НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

До захисту допущено:

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»

спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Використання Drupal REST API для інтеграції з системою управління проектами Redmine»

Виконала:
студентка IV курсу, групи ТЗ-11
Пацук Вікторія Олексіївна _____

Керівник:
Ст. викладач кафедри ТК НН ІТС, к.ф.-м.н.
Руренко Олександр Григорович _____

Рецензент:
Доцент кафедри ІТТ НН ІТС, к.т.н., доцент
Правило Валерій Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 Телекомунікації та радіотехніка

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2025 р.

ЗАВДАННЯ

на дипломну роботу студентці

Пацук Вікторії Олексіївни

1. Тема роботи «Використання Drupal REST API для інтеграції з системою управління проектами Redmine», керівник роботи Руренко Олександр Григорович, к.ф.-м.н., затверджені наказом по університету від «26» травня 2025 р. № 1755-с.

2. Термін подання студенткою роботи 9 червня 2025 р.

3. Вихідні дані до роботи: Документація Redmine REST API та Drupal 10 CMS, технічні специфікації протоколів JSON/HTTP для обміну даними, бібліотека DHTMLX Gantt для візуалізації діаграм, матеріали з Docker контейнеризації та конфігурації LAMP стеку.

4. Зміст роботи: Дослідження та розробка інтеграційного рішення між CMS Drupal та системою управління проектами Redmine з використанням REST API. Робота передбачає аналіз функціональних можливостей обох систем, створення модуля для автоматизованого імпорту даних з Redmine у Drupal, реалізацію механізмів візуалізації завдань у вигляді діаграм Ганта та оцінку

ефективності розробленого рішення для телекомунікаційної системи малого підприємства.

5. Перелік ілюстративного матеріалу:

1) Тема, актуальність, мета та завдання дипломної роботи;
2) Короткий огляд систем Redmine та Drupal;
3) Дослідження особливостей інтеграції REST API обох систем;
4) Проектування архітектури інтеграційного рішення;
5) Розробка модуля «Gant Drupal Module»;
6) Ключові методи, що забезпечують інтеграцію;
7) Оцінка ефективності розробленого рішення;
8) Перелік публікацій;
9) Висновки.

6. Дата видачі завдання 15 жовтня 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Визначення рекомендованої літератури	15.10.2024-05.11.2024	Виконано
2	Постановка завдань для виконання	06.11.2024-20.11.2024	Виконано
3	Дослідження особливостей інтеграції REST API Drupal та Redmine, розробка модуля	21.11.2024-20.05.2025	Виконано
4	Підготовка матеріалів до друку та оформлення пояснювальної записки	21.05.2025-31.05.2025	Виконано
5	Підготовка та оформлення презентації для доповіді	01.06.2025-08.06.2025	Виконано

Студентка Вікторія ПАЦУК

Керівник Олександр РУРЕНКО

РЕФЕРАТ

Текстова частина бакалаврської дипломної роботи містить: 109 сторінок, 34 рисунки, 9 таблиць, 20 джерел та 2 додатки.

Метою роботи є розробка і дослідження інтеграційного підходу до використання REST API для з'єднання двох програмних систем: CMS Drupal і системи управління проектами Redmine. У даній роботі пропонується новий підхід до створення телекомунікаційної інформаційної системи малого підприємства шляхом інтеграції системи управління проектами Redmine і Drupal сайту та проектування спеціального модуля Drupal, який в роботі позначений як «Gant Drupal Module». Зокрема, розглядається, використання REST API для інтеграції CMS Drupal із системою управління проектами Redmine підвищує ефективність управління проектами завдяки автоматизованій синхронізації даних та інтерактивній візуалізації завдань.

Розглянуто концепції та технології інтеграції систем управління проектами з веб-орієнтованими інформаційними системами. Проведено порівняльний аналіз методів інтеграції (REST API, SOAP, GraphQL) та обґрунтовано вибір REST API як оптимального рішення. Досліджено архітектуру та можливості REST API систем Drupal та Redmine. Описано архітектуру та основні компоненти розробленого інтеграційного рішення, яке включає модуль для Drupal з контролерами, формами налаштувань, шаблонами Twig та JavaScript бібліотекою dhtmlxGantt для візуалізації діаграм Ганта.

Розроблений модуль може знайти застосування в малих та середніх підприємствах, які використовують Drupal для веб-присутності та Redmine для управління проектами, забезпечуючи інтерактивну візуалізацію проектних даних на корпоративному сайті.

Ключові слова: REST API, Drupal, Redmine, інтеграція систем, управління проектами, діаграма Ганта, веб-розробка, модульна архітектура.

ABSTRACT

The text part of the bachelor's thesis contains: 107 pages, 34 figures, 9 tables, 20 sources and 2 appendixes.

The aim of the work is to develop and research an integration approach for using REST API to connect two software systems: Drupal CMS and Redmine project management system. This work proposes a new approach to creating a telecommunication information system for small enterprises through the integration of Redmine project management system and Drupal website and designing a special Drupal module, which is designated in the work as "Gant Drupal Module". In particular, it is considered that using REST API for integrating Drupal CMS with Redmine project management system increases project management efficiency through automated data synchronization and interactive task visualization.

The concepts and technologies of integrating project management systems with web-oriented information systems are examined. A comparative analysis of integration methods (REST API, SOAP, GraphQL) was conducted and the choice of REST API as an optimal solution was justified. The architecture and capabilities of REST API systems for Drupal and Redmine were studied. The architecture and main components of the developed integration solution are described, which includes a Drupal module with controllers, configuration forms, Twig templates and dhtmlxGantt JavaScript library for Gantt chart visualization.

The developed module can be applied in small and medium enterprises that use Drupal for web presence and Redmine for project management, providing interactive visualization of project data on the corporate website.

Keywords: REST API, Drupal, Redmine, system integration, project management, Gantt chart, web development, modular architecture.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	Application Programming Interface (інтерфейс програмування додатків)
CRUD	Create, Read, Update, Delete (операції для роботи з даними)
DHTMLX	Dynamic HTML eXtensions (бібліотека для створення інтерактивних діаграм)
HTTP	Hypertext Transfer Protocol (протокол передачі даних для веб-запитів)
JSON	JavaScript Object Notation (формат обміну даними)
LAMP	Linux, Apache, MySQL/MariaDB, PHP (технологічний стек для розгортання серверної частини)
MariaDB	Власна назва СУБД, розширення MySQL
OAuth	Open Authorization (механізм аутентифікації і авторизації)
PHP	Hypertext Preprocessor (мова програмування для веб-розробки)
PMI	Project Management Institute (Інститут управління проектами)
PMS	Project Management System (система управління проектами)
REST API	Representational State Transfer Application Programming Interface (інтерфейс для передачі даних між системами)
SOAP	Simple Object Access Protocol (протокол обміну структурованими повідомленнями)
SQL	Structured Query Language (мова для роботи з базами даних)
SSL	Secure Sockets Layer (протокол захищеного з’єднання)
SSO	Single Sign-On (єдина система входу)
UI	User Interface (користувацький інтерфейс)
URL	Uniform Resource Locator (уніфікований покажчик ресурсу)
XML	eXtensible Markup Language (розширювана мова розмітки)
ROI	фінансовий показник, який вимірює ефективність інвестицій шляхом порівняння отриманої вигоди (прибутку) з витраченими коштами.

ВСТУП

Розвиток комп’ютерів та інформаційних технологій спричинив низку радикальних змін у теорії та практиці проектного менеджменту, які наразі складають базову інструментальну основу для оптимізації операційної діяльності підприємств, які орієнтовані на проектну діяльність. Дана робота має прикладний характер і розширює функціональні можливості системи управління проектами Redmine через інтеграцію з сайтом на основі Drupal. Існуючі вже готові рішення або вимагають значних фінансових витрат на ліцензування комерційних продуктів, або мають обмежену функціональність і не можуть бути легко адаптовані під конкретні потреби малого бізнесу.

Мета роботи: Розробка і дослідження інтеграційного підходу до використання REST API для з'єднання двох програмних систем: CMS Drupal і системи управління проектами Redmine. У даній роботі пропонується новий підхід до створення телекомунікаційної інформаційної системи малого підприємства шляхом інтеграції системи управління проектами Redmine і Drupal сайту та проектування спеціального модуля Drupal, який в роботі позначений як «Gant Drupal Module». Зокрема, розглядається, використання REST API для інтеграції CMS Drupal із системою управління проектами Redmine підвищує ефективність управління проектами завдяки автоматизованій синхронізації даних та інтерактивній візуалізації завдань.

Об'єктом дослідження є методи інтеграції систем управління проектами з веб-орієнтованими інформаційними системами.

Предметом дослідження є технології інтеграції Drupal CMS та системи управління проектами Redmine з використанням REST API.

Завдання роботи:

1. **Дослідження особливостей інтеграції**:
    - Аналіз можливостей використання REST API Redmine для отримання та передачі даних про завдання, статуси, дедлайни.

- Дослідження принципів роботи Drupal REST API для взаємодії між платформами.

2. **Проектування архітектури інтеграції**:
   - Розробка модульної структури для Drupal, яка дозволяє обробляти дані з Redmine та виводити їх на сайт.
   - Створення механізмів автоматизованого імпорту завдань Redmine у Drupal і їх візуалізації у вигляді інтерактивних діаграм Ганта.
   - Використання HTTP-протоколів для здійснення CRUD-операцій (створення, читання, оновлення, видалення даних) для передачі даних між платформами.
3. **Розроблення модуля «Gant Drupal Module»**:
   - Створення базових компонентів модуля (файли .info.yml, .module, роутинг, контролери).
   - Реалізація функцій з'єднання з Redmine API та обробки отриманих даних.
   - Інтеграція бібліотеки **DHTMLX Gantt** для візуалізації завдань проекту у вигляді діаграм Ганта.
4. **Тестування інтеграційного рішення**:
   - Перевірка отримання даних через API.
   - Вивід діаграм Ганта на сайті Drupal із відображенням станів завдань.

