середовище Atlassian використовується команда: forge install.

Після виконання інсталяції додаток стає доступним у середовищі Atlassian Compass та інтегрується у сторінку компонента відповідно до конфігурації, визначеної у файлі manifest.yml.

Основним конфігураційним файлом програмного забезпечення є файл manifest.yml. У ньому визначаються:

– модулі Atlassian Forge;

– точки входу серверної частини;
– параметри середовища виконання;
– перелік необхідних дозволів;
– ресурси, що використовуються додатком.

Крім того, у проєкті використовується файл .env.example, який містить приклад параметрів конфігурації середовища. Даний файл може бути використаний під час розгортання програмного забезпечення в інших середовищах або під час подальшої модифікації проєкту.

Програмне забезпечення зберігається у вигляді початкового програмного коду мовою JavaScript. Під час процесу розгортання вихідний код перетворюється платформою Atlassian Forge у завантажувальний пакет та розміщується в хмарному середовищі Atlassian. Таким чином, технологічний тип представлення програмного забезпечення включає:

– початковий код мовою JavaScript;
– конфігураційні файли Forge;
– завантажувальний пакет, сформований під час розгортання;
– архівну копію проєкту для зберігання та резервного копіювання.

Оскільки програмне забезпечення працює всередині платформи Atlassian Compass та використовує хмарну інфраструктуру Atlassian Forge, користувачу не потрібно встановлювати додаткові сервери, системи керування базами даних або виконувати складні налаштування мережевого середовища. Це спрощує процес інсталяції та забезпечує швидке впровадження програмного забезпечення в існуючу інфраструктуру підприємства.

Таким чином, розроблений інтеграційний додаток має невисокі вимоги до апаратного та програмного забезпечення, легко розгортається за допомогою інструментів Atlassian Forge та може використовуватися в різних операційних системах без необхідності додаткового налаштування серверної інфраструктури.

3.2 Інструкція з використання тестових наборів

Для перевірки працездатності серверної частини інтеграційного додатка для платформи Atlassian Compass використовувався набір тестів, який охоплює всі основні функціональні можливості програмного забезпечення. Тестування проводилося в середовищі Atlassian Forge з використанням реальних даних Compass GraphQL API.

Основною метою тестування було підтвердження правильності роботи серверних методів, GraphQL-запитів, алгоритмів обробки залежностей та механізмів обробки помилок.

До складу тестового набору входять аварійні, вироджені, основні, комплексні, стикові, структурні та функціональні тести.

Аварійні тести використовуються для перевірки поведінки системи у випадку виникнення помилок під час взаємодії з Compass GraphQL API.

Під час тестування перевірялися такі ситуації:

- недоступність Compass GraphQL API;
- отримання GraphQL-помилки;
- отримання порожньої відповіді від API;
- відсутність компонента із заданим ідентифікатором;
- отримання некоректної структури даних.

У всіх наведених випадках серверна частина коректно обробляла помилки та запобігала аварійному завершенню роботи програми.

Вироджені тести застосовуються для перевірки роботи системи на мінімальних або граничних наборах даних.

Було протестовано такі випадки:

- компонент без залежностей;
- компонент із єдиною залежністю;
- порожній список залежностей;
- порожній список компонентів;

– залежність без вкладених вузлів.

Усі перевірки були виконані успішно. Алгоритми обробки даних працюють коректно навіть при мінімальному обсязі інформації.

Основні тести призначені для перевірки функціональності, передбаченої технічним завданням перелік яких наведено в таблиці 3.2.

Таблиця 3.2 – Основні тести

№	Тест	Очікуваний результат	Результат
1	Отримання інформації про компонент	Повернення інформації про компонент	Успішно
2	Отримання батьківських залежностей	Повернення списку залежностей	Успішно
3	Рекурсивна обробка залежностей	Формування дерева залежностей	Успішно
4	Отримання всіх компонентів	Повернення списку компонентів	Успішно
5	Обробка GraphQL-відповіді	Формування очищеної структури даних	Успішно
6	Обробка помилок	Коректне повідомлення про помилку	Успішно

Комплексні тести використовувалися для перевірки спільної роботи всіх модулів серверної частини.

Під час тестування перевірялася така послідовність операцій:

1) Отримання ідентифікатора компонента із контексту Forge.
2) Виклик методу getComponent.
3) Виконання GraphQL-запиту GET_COMPONENT.
4) Отримання відповіді від Compass GraphQL API.
5) Очищення та обробка структури даних.

6) Повернення результату.

Також перевірявся сценарій отримання всіх компонентів:

1) Виклик методу getAllComponents.
2) Виконання GraphQL-запиту SEARCH_ALL_COMPONENTS.
3) Отримання даних з використанням пагінації.
4) Видалення дублікатів.
5) Формування кінцевого результату.

Усі комплексні тести були виконані успішно.

Стикове тестування проводилося для перевірки взаємодії між окремими модулями серверної частини.

Було перевірено взаємодію між такими файлами:

- component-page.js та graphApi.js;
- component-page.js та resolverHelpers.js;
- graphApi.js та Compass GraphQL API;
- queries.js та graphApi.js.

Тестування показало коректну взаємодію всіх модулів та правильність передачі даних між ними.

Структурне тестування проводилося для перевірки внутрішньої логіки програмного забезпечення.

Під час тестування перевірялися:

- рекурсивний алгоритм обробки залежностей;
- механізм захисту від циклічних залежностей;
- механізм захисту від повторної обробки компонентів;
- алгоритм очищення GraphQL-відповідей;
- алгоритм отримання компонентів із використанням пагінації.

Результати тестування показали, що всі алгоритми працюють коректно та забезпечують стабільну роботу серверної частини навіть при складній структурі залежностей.

Для перевірки відповідності програмного забезпечення технічному

завданню було сформовано контрольний список функціональних можливостей, наведений у таблиці 3.3.