РОЗДІЛ 1

1.1 Основні відомості про сучасні системи управління проектами

1.1.1 Історія та походження систем управління проектами

Задля кращого розуміння сучасних інструментів управління проектами корисно коротко розглянути їх походження. Подібно до багатьох передових технологій, початок управління проектами можна відслідкувати у промислових застосуваннях, що не сильно відрізняються від сучасних методологій управління проектами, які сьогодні поширені[8]. Першим інструментом візуалізації проекту, який був схожий на сучасні системи планування, була гармонограма, також відома як Гармонічний графік, розроблена польським інженером Каролем Адамецьким у 1896 році.

Одним з найвідоміших ранніх інструментів управління проектами, що використовується і донині, є діаграма Ганта. Розроблена Генрі Гантом на початку 20-го століття та представлена між 1910 і 1915 роками, дана діаграма забезпечила простий візуальний огляд дедлайнів проекту та стала революційною завдяки цій своїй здатності [11]. Подібно до багатьох передових технологій, початок використання діаграм Ганта можна відстежувати у військових та промислових застосуваннях, що продемонструвало її ефективність в організації складних виробничих процесів.

З розвитком обчислювальних технологій ручні методи почали переходити в цифрові формати. Проектний менеджмент у його сучасній інтерпретації почав приживатися лише кілька десятиліть тому. Починаючи з початку 1960-х років, бізнеси та інші організації почали бачити переваги організації роботи довколо проектів. Цей проектно-центричний погляд на організацію розвивався далі, як компанії почали розуміти критичну необхідність для їхніх співробітників у ефективних спілкуванні та співпраці,

одночасно інтегруючи свою роботу між кількома відділами та професіями , а в деяких випадках цілими галузями [8].

Перще програмне забезпечення для управління проектами Artemis з’явилося у 1970-ті роки розроблене компаніїю Metier Management Systems. Стрімкий розвиток інтернет-технологій у 1990-х роках стимулював поширення використання веб-додатків бізнесами та сприяло народження програмного забезпечення для управління онлайн-проектами. Починаючи з 2000-х методології Agile та поява хмарного ПЗ запустили безповоротні трансформаційні процеси у сфері управління проектами.

1.1.2 Огляд методологій управління проектами

Теорія та практика управління проектами пропонують та застосовують низку методологій, з більшим чи меншим успіхом. Ці методології в основному пропонуються національними або міжнародними асоціаціями управління проектами, а також певними організаціями та установами. Деякі з найвідоміших методологій є наступними:

1. Методологія PMI
2. Методологія IPMA
3. Методологія PRINCE2
4. Методологія YUPMA
5. Методологія APM
6. Методологія HBS
7. Гнучкі Agile методології (Scrum, Kanban та інші).

Окрім вищезазначених, існує велика кількість інших методологій, розроблених різними організаціями та установами та впроваджених для їх реалізації, наприклад, методологія Японської асоціації управління проектами, методологія Сіднейського університету [11].

Усі вищеописані методології, за винятком методології IPMA, є процесно-орієнтованими методологіями, тобто методологіями, що включають

певні процеси або фази управління певним проектом. До цієї групи належать усі згадані методології (PMI, APM, HBS, YUPMA, PRINCE2). За винятком методології IPMA, вони відрізняються лише способом та обсягом визначення та охоплення певних процесів або фаз. Методологія IPMA не є класичною методологією. Вона базується на посиленні певних компетенцій, якими повинен володіти будь-який керівник проекту для ефективного управління проектом.

Традиційні методології були розроблені у великих військових інвестиційних проектах, тому вони характеризуються безперервним, фазовим підходом, що підходить для такого типу проектів. Вони мають точну та сувору процедуру виконання, що базується на визначеній структурі проекту замовника та представленні в проектній документації, а також на точному технологічному проекту виконання [11].

У відповідь на це були розроблені гнучкі (*англ. agile*) підходи до очевидних відмінностей між певними видами проектів, особливо до специфіки IT-проектів. Agile методології були розроблені групою експертів з метою вдосконалення процесу управління IT-проектами, особливо у випадку проектів розробки програмного забезпечення. Специфічні принципи гнучкої методології описано в маніфесті agile (Fowler & Highsmith, 2001) та в Декларації взаємозалежності (Декларація взаємозалежності, 2005). Ідея гнучкого підходу полягає у створенні більш гнучкий підхід до управління IT-проектами, завдяки інтерактивному підходу, створеного та реалізованого процесу по частинах, з можливістю легкого внесення змін та повторення певних фаз виконання проекту. Гнучкий підхід дозволяє аналізувати виконувану роботу після кожного етапу разом з клієнтом, а також вносити зміни та доповнення з метою задоволення побажань клієнта, що часто буває в проектах розробки програмного забезпечення, де часто відсутні точні специфікації і клієнт не може бути впевненим, якого результату він очікує від

завершеного проекту (Фернандес і Фернандес, 2008; Rasnacis & Berzisa, 2017; Сердега і Піто, 2015).

Одже гнучкі методології — це методології управління, що використовуються в проектах розробки програмного забезпечення та базуються за такими принципами: гнучкий графік роботи, постійний перегляд виконаної частини або завдання та внесення змін, активна роль клієнта, специфічний спосіб організації проектної команди, безперервність комунікація між усіма учасниками, регулярні зустрічі команди проекту та команди проекту з клієнтом.

Серед найпопулярніших гнучних методологій варто виділити Scrum та Kanban. У Kanban використовується конвеєрний підхід. На дошках Kanban відображаються окремі завдання – вид на дошку з колонками, потім завдання переміщуються на різні етапи до їх завершення. І в Kanban, і в Scrum є беклог пріоритетних завдань проекту – але замість того, щоб планувати роботу спринтами, команди канбан вибирають завдання з найвищим пріоритетом у беклозі. У таблиці 1.1 перераховано основні відмінності між традиційними та гнучкими методологіями.

Таблиця 1.1

Порівняння традиційного та гнучкого підходів до управління проектами

Традиційні методології	**Гнучкі методології**
Постійний процес	Послідовний процес
Чітка структура проекту	Змінювана структура проекту
Проектна документація є точною та детальною	Проектна документація є попередньою
Технологія виконання проекту чітко визначена та здебільшого незмінна	Технологія виконання проекту підлягає змінам
Можливі зміни	Постійні зміни
Рідкісні зустрічі з клієнтом	Регулярні зустрічі з клієнтом
Багатофункціональна команда	Самоорганізована команда
Високий рівень повноважень керівника проекту	Обмежений рівень повноважень керівника проекту

1.1.3 Порівняльний аналіз сучасних систем управління проектами

Agile-методологія є популярною на сучасному ринку розробки ПЗ через зменшену кількість формальної проектної документації. Компаніям доступні різнопрофільні інструменти, засновані на Agile-методології. Найбільш вдалими та універсальними рішеннями є Jira, Trello, та Redmine Практика використання демонструє, що ці технології не лише допомагають відстежувати прогрес проекту, але й покращують комунікацію в команді. Розглянемо детальніще ці три системи для управління проектами. (табл. 1.2)

Trello та Jira належать Atlassian, і обидва натхненні та створені для впровадження філософії Agile-менеджменту проектів. Trello — це гнучка дошка Kanban, яку можна використовувати для відстеження будь-якого проекту, великого чи малого [10]. Jira — це інструмент управління проектами для команд розробників програмного забезпечення. Він має дошки Kanban, а також дошки Scrum та інші робочі процеси Agile, орієнтовані на розробку програмного забезпечення [9]. Trello швидкий і простий, із мінімальною структурою: ідеї та завдання створюються як картки у простому візуальному форматі. Jira пропонує дошки Kanban із більшою структурою, налаштовуваними полями та акцентом на формальний підхід і стандартизацію для командних потреб. Фреймворк Scrum у Jira забезпечує більш структурований спосіб спільної роботи команд розробників, чого важко досягти в Trello [12].

Redmine є системою управління проектами з відкритим кодом, розробленою на фреймворку Ruby on Rails. На відміну від спеціалізованих Agile-інструментів, Redmine пропонує універсальний підхід до управління проектами, поєднуючи функції трекера завдань, Wiki-системи та інструментів звітності. Система підтримує одночасне ведення кількох проектів та надає потужні можливості налаштування через систему плагінів і REST API [13].

Таблиця 1.2

Порівняльна характеристика інструментів управління проектами

Характеристика	Jira	Trello	Redmine
Базова технологія	Java/Atlassian Stack	React/Node.js	Ruby on Rails
Тип ліцензії	Комерційна	Freemium	Open Source
Основна методологія	Scrum/Kanban	Kanban	Waterfall/Hybrid
API підтримка	REST API v3	REST API	REST API + XML
База даних	PostgreSQL/ MySQL	Хмарна	MySQL/PostgreSQL/ SQLite
Складність налаштування	Висока	Низька	Середня
Цільова аудиторія	Великі IT-команди	Малі команди/ стартапи	Корпоративні клієнти
Інтеграційні можливості	3000+ додатків	Обмежені	Плагіни + API
Мобільна підтримка	Нативні додатки	Веб + мобільні додатки	Веб-інтерфейс

1.1.4 Система управління проектами Redmine: архітектура та можливості

Однією з провідних систем, що виникла на початку XXI століття, є Redmine. Це програмне забезпечення є проектом з відкритим вихідним кодом, перша версія якого була випущена Жан-Філіпом Ленгом у 2006 році. Redmine, побудована на фреймворку Ruby on Rails, підтримує широкий спектр функцій: ролі учасників, управління дозволами, діаграми Ганта, календарі, управління версіями та документами. Завдяки гнучкості та розширюваності Redmine стала флагманським рішенням для управління проектами у світі програмного забезпечення з відкритим кодом. За понад півтора десятиліття існування система знайшла застосування від стартапів до агентств, компаній зі списку Fortune 500 і навіть провідних університетів, таких як Оксфорд [13].