Таблиця 3.3 – Контрольний список протестованого функціоналу

№	Функціональна можливість	Статус
1	Отримання інформації про компонент	Виконано
2	Отримання батьківських залежностей	Виконано
3	Захист від циклічних залежностей	Виконано
4	Захист від повторної обробки компонентів	Виконано
5	Отримання всіх компонентів	Виконано
6	Використання пагінації	Виконано
7	Очищення GraphQL-відповіді	Виконано
8	Обробка помилок GraphQL API	Виконано

У результаті проведеного тестування було підтверджено працездатність серверної частини інтеграційного додатка для платформи Atlassian Compass. Усі функціональні можливості, передбачені технічним завданням, реалізовано повністю та працюють відповідно до встановлених вимог. Серверна частина забезпечує стабільну взаємодію з Compass GraphQL API, коректно обробляє помилки та виконує обробку залежностей компонентів.

3.3 Інструкція з експлуатації програмного комплексу

Розроблене програмне забезпечення являє собою серверну частину інтеграційного додатка для платформи Atlassian Compass, реалізовану з використанням платформи Atlassian Forge. Програмний комплекс призначений для отримання інформації про компоненти Atlassian Compass, аналізу їх залежностей та підготовки структурованих даних для подальшого використання.

Особливістю програмного комплексу є відсутність окремого сервера,

власної бази даних та локального сховища даних. Усі серверні функції виконуються в хмарному середовищі Atlassian Forge, а необхідна інформація отримується безпосередньо з Compass GraphQL API.

Доступ до функціональних можливостей програмного комплексу здійснюється через Atlassian Compass. Після відкриття сторінки компонента автоматично викликаються серверні методи Forge Resolver, які виконують отримання та обробку даних.

Основними методами серверної частини є:

– getComponent – отримання інформації про компонент та його батьківські залежності;

– getAllComponents – отримання повного списку компонентів середовища Atlassian Compass;

– requestGraph – виконання GraphQL-запитів до Compass GraphQL API;

– fetchAllComponents – отримання компонентів із використанням механізму пагінації;

– cleanComponent – очищення та нормалізація структури даних;

– cleanAllComponents – очищення набору компонентів.

Для початку роботи програмний комплекс повинен бути розгорнутий у середовищі Atlassian Forge та встановлений у відповідне середовище Atlassian Compass. Після інсталяції користувачеві не потрібно виконувати додаткових налаштувань або запускати окремі серверні процеси.

Під час експлуатації програмного комплексу необхідно забезпечити:

– наявність доступу до мережі Інтернет;

– доступність сервісів Atlassian Forge;

– доступність Compass GraphQL API;

– наявність дозволів read:component:compass та write:component:compass, визначених у файлі manifest.yml.

Файли програмного комплексу рекомендується розміщувати відповідно до структури проєкту, наведеної в підрозділі 2.5. Основні серверні модулі повинні

знаходитися в каталозі src, а конфігураційні файли – у кореневому каталозі проєкту.

Файл manifest.yml є основним конфігураційним файлом програмного комплексу та містить інформацію про модулі Forge, параметри середовища виконання та перелік дозволів, необхідних для роботи з Compass API.

Файл queries.js містить GraphQL-запити, які використовуються для отримання інформації про компоненти та їх залежності. У файлі graphApi.js реалізовано функції взаємодії з Compass GraphQL API, а файл resolverHelpers.js забезпечує очищення та нормалізацію отриманих даних.

Під час експлуатації програмного комплексу використання оперативної пам’яті визначається кількістю компонентів та глибиною їх залежностей. Оскільки серверна частина виконується в середовищі Atlassian Forge, максимальний обсяг оперативної пам’яті для функцій визначається конфігурацією, наведеною у файлі manifest.yml. У розробленому програмному забезпеченні для середовища виконання виділено 256 МБ оперативної пам’яті, чого достатньо для виконання GraphQL-запитів, обробки отриманих даних та формування структур залежностей.

Обсяг завантажувальних модулів програмного комплексу є незначним, оскільки основна логіка реалізована мовою JavaScript та складається з окремих модулів, згрупованих за функціональним призначенням. Завдяки використанню платформи Atlassian Forge процес завантаження, оновлення та виконання серверної частини здійснюється автоматично без необхідності ручного адміністрування серверної інфраструктури.

Таким чином, експлуатація розробленого програмного комплексу не потребує складного налаштування або спеціального обладнання. Програмне забезпечення інтегрується в середовище Atlassian Compass, автоматично взаємодіє з Compass GraphQL API та забезпечує стабільне виконання серверних функцій, передбачених технічним завданням.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини даного дипломного проєкту є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки серверної частини інтеграційного додатка для платформи Atlassian Compass, прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Об’єктом розробки є серверна частина інтеграційного додатка для платформи Atlassian Compass.

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 НДР

У цьому підрозділі визначено основні етапи розробки серверної частини інтеграційного додатка для платформи Atlassian Compass та розраховано тривалість їх виконання. Оцінювання витрат часу дає змогу визначити загальну трудомісткість створення програмного продукту та є основою для подальших економічних розрахунків. Перелік виконуваних робіт і відповідні витрати часу наведено в таблиці 4.1.

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 Pm	10
		Інженер (l1)	7
2	Розробка технічного завдання	Кер. проєкту (Pm)	9
		Інженер (l1)	5
3	Дизайн інтерфейсу	Інженер (l1)	14
		Інженер (l2)	25
4	Розробка функціоналу	Інженер (l1)	45
5	Тестування та відладка	Тестувальник	12
6	Документування	Інженер (l1)	3
7	Розгортання та підтримка	Інженер (l2)	18
8	Управління проєктом	Кер. проєкту (Pm)	22
Разом			170

Сумарний час виконання операцій технологічного процесу становить 170 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 серверної частини інтеграційного додатка для платформи Atlassian Compass

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:

$$З_{осн.} = Т_с * К_г \quad (4.1)$$

де: T_c – тарифна ставка, грн. (приймаємо для керівника проєкту (Pm) – 470 грн./год, інженера (l2) – 264 грн./год.), інженера (l1) – 121 грн./год., тестувальник – 110 грн./год.; $К_г$ – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

Керівника проекту (Pm) $З_{осн}$= 41 * 470 = 19 270 грн.

Інженера (l2) $З_{осн}$= 43 * 264 = 11 352 грн.

Інженера (l1) $З_{осн}$= 74 * 121 = 8 954 грн.

Тестувальник $З_{осн}$= 12 * 110 = 1 320 грн.

Сумарна основна заробітна плата становить

$З_{осн}$ = 19 270 + 11 352 + 8 954 + 1 320 = 40 896 грн.

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

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

де: $К_{допл.}$ – коефіцієнт додаткових виплат працівникам.

Отже додаткова заробітна плата по категоріях працівників становить:

Керівника проекту $З_{дод}$= 19 270 * 0,1 = 1 927 грн.

Інженера (l2) $З_{дод}$= 11 352 * 0,1 = 1 135 грн.

Інженера (l1) $З_{дод}$= 8 954 * 0,1 = 895 грн.

Тестувальник $З_{дод}$= 1 320 * 0,1 = 132 грн.

Загальна додаткова заробітна плата становить:

$З_{дод}$= 1 927 + 1 135 + 895 + 132 = 4 089 грн.

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

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п} = 40\ 896 + 4\ 089 = \quad 44\ 985 \text{ грн.}$$

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

$$В_{єсв} = Воп * 0{,}22 \quad (4.4)$$

$$В_{єсв} = 44\ 985 * 0{,}22 = 9\ 897 \text{ грн.}$$

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
		Тарифна ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
		1	2	3	4	5	6
1	Кер. проєкту (Pm)	470	41	19 270	1 927		
2	Інженера (l2)	264	43	11 352	1 135		
3	Інженера (l1)	121	74	8 954	895		
4	Тестувальник	110	12	1 320	132		
Разом				40 896	4 089	9 897	54 882

Отже, загальні витрати на оплату праці становлять 54 882 грн.

4.3 Розрахунок витрат на електроенергію

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

$$З_в = W * Т * S \quad (4.5)$$

де: W – необхідна потужність, кВт; Т – кількість годин роботи обладнання; 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 (згідно табл. 4.1) і споживані потужності наступні: комп'ютер – 0,82 кВт/год.

$З_{ек}$= 0,82*170*15,94= 2 222 грн.

Витрати на електроенергію становлять 2 222 грн.

4.4 Розрахунок суми амортизаційних відрахувань серверної частини інтеграційного додатка для платформи Atlassian Compass

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:

$$А = \frac{Б_В * Н_А}{100\%} * T, \quad (4.6)$$

де: А – амортизаційні відрахування за звітний період, грн.; $Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.; $Н_А$ – норма амортизації, 0,04 %.

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

$$А = \frac{50\,000{,}00 * 0{,}04}{150} * 170 = 2\,266 \text{ грн.}$$

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.

$$Н_в = В_{о.п.} * \ 0{,}2..0{,}6 \qquad (4.7)$$

де: $Н_В$ – накладні витрати.

$$Н_В = 44\,985 * 0{,}4 = 17\,994 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості серверної частини інтеграційного додатка для платформи Atlassian Compass

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.3.

Таблиця 4.3 - Кошторис витрат серверної частини інтеграційного додатка для платформи Atlassian Compass

№	Зміст витрат	Сума, грн.	В % до загальної суми
1.	Витрати на оплату праці	54 882	71
2.	Витрати на електроенергію	2 222	3
3.	Амортизаційні відрахування	2 266	3
4.	Накладні витрати	17 994	23
5.	Собівартість	77 364	100

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$Св = Во.п + Вс.з + Зе + А + Нв \quad (4.8)$$

Отже, собівартість дорівнює $С_В$= 77 364 грн.

4.7 Розрахунок ціни серверної частини інтеграційного додатка для платформи Atlassian Compass

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:

$$Ц = Св * (1+Р_{рен}) * (1+ПДВ), \quad (4.9)$$

де: Св – собівартість; $Р_{рен.}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість.

Ц = 77 364*(1+0,3)*(1+0,2)= 120 687 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).

$$ЧТВ = -Св + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad (4.10)$$

де: Св – собівартість розробки; $Г_п$ – грошовий потік за t – ий рік; t – відповідний рік проекту; і – величина дисконтної ставки (10…15%).

$$ЧТВ = -77\,364 + \frac{43323}{(1+0{,}1)^1} + \frac{43323}{(1+0{,}1)^2} + \frac{43323}{(1+0{,}1)^3} = 30\,104 \text{ грн}$$

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

Термін окупності визначається за формулою:

$$T_{ОК} = T_{ПВ} + \frac{Н_В}{Г_{ПР}} \quad (4.11)$$

де: $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + \frac{2\,469}{43323} = 2{,}1 \text{ р.}$$

Основні результати економічних розрахунків узагальнено у таблиці техніко-економічних показників. Її наведено у графічній частині кваліфікаційної роботи на 4 аркуші 2026.КВР.122.423.06.00.00 ТБ.

Прибутковість проєкту та термін окупності свідчать про його фінансову

ефективність та здатність повернути капітальні вкладення протягом 2,1 року. Отже, на основі отриманих показників можна зробити висновок, що розробка серверної частинии інтеграційного додатка для платформи Atlassian Compass є доцільною з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Створення на підприємстві служби охорони праці

Під час розроблення програмного забезпечення важливе значення має забезпечення безпечних та комфортних умов праці працівників, які виконують роботи із застосуванням персональних комп'ютерів та сучасних інформаційних технологій. Розроблення серверної частини інтеграційного додатка для платформи Atlassian Compass пов'язане з тривалою роботою інженера-програміста за комп'ютером, що обумовлює необхідність дотримання вимог законодавства України у сфері охорони праці.

Основним нормативно-правовим актом, який регулює питання охорони праці в Україні, є Закон України «Про охорону праці». Згідно з даним законом роботодавець зобов'язаний створити на робочих місцях умови праці відповідно до нормативно-правових актів, а також забезпечити дотримання вимог законодавства щодо прав працівників у галузі охорони праці.

На підприємствах, установах та організаціях для організації та контролю виконання вимог охорони праці створюється служба охорони праці. Основними завданнями цієї служби є організація системи управління охороною праці, контроль за дотриманням працівниками вимог нормативних документів, проведення профілактичних заходів щодо запобігання виробничому травматизму та професійним захворюванням.

Служба охорони праці створюється відповідно до чисельності працівників підприємства. На великих підприємствах вона функціонує як самостійний структурний підрозділ, а на невеликих підприємствах її функції можуть виконувати спеціалісти, які мають відповідну підготовку та пройшли перевірку знань з питань охорони праці.