Redmine успішно поєднує функції системи управління проектами та трекера завдань. Ключовою особливістю системи, є повністю налаштовуваний робочий процес, який дозволяє визначати дозволи на зміну статусу завдання для кожної ролі та типу трекера. Як демонструє схема взаємодії (див. рис. 1.1), система забезпечує ефективну комунікацію між керівником проекту та розробниками через централізовану платформу. Redmine надає комплексні можливості для управління та звітності, дозволяючи керівництву оперативно отримувати детальні звіти про стан проектів та активність команд. Система підтримує автоматичну генерацію різноманітних аналітичних матеріалів, включаючи діаграми Ганта, табличні звіти у форматі PDF та інші документи через вбудовані інструменти звітності.

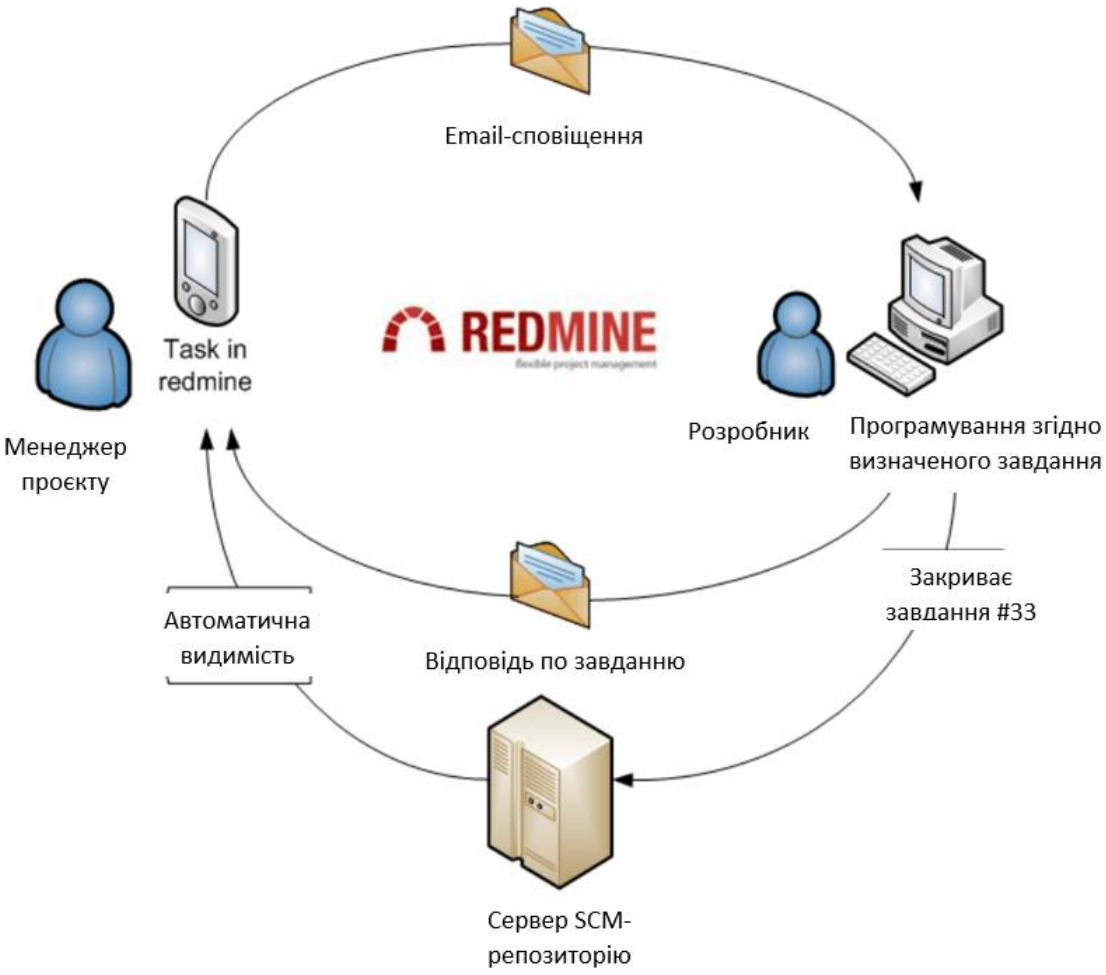

Рис. 1.1 Схема взаємодії учасників проекту в системі Redmine

Redmine також підтримує такі важливі функції, як пріоритети, підзавдання, спостереження, коментарі, власні поля, фільтри списків тощо. Враховуючи важливість документації в командній роботі, багато команд створюють веб-сайти Wiki. Redmine постачається з власною системою Wiki для кожного проекту, що підтримує синтаксис розмітки Textile та підсвічування синтаксису вихідного коду. Важлвио відзначити, що цей синтаксис Wiki підтримується скрізь у Redmine — в описах завдань, коментарях, новинах тощо. Цей синтаксис також дозволяє створювати перехресні посилання на інші завдання та проекти.

Для підтримки публічних проектів Redmine має фунуціональний модуль форуму, що дозволяє створювати необхідну кількість форумів. Кожен форум може мати довільну кількість тем.

Технологічною основою Redmine є мова програмування Ruby та веб-фреймворк Rails. Ruby є сучасною метапрограмною об'єктно-орієнтованою мовою. Ця мова програмування дуже гнучка і призначена для створення потужних додатків з високою швидкістю розробки. Дані характеристики повною мірою притаманні як Redmine, так і Ruby-on-Rails [16].

Rails - це веб-фреймворк, подібний до Symfony та Zend Framework, але на відміну від них, це де-факто стандарт для Ruby. Ruby підтримує метапрограмування. Це техніка, яка дозволяє програмі змінювати свій код під час виконання. Кожну частину функціоналу Redmine можна змінити програмно. Зазвичай API програми обмежується певною функціональністю, але в Ruby немає обмежень завдяки метапрограмуванню. Це створює потужні можливості для розробки плагінів через API.

Ruby-on-Rails надає плагін API, який використовується для розробки плагінів Rails, що називаються engines. Тому, Redmine насправді не потрібно надавати API для плагінів, щоб бути розширюваним, але система це забезпечує. API плагінів Redmine створений на основі API рушія Rails.

Система плагінів є ключовим фактором популярності Redmine. Якщо розробник знайомий з Ruby та Ruby-on-Rails, освоєння процесу створення плагінів для Redmine не становить значної складності. Враховуючи, що Ruby-on-Rails популярний в сучасній веб-розробці, Redmine має величезну кількість потенційних розробників. Тому існує велика різноманітність плагінів. З цими плагінами можна перетворити Redmine на CRM або службу підтримки.

Попри широким функціоналом існуючих плагінів для Redmine залишаються потреби, які вони не здатні задовільнити. Підприємствам часто потрібні зовнішні інструменти, але без зміни базового середовища роботи. Одним із способів розширення функціоналу Redmine є інтеграція з іншими платформами через REST API.

Серед численних інструментів для управління проектами, деякі з яких використовують подібні функції, Redmine вирізняється завдяки відкритому вихідному коду. Дана особливість надає користувачам можливість аналізувати та модифікувати компоненти коду, що сприяє розвитку проекту. Така модль забезпечує рівень різноматнітності та інноваційності, характерний для спільноти програмного забезпечення з відкритим кодом. Архітектура системи Redmine представлена на рис. 1.2.

Рис. 1.2 Архітектура системи Redmine

Важливою перевагою Redmine є кросплатформна сумісність та універсальна доступність. Система забезпечує стабільну роботу на різних операційних системах та пристроях, усуваючи проблеми сумісності. Користувачі можуть отримати повноцінний доступ до функціоналу системи з будь-якого пристрою з підключенням до Інтернету, незалежно від типу операційної системи.

1.2 Основні відомості про системи управління контентом

1.2.1 Концепція та еволюція систем управління контентом

Сьогодні одним з основних інструментів комунікації є контент. Формати контенту охоплюють широкий діапазон інформаційних структур, зокрема текстові матеріали, зображення, відеоконтент та ін. Зростання обсягів контенту зумовило необхідність розробки спеціалізованих методів до його систематизації та координації. Відповідно, виникла потреба створення систем управління контентом, що забезпечили б ефективне адміністрування інформаційних ресурсів у формі веб-орієнтованих інформаційних платформ.

Інтернет-середовище зазнало фундаментальних трансформаційних процесів протягом свого розвитку. Початковий етап характеризувався появою Web 1.0, архітектура якого переважно базувалася на статичних веб-ресурсах з обмеженою функціональністю. Дані веб-ресурси забезпечували виключно односпрямовану комунікаційну модель та були реалізовані на основі трьох фундаментальних веб-технологічних стандартів, а саме мови розмітки гіпертексту (HTML), протоколу передачі гіпертекстових даних (HTTP) та системи уніфікованих локаторів ресурсів (URL).

Статична природа цих веб-ресурсів суттєво обмежувала користувачів, надаючи їм виключно можливість пасивного споживання представленої інформації без будь-яких форм інтерактивної взаємодії з контентом або платформою.

Потреба у інтерактивної взаємодії з користувачем зумовила перехід до концепції Web 2.0, що відома динамічними веб-ресурсами з інтегрованими

функціональними можливостями такими як читання та запис інформації. Даних перехід забезпечив двосторонню комунікаційну модель між власником веб-ресурсу та кінцевим користувачем.

Розповсюдження динамічних веб-платформ спонукало компанії до розробки власних веб-ресурсів з метою забезпечення присутності у віртуальному середовищі. Створення відповідних веб-платформ довго потребувало спеціалізованих знань скриптових мов та веб-розробки.

Відповідно, нестача кваліфікованих кадрів, особливо актуальна для малих підприємств, зумовила виникнення сучасних систем управління контентом (англ. CMS - Content Management System). Сучасні CMS не потребують спеціалізованих знань програмування та верстки для організації цифрового контенту [6].