Основними функціями служби охорони праці є:

- розроблення комплексних заходів щодо покращення умов праці;
- проведення вступного та періодичного інструктажів;

– контроль за дотриманням працівниками вимог нормативних документів;
– організація навчання працівників з питань охорони праці;
– участь у розслідуванні нещасних випадків та професійних захворювань;
– контроль технічного стану обладнання та робочих місць;
– участь у проведенні атестації робочих місць за умовами праці.

Для підприємств, діяльність яких пов’язана з розробленням програмного забезпечення, служба охорони праці повинна приділяти особливу увагу організації робочих місць працівників, які постійно працюють із персональними комп’ютерами. До основних небезпечних та шкідливих виробничих факторів для інженерів-програмістів належать:

– тривале статичне навантаження;
– перенапруження органів зору;
– нервово-емоційне навантаження;
– недостатня або надмірна освітленість робочого місця;
– несприятливі параметри мікроклімату;
– електромагнітні випромінювання комп’ютерної техніки;
– підвищене психоемоційне навантаження.

Під час розроблення серверної частини інтеграційного додатка для платформи Atlassian Compass інженер-програміст значну частину робочого часу проводить за персональним комп’ютером, виконуючи проєктування архітектури програмного забезпечення, розроблення серверних модулів, написання GraphQL-запитів та тестування функціональних можливостей додатка. У зв’язку з цим важливим завданням служби охорони праці є забезпечення безпечних умов праці, які сприяють підтриманню високої працездатності та збереженню здоров’я працівників.

Одним із важливих напрямів діяльності служби охорони праці є організація навчання працівників. Працівники повинні проходити вступний інструктаж, первинний інструктаж на робочому місці, повторний, позаплановий та цільовий інструктажі залежно від характеру виконуваних робіт. Особлива увага

приділяється правилам роботи з електрообладнанням, порядку дій у разі виникнення аварійних ситуацій та пожеж, а також дотриманню санітарно-гігієнічних вимог.

Крім того, служба охорони праці здійснює контроль за виконанням заходів, спрямованих на покращення умов праці, проводить аналіз причин виробничого травматизму та професійних захворювань, а також бере участь у розробленні рекомендацій щодо підвищення рівня безпеки праці на підприємстві.

Таким чином, створення та ефективне функціонування служби охорони праці є важливою складовою діяльності сучасного підприємства, що займається розробленням програмного забезпечення. Забезпечення безпечних та комфортних умов праці сприяє підвищенню продуктивності праці, зменшенню ризику професійних захворювань та створює передумови для успішного виконання робіт із розроблення серверної частини інтеграційного додатка для платформи Atlassian Compass.

5.2 Загальні заходи та засоби нормалізації параметрів мікроклімату

Під час розроблення серверної частини інтеграційного додатка для платформи Atlassian Compass інженер-програміст виконує роботи, пов’язані з тривалим перебуванням за персональним комп’ютером у приміщенні офісного типу. Ефективність праці та стан здоров’я працівника значною мірою залежать від параметрів мікроклімату виробничого приміщення. Несприятливі умови можуть призводити до зниження працездатності, швидкої втомлюваності, погіршення концентрації уваги та виникнення професійних захворювань.

Мікроклімат виробничих приміщень характеризується сукупністю фізичних параметрів навколишнього середовища, до яких належать температура повітря, відносна вологість повітря, швидкість руху повітря та інтенсивність теплового випромінювання. Відповідно до вимог ДСН 3.3.6.042-99 «Державні санітарні норми мікроклімату виробничих приміщень» для працівників, які виконують роботи сидячи без значного фізичного навантаження, повинні

забезпечуватися оптимальні або допустимі параметри мікроклімату.

До категорії таких працівників належать інженери-програмісти, системні аналітики та інші спеціалісти, діяльність яких пов’язана з розробленням програмного забезпечення. Тому під час організації робочого місця розробника серверної частини інтеграційного додатка необхідно забезпечити відповідність параметрів мікроклімату встановленим нормативам.

Оптимальні значення параметрів мікроклімату для приміщень, у яких виконуються роботи за персональним комп’ютером, наведено в таблиці 5.1.

Таблиця 5.1 – Оптимальні параметри мікроклімату для приміщень з персональними комп’ютерами

Параметр	Холодний період року	Теплий період року
Температура повітря, °С	22–24	23–25
Відносна вологість, %	40–60	40–60
Швидкість руху повітря, м/с	до 0,1	до 0,2

Підтримання оптимальної температури повітря є одним із найважливіших чинників забезпечення комфортних умов праці. При зниженні температури збільшується тепловіддача організму, що може призвести до переохолодження та зниження працездатності. Підвищення температури, навпаки, викликає перегрів організму, сонливість та погіршення концентрації уваги. Для працівників, які займаються розробленням програмного забезпечення, підтримання оптимальної температури є особливо важливим, оскільки їхня діяльність пов’язана з високим розумовим навантаженням.

Важливим параметром мікроклімату є також відносна вологість повітря. Зниження вологості призводить до пересихання слизових оболонок очей та дихальних шляхів, що негативно впливає на самопочуття працівників, які тривалий час працюють за моніторами. Надмірна вологість, у свою чергу, погіршує тепловіддачу організму та створює відчуття дискомфорту. Оптимальною

вважається вологість у межах від 40 до 60 відсотків.

Швидкість руху повітря також впливає на тепловий стан людини. Надмірний рух повітря викликає відчуття протягів і може стати причиною простудних захворювань, тоді як недостатня циркуляція повітря погіршує умови теплообміну та сприяє накопиченню вуглекислого газу в приміщенні. Для офісних приміщень рекомендується підтримувати швидкість руху повітря в межах до 0,1 м/с у холодний період року та до 0,2 м/с у теплий період.

Для забезпечення нормативних параметрів мікроклімату застосовується комплекс організаційних та технічних заходів. До організаційних заходів належать:

- раціональна організація робочого часу та часу відпочинку;
- встановлення регламентованих перерв під час роботи за комп'ютером;
- контроль стану виробничого середовища;
- проведення періодичних перевірок параметрів мікроклімату;
- дотримання санітарно-гігієнічних вимог до приміщень.