1.2.2 Принципи роботи та архітектура CMS

Суть системи управління контентом полягає у зборі, управлінні та публікації контенту. Система керування розроблена, щоб максимально спростити керування веб-сайтом, зберігаючи при цьому гнучкість у налаштуваннях і контролі. Оскільки створенню систем управління контентом стало приділятися все більше уваги, з’явилися системи з відкритим кодом. Найбільш типовою перевагою систем керування вмістом з відкритим кодом є те, що велика група розробників щодня покращує ці системи, роблячи їх зручнішими у використанні [17].

На рис. 1.3 представлено три основні етапи CMS. Спочатку система створює вміст у світлі знання про те, що інтерфейс CMS є дуже зручним для користувача інтерфейсом, створюючи нові сторінки або оновлюючи існуючі без знання HTML. Після створення будь-якої сторінки вмістом можна легко керувати, оскільки він зберігається в репозиторії та оновлюється, коли контент змінюється.

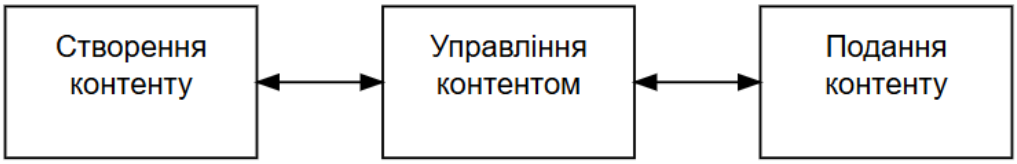

Рис. 1.3 Концептуальна модель функціонування

Як показано на рис. 1.4, інформація збирається та формується через CMS для публікації на веб-сторінках. Перші два компоненти є основними вимогами для користувача та бізнес-стратегії. Це початкові кроки для збору інформації, яка необхідна для публікації веб-сайту. Далі йде процес публікації. Архітектура інформації гарантує, що база є безпечною, а сайт буде найдіним і здатним до розширення.

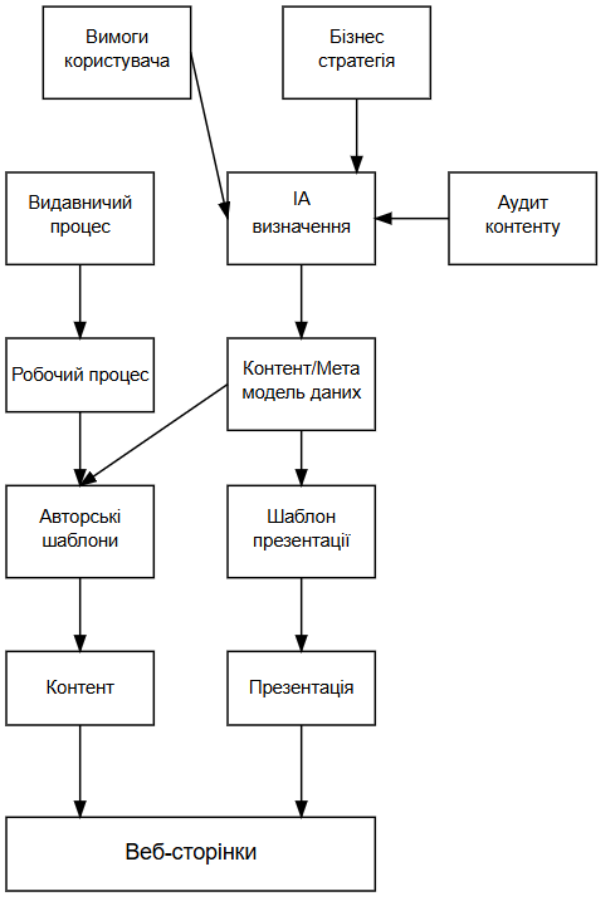

Рис. 1.4 Інформаційна архітектура CMS

Загальні багатошарові архітектурні компоненти та їх комунікації показані на рис. 1.5. Існує шість основних компонентів. Архітектура операційної системи знаходиться внизу. Над ним вказано тип веб-сервера та

бази даних. Потім йдуть мови сценаріїв, за якими йде фреймворк і плагіни, які додають більше функціональності веб-сайтам [5].

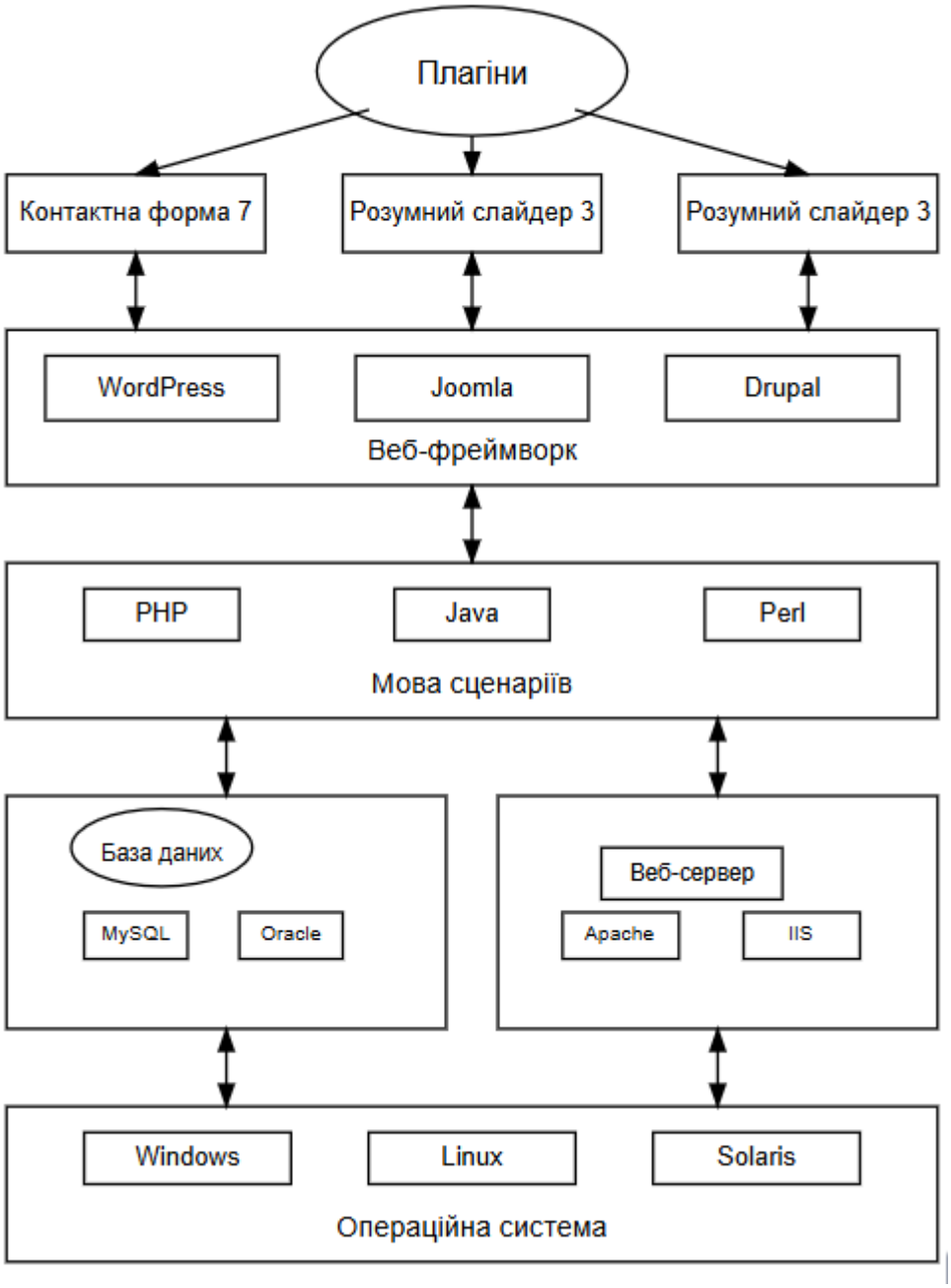

Рис. 1.5 Багаторівнева архітестура CMS

1.2.3. Порівняльна характеристика основних CMS платформ

Системи управління контентом, які широко використовуються для створення веб-сайтів: Joomla та Drupal.

Фреймворк Joomla здобув репутацію стати одним з найнадійніших CMS-додатків для створення привабливих веб-сайтів (рис. 1.6). Joomla побудована на програмних засобах з відкритим кодом. Модулі Joomla, компоненти та плагіни використовуються для розширення функціональності базового фреймворку Joomla. Модулі є розширеннями, що використовуються при відображенні сторінки. Вони можуть використовуватися для відображення даних з компонента і можуть стояти самостійно. Компоненти є найбільш інтерактивними серед них, і саме тому їх можна розглядати як "міні-додатки". Біти коду, що виконуються на певній події, називаються плагінами. Фреймворк Joomla може бути розширений за допомогою потужних плагінів. Крім того, Joomla надає відмінний та організований плагін-менеджер для встановлення плагінів. Шаблони визначають зовнішній вигляд веб-сайту.