Технічні заходи передбачають використання спеціальних засобів підтримання комфортних умов праці. До них належать:

- системи природної вентиляції;
- системи механічної вентиляції;
- системи кондиціонування повітря;
- системи центрального або автономного опалення;
- зволожувачі повітря;
- прилади контролю температури та вологості.

Природна вентиляція забезпечується через вікна, кватирки та вентиляційні отвори. Однак у сучасних офісних приміщеннях її часто недостатньо для підтримання необхідних параметрів мікроклімату. Тому широко використовуються системи механічної вентиляції, які забезпечують примусову подачу свіжого повітря та видалення забрудненого.

Для підтримання комфортної температури в приміщеннях, де здійснюється розроблення програмного забезпечення, застосовуються системи кондиціонування

повітря. Вони дозволяють автоматично підтримувати задані параметри температури та вологості незалежно від пори року та зовнішніх погодних умов. Використання кондиціонерів особливо актуальне у літній період, коли підвищення температури може негативно впливати на продуктивність праці програмістів.

Не менш важливим є правильне розташування робочих місць у приміщенні. Робочі місця необхідно розміщувати таким чином, щоб уникнути впливу прямих сонячних променів, протягів та локальних джерел тепла. Монітори персональних комп’ютерів повинні розташовуватися таким чином, щоб не виникало відблисків від вікон або освітлювальних приладів.

Під час розроблення серверної частини інтеграційного додатка для платформи Atlassian Compass інженер-програміст виконує складні інтелектуальні завдання, пов’язані з проєктуванням архітектури програмного забезпечення, розробленням GraphQL-запитів, реалізацією серверних модулів та тестуванням програмного коду. Тому створення комфортних умов праці та підтримання нормативних параметрів мікроклімату є важливими факторами забезпечення високої продуктивності праці та збереження здоров’я працівників.

Таким чином, нормалізація параметрів мікроклімату виробничих приміщень досягається шляхом застосування комплексу організаційних та технічних заходів, спрямованих на підтримання оптимальної температури, вологості та швидкості руху повітря. Дотримання встановлених нормативів створює безпечні та комфортні умови праці для інженерів-програмістів, діяльність яких пов’язана з розробленням серверної частини інтеграційних програмних систем.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено серверну частину інтеграційного додатка для платформи Atlassian Compass. Основною метою роботи було створення програмного забезпечення, яке забезпечує отримання та обробку інформації про залежності між компонентами програмних систем із використанням можливостей Compass GraphQL API.

У процесі виконання роботи було проведено аналіз предметної області, існуючих підходів до представлення залежностей між програмними компонентами та функціональних можливостей платформи Atlassian Compass. За результатами аналізу було визначено основні вимоги до програмного забезпечення та сформовано технічне завдання на розробку серверної частини інтеграційного додатка.

У ході проєктування було розроблено структуру програмного забезпечення, визначено основні модулі системи, їх взаємозв’язки та принципи взаємодії з Compass GraphQL API. Реалізовано серверні методи для отримання інформації про компоненти, обробки батьківських залежностей, формування дерева залежностей та отримання повного списку компонентів середовища Atlassian Compass.

Розроблений програмний комплекс забезпечує побудову дерева залежностей із використанням рекурсивного пошуку, підтримує захист від циклічних залежностей та повторної обробки компонентів, а також дозволяє формувати таблицю залежностей компонентів типу Capability із розрахунком рівня їх зв’язаності. Усі дані отримуються безпосередньо через Compass GraphQL API без використання власної бази даних або проміжного сховища інформації.

Під час виконання роботи було проведено комплексне тестування програмного забезпечення, яке включало функціональне, структурне, стикове, комплексне, аварійне та вироджене тестування. Результати тестування підтвердили коректність реалізованих алгоритмів, стабільність роботи серверної частини та відповідність програмного забезпечення вимогам технічного завдання.

Практична цінність розробленого програмного забезпечення полягає у

спрощенні аналізу структури залежностей між компонентами програмних систем, підвищенні наочності представлення інформації та розширенні стандартних можливостей платформи Atlassian Compass щодо роботи із залежностями компонентів.

ПЕРЕЛІК ПОСИЛАНЬ

1) Марціяш Г.Я., Слободян Р.О. Методичні вказівки до виконання кваліфікаційної роботи для здобувачів фахової передвищої освіти спеціальності 122 «Комп'ютерні науки». Тернопіль : ВСП «ТФК ТНТУ ім. І. Пулюя».

2) Atlassian Compass Documentation : вебсайт. URL: https://developer.atlassian.com/cloud/compass/ (дата звернення: 02.12.2025).

3) Erl T., Puttini R., Mahmood Z. Cloud Computing: Concepts, Technology & Architecture. Upper Saddle River : Prentice Hall, 2013. 528 p

4) Atlassian. Compass GraphQL API : вебсайт. URL: https://developer.atlassian.com/cloud/compass/graphql/ (дата звернення: 04.12.2025).

5) Atlassian. Forge Platform Documentation : вебсайт. URL: https://developer.atlassian.com/platform/forge/ (дата звернення: 07.12.2025).

6) GraphQL Foundation. GraphQL Specification : вебсайт. URL: https://spec.graphql.org/ (дата звернення: 08.12.2025).

7) diagrams.net (draw.io). Онлайн-сервіс для створення структурних схем та блок-діаграм : вебсайт. URL: https://app.diagrams.net/ (дата звернення: 22.05.2026).

8) MDN Web Docs. JavaScript Reference : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 02.12.2025).

9) Node.js Documentation : вебсайт. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 02.12.2025).

10) Freeman E., Robson E. Head First Design Patterns. Sebastopol : O'Reilly Media, 2020. 694 p.

11) Atlassian. Forge API : вебсайт. URL: https://developer.atlassian.com/platform/forge/apis-reference/ (дата звернення: 10.12.2025).

12) What is npm : вебсайт. URL: https://www.w3schools.com/whatis/whatis_npm.asp (дата звернення: 13.12.2025).

13) IEEE Std 29119-1-2022. Software and systems engineering – Software testing – Part 1: Concepts and definitions.

14) Закон України «Про охорону праці» № 2694-XII від 14.10.1992 р.

15) ДСН 3.3.6.042-99. Державні санітарні норми мікроклімату виробничих приміщень.