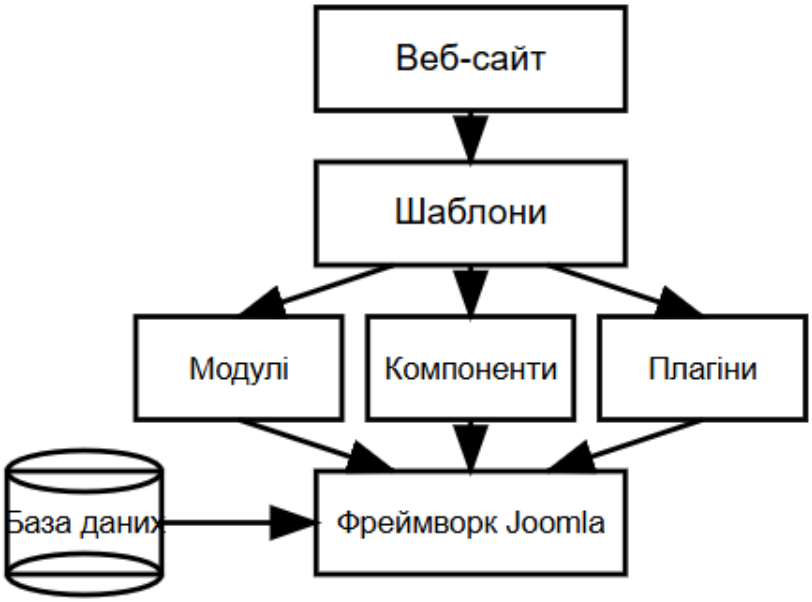

Рис. 1.6 Архітектура Joomla

З економічних причин інструменти з відкритим кодом завжди були кращим вибором для розробки різних веб-додатків. Велика кількість користувачів використовує Drupal для розробки веб-сайтів завдяки його численним функціям, таким як гнучкість контенту, легке адміністрування

користувачів та здатність обробляти складні робочі процеси. Рис. 1.7 є графічним представленням базової аріхтектури Drupal.

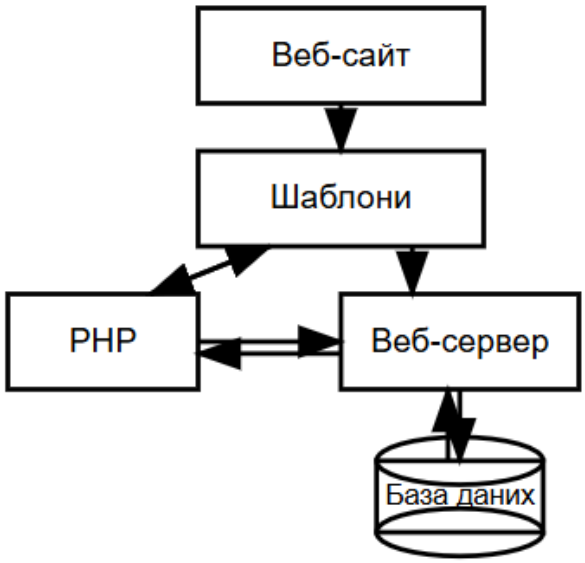

Рис. 1.7 Базова архітектура Drupal

Наступні кроки висвітлюють роботу Drupal CMS:

- Користувачі надсилають запити на сервер через Drupal CMS, де веб-браузер виступає як клієнт.
- Drupal CMS отримує запити від клієнта.
- Drupal використовує PHP для запуску користувацького коду.
- PHP надсилає користувацький код через веб-сервер за допомогою протоколу HTTP
- База даних зберігає інформацію користувача

У таблиці 1.3 показано порівняння двох CMS на основі параметрів, таких як безпека, плагіни, архітектура тощо. Згідно з таблицею, ми можемо зробити висновок, що Drupal демонструє переваги у багатьох ключових аспектах, включаючи безпеку, гнучкість архітектури та можливості розширення. Особливо важливою перевагою Drupal є його вища безпека та потужніші можливості обробки складного контенту. Крім того, можливості кібератак (таких як SQL-ін'єкція, DDoS) ефективніше запобігаються у Drupal завдяки вбудованим механізмам безпеки [15]. Ці переваги включають використання

надійних паролів, використання SSL-сертифікатів та обмеження логінів за допомогою root, розміщення обмеження на логіни тощо.

Таблиця 1.3

Порівняння CMS платформ на основі атрибутів

Атрибут	**Joomla**	**Drupal**
Безпека	Низька	Висока
Плагіни	34,000+	7,000+
Експорт демо-даних	Ні	Ні
Мобільний вигляд	Ні	Так
Веб-сайт аналітика	Так	Так
Безкоштовні теми	2,000+	1,000+
Складність	Середня	Середня
Налаштування віджетів	Ні	Ні
Налаштування тем	Ні	Так

1.2.4. CMS Drupal як платформа для веб-розробки

Drupal – одна з найбільш універсальних і широко використовуваних систем управління контентом (CMS), була створена з бачення розширення можливостей розробників для створення масштабованих і настроюваних веб-сайтів. Розвиток цієї системи розпочався у 2000 році, коли Дріс Буйтаерт, засновник і провідний розробник, створив Drupal як дошку оголошень для обміну ідеями з друзями. Усвідомлюючи потенціал розробленої системи, у 2001 році він випустив Drupal як проект з відкритим вихідним кодом, що ознаменувало початок трансформаційного шляху для веб-розробки. За роки свого існування Drupal Core перетворився на надійну CMS, поточною версією якої є Drupal 11. На основі цієї платформи функціонують веб-сайти підприємств, урядових установ та організацій по всьому світу. Його гнучкість, модульна архітектура та активна спільнота розробників зробили його кращим вибором для створення складного та масштабованого цифрового досвіду.

Комбінація операційної системи, на якій працює CMS, скриптинг мовою, на якій він написаний, базою даних, в якій він зберігає свою інформацію, та веб-сервер, який запускає скрипти для отримання інформації та повернення її до веб-браузер відвідувача сайту відомий як стек, на якому працює CMS. Часто використовувана комбінація, що відома як стек LAMP: операційна система Linux, веб-сервер Apache, база даних MySQL і скриптова мова PHP.

Drupal є гнучкою CMS на основі стека LAMP стека, модульна структура якої дозволяє за потреби додавати та видаляти модулі; дозволяючи змінювати весь зовнішній вигляд веб-сайту шляхом встановлення та видалення тем. Ядро Drupal, відоме як Drupal Core, містить PHP скрипти, необхідні для запуску базового функціоналу CMS, кілька необов'язкових модулі та теми, а також багато JavaScript, CSS та графічних ресурсів. Багато додаткових модулів та тем можна завантажити з офіційного сайту Drupal.org [15].

Drupal також може працювати на інших технологічних стеках:

- Операційною системою може бути Windows або Mac OS замість Linux.
- Веб-сервер може бути Nginx або IIS замість Apache.
- База даних може бути PostgreSQL або SQLite замість MySQL, або MySQL-сумісна заміна, така як MariaDB або Percona.

Інші операційні системи, веб-сервери та бази даних також можуть бути адаптовані для роботи з Drupal. Слід зазначити, що скрипти, які використовує програмне забезпечення, написані на PHP, тому мова програмування залишається незмінною.

1.2.5 Система управління контентом Drupal: архітектура

У цьому пункті розглянемо архітектурний стиль Drupal для реалізації користувацьких інтерфейсів. На рис. 1.8 показана архітектура Drupal з ролями користувачів:

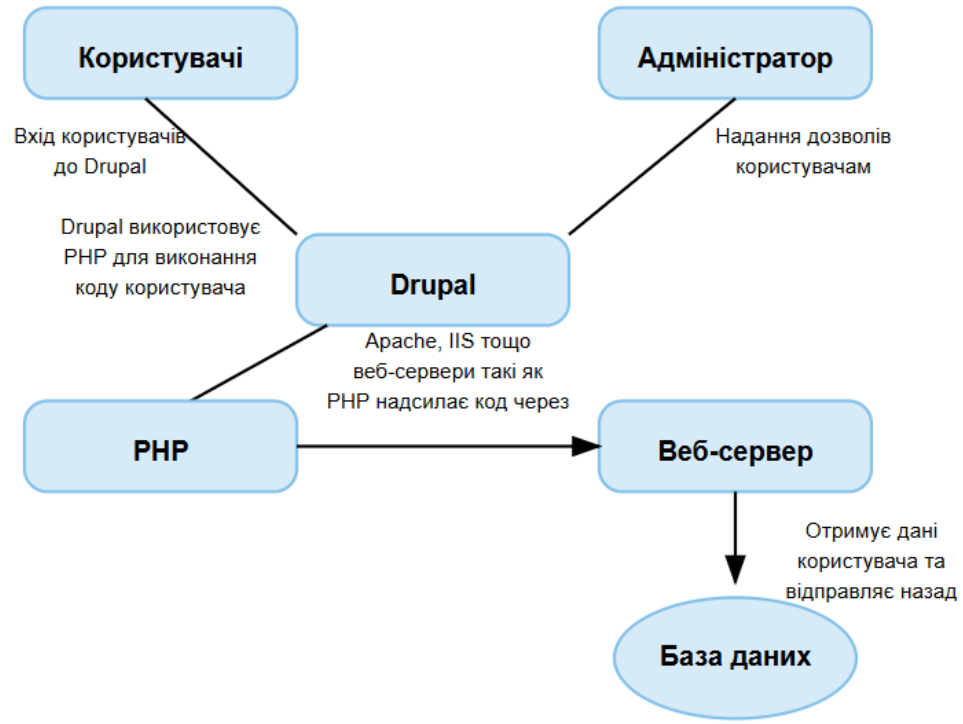

Рис. 1.8 Повна архітектурна схема Drupal з ролями користувачів

Архітектура Drupal містить наступні шари:

1. Користувачів;
2. Адміністратор;
3. Drupal ядро;
4. PHP;
5. Веб-сервер;
6. База даних.

Користувач формує запит до серверної частини засобами Drupal CMS, тоді як веб-браузери, пошукові роботи тощо виконують функції клієнтів. Системний адміністратор уповноважений надавати права доступу автентифікованим користувачам та припиняти несанкціоновані з'єднання. Адміністративний акаунт має всеосяжні права для керування змістом та налаштування платформи.

Веб-сервер є серверною інфраструктурою, яка підтримує користувацьку взаємодію та процес запитів через HTTP (Hyper Text Transfer Protocol), забезпечуючи передачу файлів для генерації веб-сторінок споживачам. Зв'язок між клієнтською та серверною частинами функціонує на основі HTTP протоколу. Існує можливість використання різних веб-серверних технологій, включаючи Apache, IIS, Nginx, Lighttpd та подібні платформи.