ДОДАТКИ

Додаток А. manifest.yml

```
modules:
  compass:componentPage:
    - key: compass-map-component-page
      resource: component
      resolver:
        function: component
      title: compass-map
  function:
    - key: component
      handler: resolvers/component-page.run
resources:
  - key: component
    path: ../client/dist
app:
  id: ari:cloud:ecosystem::app/cf0d7f21-64c3-4450-896f-01ee7207d117
  runtime: name: nodejs24.x
    memoryMB: 256
    architecture: arm64
permissions:
  external:
    images:
      - https://compass-ui.prod-east.frontend.public.atl-paas.net
  content:
    styles:
      - unsafe-inline
    scripts:
      - unsafe-inline
  scopes:
    - read:component:compass
    - write:component:compass
```

Додаток Б. component-page.js

```
import Resolver from '@forge/resolver';
import { ComponentCleaner } from '../helpers/resolverHelpers.js';
import { GraphQueries } from '../constants/queries.js';
import { GraphApiClient } from '../helpers/graphApi.js';

class ComponentPageResolver {
    constructor() {
        this.resolver = new Resolver();
        this.cleaner = new ComponentCleaner();
        this.graphApi = new GraphApiClient();

        this.registerHandlers();
    }

    registerHandlers() {
        this.resolver.define('getComponent', this.getComponent.bind(this));
        this.resolver.define('getAllComponents',
this.getAllComponents.bind(this));
    }

    async getComponent({ context }) {
        const componentId = context.extension.componentId;

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

        return this.cleaner.cleanComponent(componentResult);
    }

    async getAllComponents({ context }) {
        const { cloudId } = context;
```

```
        const allComponents = await
this.graphApi.fetchAllComponents(GraphQueries.SEARCH_ALL_COMPONENTS,
cloudId);

        return this.cleaner.cleanAllComponents(allComponents);
    }

    getDefinitions() {
        return this.resolver.getDefinitions();
    }
}

const componentPageResolver = new ComponentPageResolver();

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

Додаток В. graphApi.js

```
import api from '@forge/api';

export class GraphApiClient {
    async requestGraph(query, variables) {
        const response = await api.asApp().requestGraph(query, variables, {
            'X-ExperimentalApi': ['compass-beta'],
        });

        const body = await response.json();

        if (body.errors?.length) {
            console.error('GraphQL errors:', body.errors);
            throw new Error(`GraphQL помилка: ${body.errors[0].message}`);
        }

        return body.data;
    }

    async fetchAllComponents(query, cloudId) {
        const allComponents = [];
        let after = null;
        let hasNextPage = true;

        while (hasNextPage) {
            const variables = {
                cloudId,
                query: {
                    first: 100,
                    after,
                },
            };
```

```
            const data = await this.requestGraph(query, variables);
            const searchResult = data?.compass?.searchComponents;

            if (!searchResult) {
                throw new Error('Нема даних');
            }

            if (searchResult.nodes) {
                allComponents.push(...searchResult.nodes);
            }

            hasNextPage = searchResult.pageInfo?.hasNextPage ?? false;
            after = searchResult.pageInfo?.endCursor ?? null;
        }

        const uniqueComponents = Array.from(new Map(allComponents.map((item)
=> [item.component.id, item])).values());

        return uniqueComponents;
    }
}

const defaultGraphApiClient = new GraphApiClient();

export function requestGraph(query, variables) {
    return defaultGraphApiClient.requestGraph(query, variables);
}

export function fetchAllComponents(query, cloudId) {
    return defaultGraphApiClient.fetchAllComponents(query, cloudId);
}
```

Додаток Г. resolverHelpers.js

```
export class ComponentCleaner {
    cleanComponent(data, seenIds = null) {
        if (!data || typeof data !== 'object') {
            return data;
        }

        if (Array.isArray(data)) {
            return     data.map((item)     =>     this.cleanComponent(item,
seenIds)).filter(Boolean);
        }

        const cleaned = {};

        if (seenIds) {
            const componentId = data?.component?.id;
            if (componentId) {
                if (seenIds.has(componentId)) {
                    return null;
                }
                seenIds.add(componentId);
            }
        }

        for (const [key, value] of Object.entries(data)) {
            if (key === 'typeMetadata') {
                if (value?.iconUrl) {
                    cleaned.iconUrl = value.iconUrl;
                }
                continue;
            }

            if (key === 'relationships') {
```

```
                const processed = this.processRelationships(value, seenIds);
                if (processed && processed.length > 0) {
                    cleaned.relationships = processed;
                }
                continue;
            }

            cleaned[key] = this.cleanComponent(value, seenIds);
        }

        return cleaned;
    }

    processRelationships(rel, seenIds) {
        if (!rel || !Array.isArray(rel.nodes) || rel.nodes.length === 0) {
            return null;
        }

        const cleanedNodes = rel.nodes
            .map((node) => {
                if (!node?.startNode) return null;

                const                    cleanedStartNode                    =
this.cleanComponent(node.startNode, seenIds);
                if (!cleanedStartNode) return null;

                return { startNode: cleanedStartNode };
            })
            .filter(Boolean);

        return cleanedNodes.length > 0 ? cleanedNodes : null;
    }

    cleanAllComponents(data) {
```

```
        return this.cleanComponent(data, new Set());
    }
}

const defaultComponentCleaner = new ComponentCleaner();

export function cleanComponent(data, seenIds = null) {
    return defaultComponentCleaner.cleanComponent(data, seenIds);
}