База даних представляє собою систему зберігання даних користувачів, контенту та додаткової інформації сайту. Вона слугує для акумулювання управлінських даних, потрібних для адміністрування Drupal-платформи. Drupal використовує базу даних для доступу до інформації та підтримує операції збереження, зміни та оновлення баз даних.

Висновки

Аналіз показав, що серед сучасних систем управління проектами та контентом найбільш широко застосовують наступні рішення: Redmine, Jira, Trello для управління проектами та Drupal, Joomla для управління контентом.

Кожна з систем має свої переваги та недоліки і використовується залежно від специфіки бізнес-потреб підприємства.

Для ефективної організації корпоративних процесів, автоматизації обміну даними та покращення візуалізації проектної інформації використовують сучасні методи інтеграції систем на основі REST API.

Інтеграційні рішення між системами управління проектами та CMS широко застосовують у малому та середньому бізнесі, IT-компаніях, консалтингових агентствах, освітніх установах та інших сферах, де потрібна синхронізація проектних даних з корпоративними веб-ресурсами.

РОЗДІЛ 2

МЕТОДИ ІНТЕГРАЦІЇ ПРОГРАМНИХ СИСТЕМ ТА ЇХ ЗАСТОСУВАННЯ

2.1 Аналіз сучасних підходів до інтеграції веб-систем. API

Використання різних IT-компонентів для різних завдань є звичайною практикою. Але в міру розширення бізнес-функцій компанії можуть зіткнутися з безліччю розрізнених інструментів, які не можуть обмінюватися даними та працювати разом. Саме тоді на допомогу приходить системна інтеграція.

Системна інтеграція (також відома як IT-інтеграція або програмна інтеграція) – це процес об'єднання програмних та апаратних модулів в одну цілісну інфраструктуру. Мета такої інтеграції полягає в досягненні безперебійного обміну інформацією та процесами між системами [2].

Сучасні додатки побудовані на моделі клієнт-сервер. Дана модель працює за принципом структурування завдань між сервером постачальником послуг та сервісом клієнтом. Клієнтський бік зазвичай представлений фронтенд-додатками, а сервери внутрішніми додатками.

Інтерфейси прикладного програмування (API) забезпечують передачу даних між системами у стандартизованому форматі. Системна інгеграція може бути реалізована за допомогою різних архітектурних моделей, вибір залежить від кількості і характеру компонентів, що потребують з’єднання [14].

Інтерфейс прикладного програмування потрібно чітко відрізняти від інтерфейсу користувача. Інтерфейс користувача приймає дані від користувачів, пересилає їх до API для обробки та повертає результати користувачеві. API не взаємодіє з користувачем, а обробляє дані, отримані від одного програмного модуля, та передає результати назад до іншого модуля. Що стосується того, як API взаємодіють, то найчастіше використовується протокол HTTP і обмін контентом відбувається у форматі JSON або XML [1]. Але цілком можливі інші протоколи та формати контенту.

Принцип роботи API зазвичай виражається через зв'язок запит-відповідь між клієнтом і сервером (рис. 2.1). Клієнт – це будь-який фронтенд-застосунок, з яким взаємодіє користувач. Сервер відповідає за логіку серверної частини та операції з базою даних. У цьому сценарії API працює як проміжний рівень між клієнтом і сервером, що дозволяє надсилати запити на дані та відповіді.

Рис. 2.1 Принцип роботи API

API іноді вважаються контрактами, де документація є угодою між сторонами: «Якщо сторона 1 надсилає віддалений запит, структурований певним чином, саме так відреагує програмне забезпечення сторони 2». Цей контракт має форму специфікації API або протоколу — архітектурного плану, який описує поведінку інтерфейсу та служить приблизним керівництвом щодо створення певного API (ми поговоримо про найпоширеніші специфікації API пізніше). Серед іншого, протокол визначає основні компоненти API.

2.2 Аналіз архітектурних підходів до API: REST, GraphQL, SOAP

Сучасна веб-розробка потребує грунтовного підходу до вибору технології API (див. рис. 2.2), адже кожна з них має власний архітектурний стиль та власні стандартизації обміну даними [3].

Командам розробників важливо розуміти відмінність між ними, щоб досягти максимальної ефективності власного проекту.

Рис. 2.2 Хронологія виникнення архітектур API

2.3 API SOAP

SOAP (з англ. Simple Object Access Protocol) – це протокол обміну повідомленнями, що працює шляхом кодування даних у форматі XML. Даний формат відомий надмірною формальністю, адже в поєднанні з масивною структурою повідомлень SOAP стає найбільшбагатослівним стилем API.

Вважається одним з найперших протоколів веб-сервісів. Представлений у 1999 році, метою розробки стало забезпечення незалежного від платформи способу обміну даними між різними системами через мережу Інтернет [20].

Повідомлення SOAP складаються з:

- тегу на початку та кінці;
- органу з запитом або відповіддю;
- заголовку, що визначає специфіки або додаткові вимоги;
- опису несправності, що можуть виникнути під час обробки запиту.

```xml
<!-- SOAP запит -->
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <GetUser>
            <UserId>123</UserId>
        </GetUser>
    </soap:Body>
</soap:Envelope>

<!-- SOAP відповідь -->
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <GetUserResponse>
            <User>
                <Id>123</Id>
                <Name>Іван Петренко</Name>
                <Email>ivan@example.com</Email>
            </User>
        </GetUserResponse>
    </soap:Body>
</soap:Envelope>
```

Рис. 2.3 Приклад повідомлення SOAP

Логіка архітектури SOAP написана на мові опису веб-сервісів WSDL, що визначає кінцеві точки та процеси. Завдяки цьому різні мови пракрамування та IDE здатні швидко налаштовувати зв’язок. Перевагою SOAP є гнучкі протоколи передачі, що здатні пристосовуватись до кількох сценаріїв.

Вагомою функцією є вбудована обробка помилок, повертається повідомлення з кодом помилки та поясненням. Не менш важливим фактором є ряд розширень безпеки (див. рис. 2.4). SOAP відповідає якості транзакцій корпоративного рівня, забезпечує цілісність всередині транзакції, при цьому здійснює шифрування на рівні повідомлення.

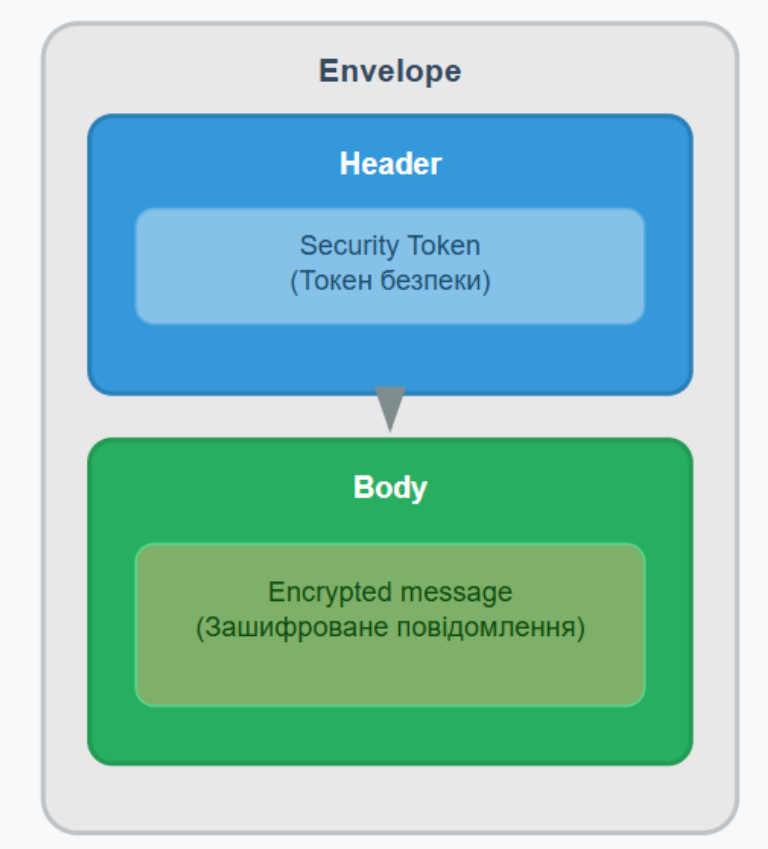

Рис. 2.4 Реалізація SOAP WS-SECURITY

На сьогодні архітектура SOAP використовується для внутрішньої інтеграції всередині підприємств. Жорстка структура архітектури, можливості безпеки та авторизації роблять SOAP найбільш вддалим варіантом для забезпечення виконання офіційних договорів про програмне забезпечення між API та клієнтом, дотримуючись при йьому юридичного договору. Тому фінансові організації, корпоративні користувачі зупинять свій вибір на SOAP.

2.3 REST API

REST був вперше представлений у 2000 році Роєм Філдінгом у його докторській дисертації «Архітектурні стилі та дизайн мережевих програмних архітектур». Філдінг, який також був одним з основних авторів протоколу HTTP, визначив REST як архітектурний стиль, який заснований на принципах вебу. API RESTful розроблені таким чином, щоб бути простими, масштабованими та гнучкими. Вони часто використовуються у веб- та

мобільних додатках, а також в архітектурах Інтернету речей (IoT) та мікросервісів.

Навідміну від жорстко визначеної архітектури SOAP, RESTful має відповідати архітектурним обмеженням:

- Уніфікований інтерфейс: визначає єдиний спосіб взаємодії з сервером.
- Стан Stateless: необхідний стан обробки запиту, що міститься в запиті без зберігання сервером інформації пов'язаної з сеансом.
- Кешування.
- Клієнт-серверна архітектура.
- Багаторівнева система використання.
- Клієнт має доступ до виконуваного коду від серверів.

З кожною відповіддю REST API надає посилання з інформацією про те, як використовувати API. Цей механізм дозволяє відмежовувати клієнта та сервер, що в результаті мають можливість розвиватися незалежно один від одного без перешкод в комунікації.

У REST архітектурі (див. рис. 2.5) операції з ресурсами виконуються через стандартні HTTP методи, такі як GET, POST, PUT, DELETE.

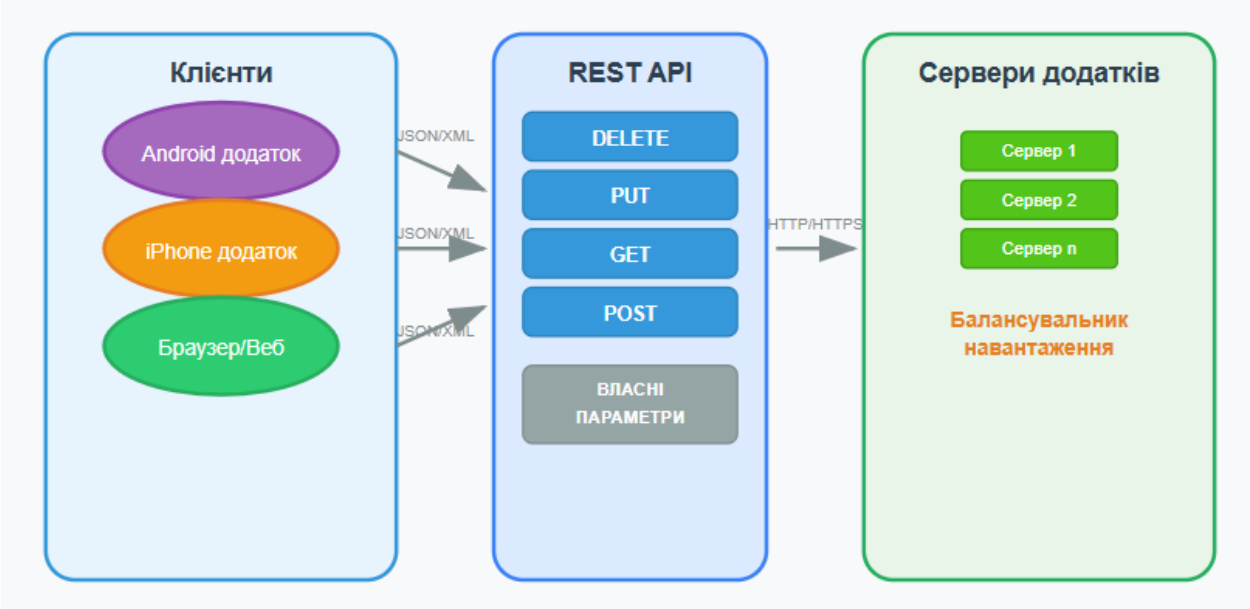

Рис. 2.5 Архітектура REST API

2.4 API GraphQL

GraphQL – це мова запитів та серверне середовище для API з відкритим кодом. З'явився у 2012 році та був розроблений для полегшення керування кінцевими точками для API на основі REST. З 2015 року GraphQL має відкритий код.

Розробники шукали можливість прискорити роботу свого мобільного додатку. Постала проблема , при одночасних запитах з різних баз даних типу MySQL, дотаток втрачав свою продуктивність. Для вирішення цієї задачі у Facebook розробили власну мову запитів, яка спростила форму даних для запиту. Особливо це актуально для соцмереж, де багато зв'язків та запитів по створених елементах.

Перевагою GraphQL на відміну від REST API стала відсутність надлишку чи нестача даних у відповіді. У REST API клієнти часто отримують або занадто багато даних, які їм не потрібні, або занадто мало, через що доводиться робити кілька запитів для отримання необхідної інформації. GraphQL дозволяє клінтам запрошувати тільки ті дані, котрі їм потрібні, та отримувати їх в одному запиті (див. рис. 2.6), що робить комунікацію більш ефективною.

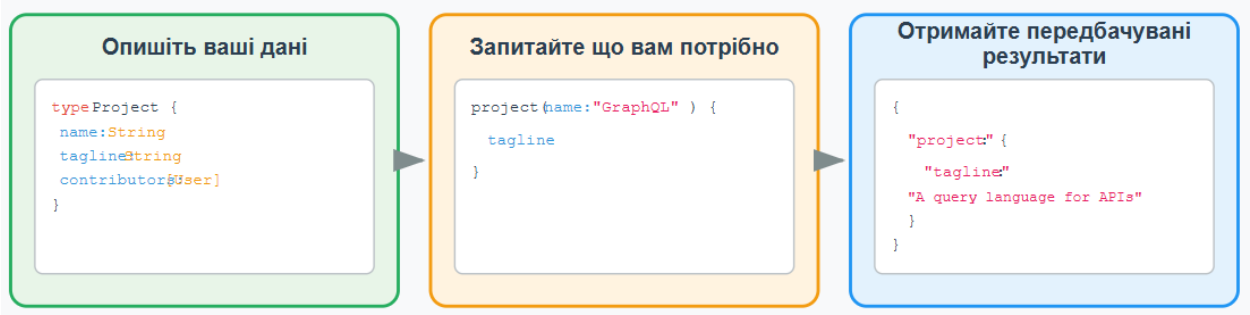

Рис. 2.6 Робочий процес GraphQL

GraphQL використовується в мобільних API, адже саме у цьому випадку важлива продуктивність мережі. Також GraphQL, агрегуючи дані з кількох місць, об'єднує їх в одну глобадьну схему, що актуально для застарілих інфраструктур, що розщирилися.

Отже, GraphQL поступається складністю заради потужності. Наявність великої кількості вкладених полів в запиті може призвести до перенавантаження системи в деяких сценаріях. Тому REST залишається більш універсальним та стабільним варіантом для складних запитів.

2.5 Архітектура інтеграційного рішення між Drupal та Redmine

Розроблене інтеграційне рішення базується на клієнт-серверній архітектурі з використанням принципів REST. Система складається з двох основних компонентів: серверної частини та клієнтської частини Drupal, між якими здійснюється взаємодія через REST API (рис. 2.7).

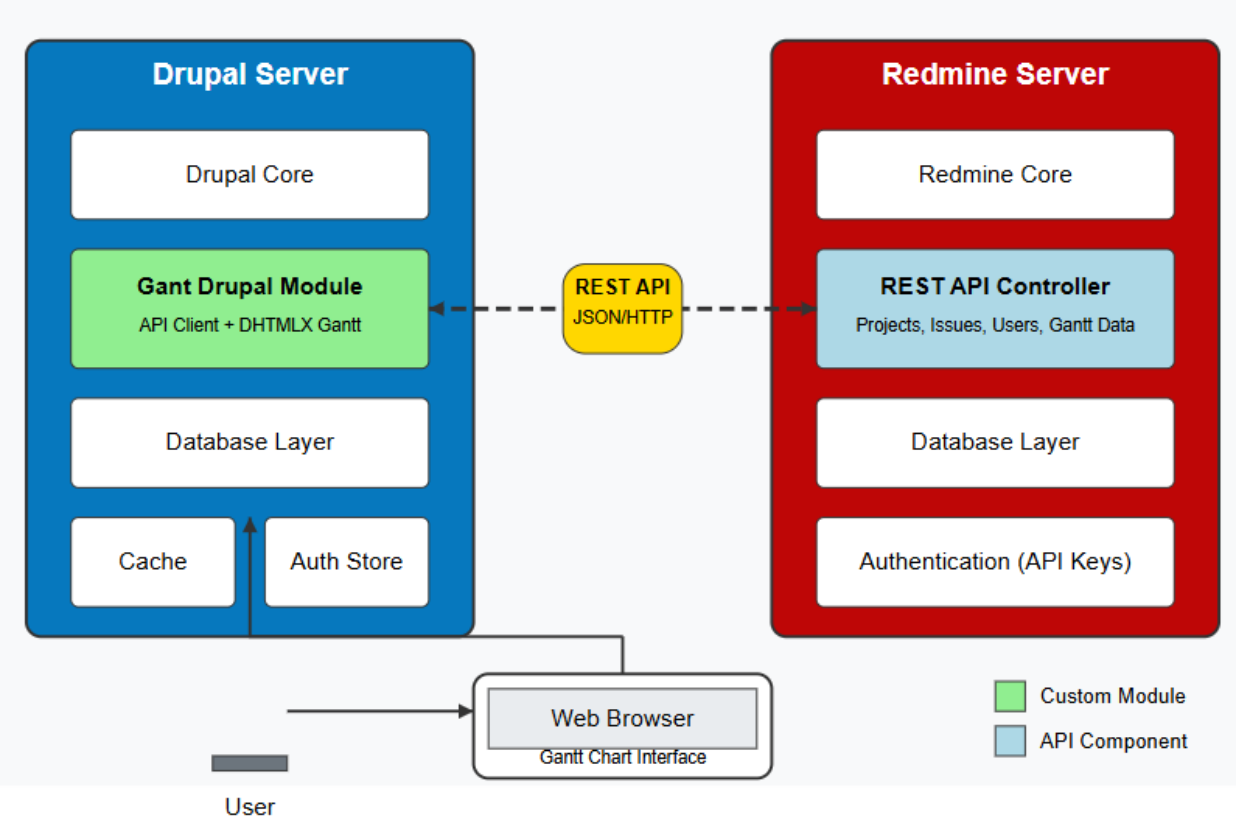

Рис. 2.7 Архітектура інтеграційного рішення Drupal-Redmine

Серверна частина Redmine складається з:

- Ядро Redmine , що реалізує логіку управлля проектами;
- REST API Controller, що обробляє зовнішні запити та формує відповіді у форматі JSON;
- База даних, що зберігає інформацію проект, завдання, користувачів;

- Система аутентифікації на основі API – ключів.