export function cleanAllComponents(data) {
    return defaultComponentCleaner.cleanAllComponents(data);
}
```

Додаток Д. queries.js

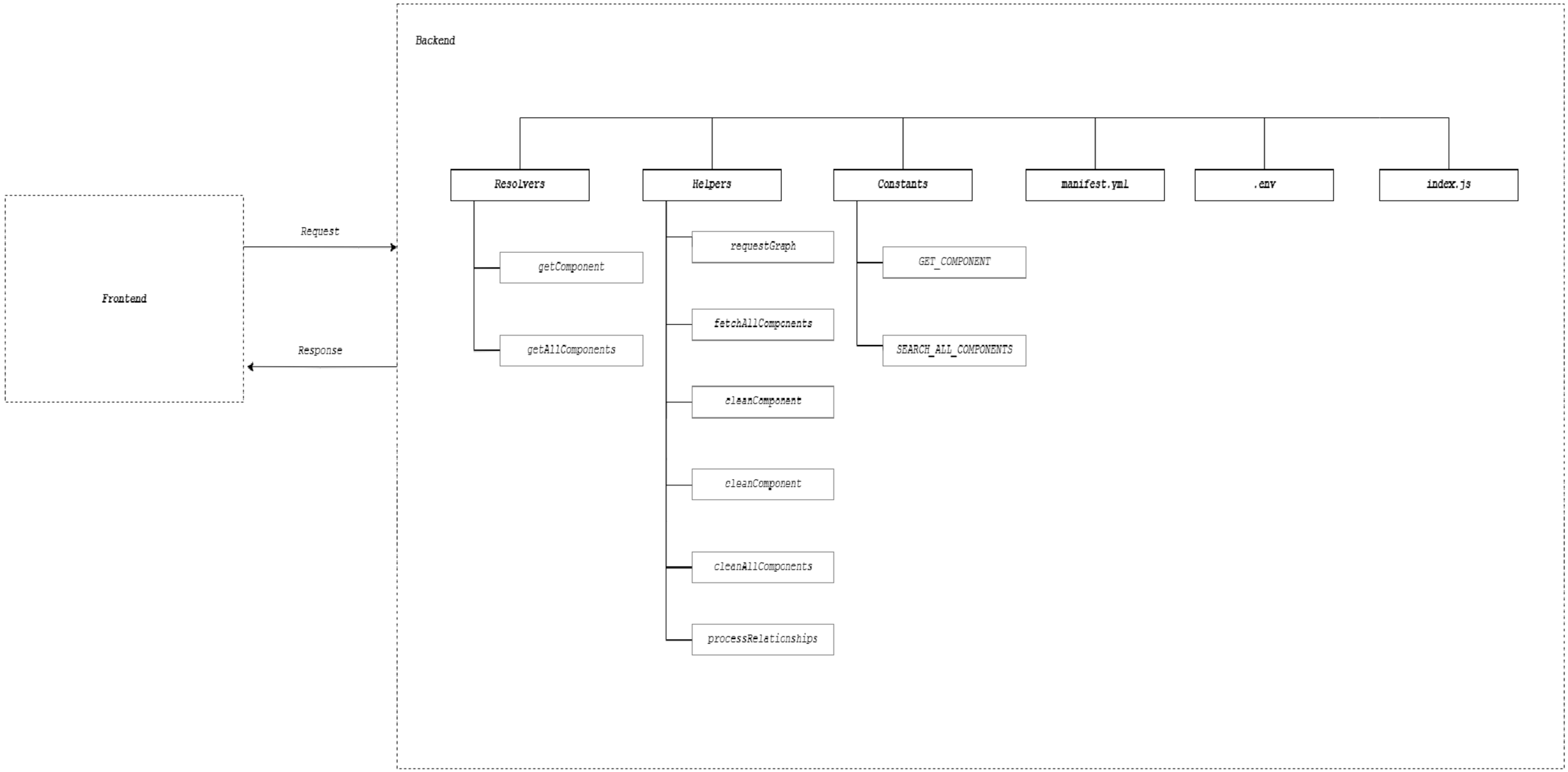
```
export class GraphQueries {
  static GET_COMPONENT = `
  query GetComponentWithDeepRelationships($componentId: ID!) {
    compass {
      component(id: $componentId) {
        ... on CompassComponent {
          id
          name
          relationships(query: { direction: INWARD}) {
            ... on CompassRelationshipConnection {
              nodes {
                startNode {
                  id
                  name
                  typeId
                  typeMetadata {
                    iconUrl
                  }
                  relationships(query: { direction: INWARD}) {
                    ... on CompassRelationshipConnection {
                      nodes {
                        startNode {
                          id
                          name
                          typeId
                          typeMetadata {
                            iconUrl
                          }
                          relationships(query: { direction: INWARD}) {
                            ... on CompassRelationshipConnection {
                              nodes {
                                startNode {
```

```
                                id
                                name
                                typeId
                                typeMetadata {
                                  iconUrl
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
          typeId
          typeMetadata {
            iconUrl
          }
        }
        ... on QueryError {
          message
        }
      }
    }
  }
`;

  static SEARCH_ALL_COMPONENTS = `
      query     SearchCompassComponents($cloudId:     String!,     $query:
CompassSearchComponentQuery) {
```

```
compass {
  searchComponents(cloudId: $cloudId, query: $query) {
    __typename
    ... on CompassSearchComponentConnection {
      nodes {
        component {
          id
          name
          typeId
          typeMetadata {
            iconUrl
          }
          relationships(query: { first: 50, direction: INWARD }) {
            ... on CompassRelationshipConnection {
              nodes {
                startNode {
                  id
                  name
                  typeId
                  typeMetadata {
                    iconUrl
                  }
                }
              }
              pageInfo {
                hasNextPage
                endCursor
              }
            }
          }
        }
      }
      pageInfo {
        hasNextPage
```

```
            endCursor
          }
        }
      }
    }
  }
    `;
}

export const GET_COMPONENT = GraphQueries.GET_COMPONENT;
export const SEARCH_ALL_COMPONENTS = GraphQueries.SEARCH_ALL_COMPONENTS;
```

Схема структурна додатку



Зм.	Арк.	№ докум.	Підп.	Дата	2026.КВР.122.423.06.00.00 СС			
Розроб.		Когут А. І.			Розробка серверної частини інтеграційного додатка для платформи Atlassian Compass Схема структурна додатку	Літ.	Маса	Масштаб
Перев.		Лісовий В. М.						
Т.контр.						Арк.	Аркушів 4	
Рецензент						ВСП ТФК ТНТУ КН-423 м. Тернопіль		
Н.контр.		Приймак В.А.						
Затв.								

Копіював

Формат А1

Перш. застос.

Справ. №

Інв. № подл.

Підп. і дата

Взам. інв. №

Інв. № дубл.

Підп. і дата

А4 Горіз.

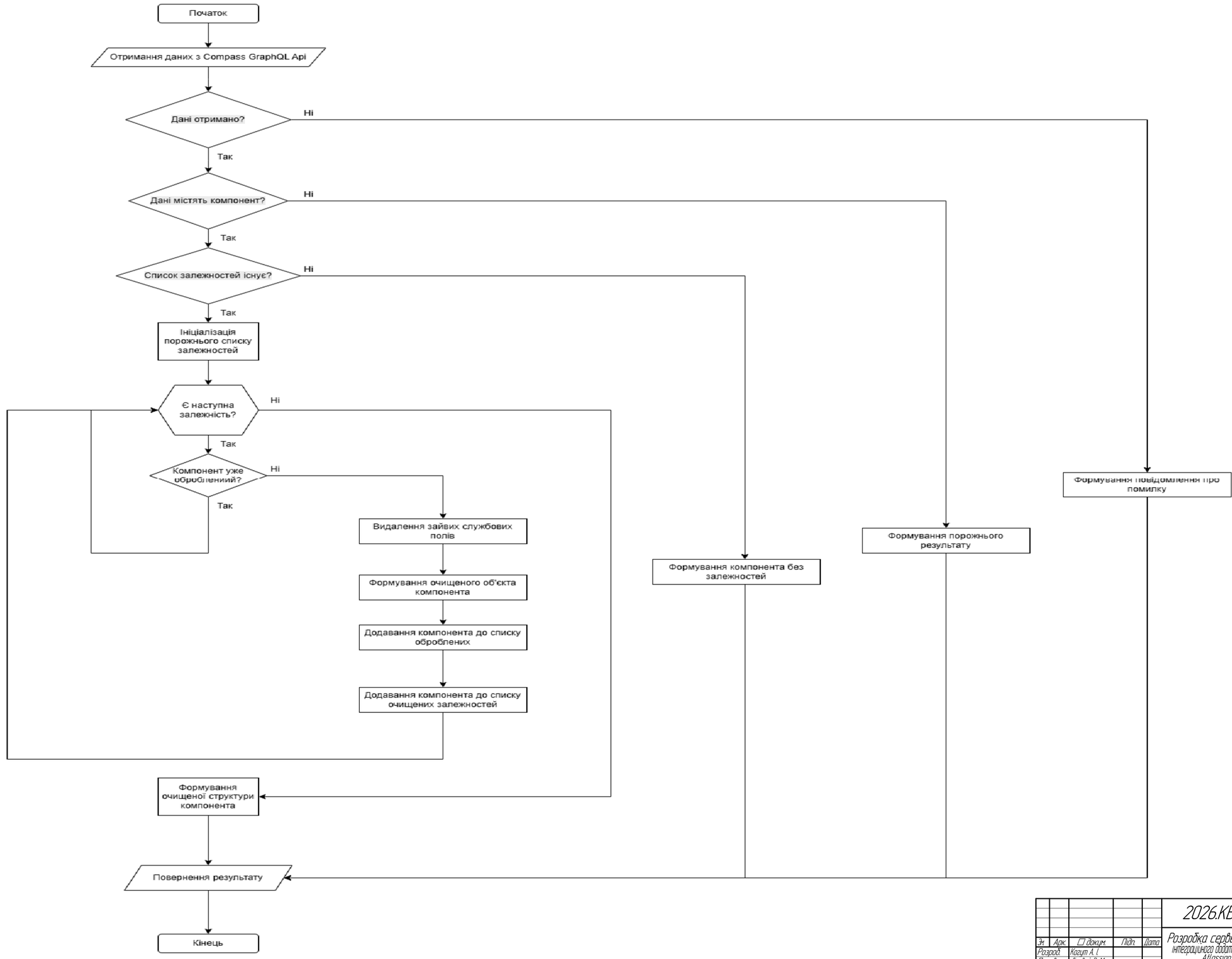
Таблиця техніко-економічних показників

№	Назва показника	Значення	№	Назва показника	Одиниці вимірювання	Значення
1	Платформа розробки	Atlassian Forge	6	Трудомісткість розробки	год	170
2	Мова написання	JavaScript (ES6+)	7	Собівартість	грн	77 364
3	Менеджер пакетів	npm	8	Плановий прибуток	грн	43 323
4	Api для отримання даних	Compass Graph QL Api	9	ЧТВ	грн	30 104
5	Загальний об'єм проекту без модулів	~120 КБ*	10	Термін окупності	рік	2,1

Лістинг файлу component-page.js

```
import Resolver from '@forge/resolver';
import { ComponentCleaner } from '../helpers/resolverHelpers.js';
import { GraphQueries } from '../constants/queries.js';
import { GraphApiClient } from '../helpers/graphApi.js';

class ComponentPageResolver {
    constructor() {
        this.resolver = new Resolver();
        this.cleaner = new ComponentCleaner();
        this.graphApi = new GraphApiClient();

        this.registerHandlers();
    }

    registerHandlers() {
        this.resolver.define('getComponent', this.getComponent.bind(this));
        this.resolver.define('getAllComponents', this.getAllComponents.bind(this));
    }

    async getComponent({ context }) {
        const componentId = context.extension.componentId;

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

        return this.cleaner.cleanComponent(componentResult);
    }

    async getAllComponents({ context }) {
        const { cloudId } = context;

        const allComponents = await this.graphApi.fetchAllComponents(GraphQueries.SEARCH_ALL_COMPONENTS, cloudId);

        return this.cleaner.cleanAllComponents(allComponents);
    }

    getDefinitions() {
        return this.resolver.getDefinitions();
    }
}

const componentPageResolver = new ComponentPageResolver();

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

Блок-схема алгоритму очищення даних



Зм.	Арк.	☐ докум.	Підп.	Дата	2026.КВР.122.423.06.00.00 БС	
Розроб.		Кагут А. І.			Розробка серверної частини інтеграційного додатка для платформи Atlassian Compass. Блок-схема	Літ. / Маса / Масштаб
Перев.		Лісовий В. М.				Арк. / Аркушів 4
Т.контр.						ВСП ТФК ТНТУ КН-423 м. Тернопіль
Рецензент						
Н.контр.		Приймак В.А.				
Затв.						

Копіював

Формат А1

Перш. застос. | Строй ☐ | Підп. і дата | Інв. № дубл. | Взам. інв. № ☐ | Підп. і дата | Інв. № орига.