Клієнтська частина Drupal складається з:

- Ядро Drupal, що забезпечує базовий функціонал CMS;
- Модуль "Gant Drupal Module", який є основою інтеграції;
- Компонент API Client для з'єднання з Redmine;
- Інтеграція з DHTMLX Gantt для візуалізації даних;
- Рівень кешування для оптимізації продуктивності.

2.6 Алгоритми обробки даних між системами

Алгоритм синхронізації даних між системами

Забезпечення коректної взаємодії між системами у рамках проекту здійснено за допомогою розробленого алгоритму синхронізації даних. Його мета автоматизувати процес передачі інформації та забезпечити актуальність даних у обох системах.

Для розуміння роботи алгоритму було створено псевдокод, який в подальшому реалізовий у вигляді фунції у модуль Drupal на мові програмування PHP.

```
ФУНКЦІЯ SynchronizeProjects(apiKey, serverId)
  // Крок 1: Отримання даних з Redmine
  projects ← GetProjectsFromRedmine(apiKey, serverId)
  // Крок 2: Фільтрація проектів за часом останнього оновлення
  IF CacheExists(serverId) THEN
    lastUpdate ← GetLastUpdateTimestamp(serverId)
    projects ← FilterByUpdateTime(projects, lastUpdate)
  END IF
  // Крок 3: Збереження даних у Drupal
  FOR EACH project IN projects DO
    SaveProjectToDrupal(project)
    // Крок 4: Синхронізація завдань для кожного проекту
```

```
        tasks ← GetTasksForProject(project.id, apiKey)
        FOR EACH task IN tasks DO
            SaveTaskToDrupal(task)
        END FOR
    END FOR
    // Крок 5: Оновлення часу останньої синхронізації
    UpdateLastSyncTimestamp(serverId, CurrentTime())
    RETURN SuccessMessage
КІН_ФУНКЦІЇ
```

Алгоритм починається з надсилання запиту до REST API системи Redmine за допомогою переданого apiKey (ключ доступу) та ідентифікатора сервера serverId. У відповідь повертається список проектів. Якщо раніше вже виконувалася синхронізація, алгоритм перевіряє наявність кешованого часу останнього оновлення і фільтрує отримані проекти, враховуючи лише ті, що були змінені після цього часу. Кожен із відфільтрованих проектів зберігається у базі даних Drupal. Для кожного проекту отримується список завдань через API Redmine, які також зберігаються у Drupal. Алгоритм завершується оновленням часу останньої синхронізації, що забезпечує точність у майбутніх викликах алгоритму.

Алгоритм генерації діаграми Ганта

Для інтерактивного відображення проектних завдань у вигляді діаграми Ганта був реалізований алгоритм, який готує структуровані дані для бібліотеки DHTMLX Gantt. Ця діаграма є ключовим інструментом управління проектами, що дозволяє візуалізувати прогрес, залежності між завданнями та критичний шлях.

Псевдокод алгоритму:

```
ФУНКЦІЯ GenerateGanttData(projectId)
    // Крок 1: Отримання завдань проекту
    tasks ← GetTasksForProjectFromLocalDB(projectId)
```

```
    // Крок 2: Обробка ієрархічної структури завдань
    hierarchicalTasks ← BuildTaskHierarchy(tasks)
      // Крок 3: Обчислення критичного шляху
    CalculateCriticalPath(hierarchicalTasks)
      // Крок 4: Формування даних для DHTMLX Gantt
    ganttData ← []
      FOR EACH task IN hierarchicalTasks DO
      ganttItem ← {
        "id": task.id,
        "text": task.name,
        "start_date": FormatDate(task.start_date),
        "end_date": FormatDate(task.due_date),
        "progress": task.completion_percentage / 100,
        "parent": task.parent_id,
        "priority": task.priority,
        "critical": task.is_critical
      }
      ganttData.APPEND(ganttItem)
    END FOR
      RETURN ganttData
КІН_ФУНКЦІЇ
```

З локальної бази даних Drupal отримуються завдання, що належать до певного проекту. Цей крок забезпечує швидкий доступ до даних і знижує потребу у зверненні до зовнішнього API. Завдання організовуються у вигляді ієрархії, наприклад, створюються зв’язки між головними завданнями та їхніми підзадачами, що забезпечує ясність у взаємозалежностях. Розраховується критичний шлях, який включає завдання, без виконання яких неможливо завершити проект у встановлені терміни. Це допомагає визначити

найважливіші завдання. ідготовка структурованих даних у форматі, який може використовувати бібліотека DHTMLX Gantt. Це включає ключові параметри завдань, такі як:

- ID завдання.
- Назва завдання.
- Дати початку та завершення.
- Прогрес виконання (у відсотках).
- Ідентифікатор батьківського завдання (для створення ієрархії).
- Пріоритет завдання і позначення критичних.

Висновки

Проаналізувавши сучасні методи інтеграції програмних систем можна стверджувати, що серед архітектурних підходів до API кожен має специфічні переваги та власну область застосування (див. рис. 2.8).

АРХІТЕКТУРНІ СТИЛІ API

	SOAP	REST	GraphQL
Організований у термінах	структура обгорнутого повідомлення	відповідність шести архітектурним обмеженням	схема і система типів
Формат	тільки XML	XML, JSON, HTML, звичайний текст	JSON
Крива навчання	Складно	Легко	Середньо
Спільнота	Мала	Велика	Зростає
Випадки використання	Платіжні шлюзи, управління ідентичністю, CRM рішення, фінансові та телекомунікаційні сервіси, підтримка застарілих систем	Публічні API, прості ресурсо-орієнтовані додатки	Мобільні API, складні системи, мікросервіси

Рис. 2.8 Порівняння особливостей SOAP, REST, GraphQL

SOAP забезпечує навищий рівень безпеки, є вдалим для корпоративних рішень, має складну структуру. GraphQL, у свою чергу, демонструє високу ефективність у сфері мобільних додатків, завдяки точним запитам необхідних

даних та високій потужності, проте має обмеження щодо складних запитів. На сьогодні REST API є найбільш універсальним рішенням для інтгерації систем Drupal та Redmine. Поєднує в собі простоту реалізації, гнучкість архітектури. Принципи REST забезпечують надійність та перспективу масштабування взаємодії без надмірної складності SOAP або обмежень GraphQL. Наразі REST має більшу екосистему веб-сервісів, через простоту впровадження та глобальній спільноті розробників. Проте GraphQL, що є відносно новим рішенням, має всі перспективи заполонити сфери, де критично важлива оптимізація мережевого трафіку.

РОЗДІЛ 3

РОЗРОБКА ПРОТОТИПУ ІНТЕГРАЦІЙНОГО РІШЕННЯ

3.1 Аналіз вимог до апаратного забезпечення серверного рішення

Redmine, будучи веб-додатком на Ruby-on-Rails, згідно з вимогами має працювати під веб-сервером. Роль незалежного веб-сервера можуть відігравати Apache або Nginx, що запускає Ruby-on-Rails через Passenger. Також може бути використаний спеціалізований Ruby веб-сервер, такий як Unicorn або Thin. Redmine є складною системою з рядом технічних залежностей, які дозволяють їй працювати безперебійно. Існує безліч операційних систем, серверів баз даних, утиліт і т.д. Щоб досягти оптимальної продуктивності та функціонування в цілому, визначено певний набір підтримуваних компонентів.

У таблиці 3.1 представлено мінімальні вимоги до обладнання. Ці специфікації розраховані на 25 одночасних користувачів із середньою частотою запитів на секунду.

Таблиця 3.1

Мінімальні вимоги до обладнання для реалізації інтеграції

Компонент	**Специфікація**	**Примітки**
Процесор	AMD Ryzen 7 PRO 7745	Або еквівалент Intel
Оперативна пам'ять	4 Гб	DDR4 або вище
Дисковий простір	~ 50 Гб	3 ГБ для БД + простір для вкладень
Мережеве підключення	Ethernet 1 Гбіт/с	Обов'язкова наявність мережевої карти

Для більших реалізацій технічні вимоги зростають експоненціально залежно від кількості користувачів, табл. 3.2:

Таблиця 3.2

Вимоги до обладнання з перспективою масштабування

Користувачі	Потоки/vCPU	ОЗП	Дисковий простір
50	8	12 ГБ	50 ГБ
100	12	32 ГБ	100 ГБ
200	24	64 ГБ	200 ГБ
500	32	128 ГБ	500 ГБ

Значення в таблицях не є жорстко визначеними та конфігурація залежить від масштабів і значимості використання Redmine на підприємстві. А також на частоту доступу користувачів і запитів, що відправляються на сервер.

З'єднання користувача з сервером, де знаходиться Redmine, має бути не менше 10 Мбіт. Потрібно враховувати весь маршрут від користувача до сервера, тому при географічно віддалених локаціях відповідь сервера може зайняти більш тривалий час, незалежно від заявленого інтернет-з'єднання користувача від свого провайдера.

3.2 Аналіз варіантів розміщення інфраструктури

Інфраструктура інтеграційного рішення може бути розміщена на: локальному хостингу, провайдерських серверах та хмарних рішеннях.

Локальних хостинг передбачає для розміщення додатків використання безспосередньо власного обладнання. Це стало найпоширенішим підходом для організації серверної інфраструктури, особливо для компаній, що мають серверні кімнати та команди для їх обслуговування. Ключовою перевагою такого рішення є абсолютний контроль над апаратним та програмним

забезпеченням, що гарантує максимальний рівень захисту конфіденційної інформації. Не менш важливими аспектами є незалежність від зовнішніх провайдерів та можливість налаштування середовища під специфічні потреби.

Альтернативою локальному хостингу є використання послуг провайдерів серверів, що надають в оренду серверну інфраструктуру. Перевагою такого підходу є зниження витрат, адже відсутні інвестиції у власне обладнання. Замовлення послуг у провайдера відкриває можливість зосередитися на розробці та основних завдання підприємства.

Хмарні сервіси (AWS, Google Cloud, Microsoft Azure) представляють найсучасніший підхід до організації інфраструктури, використовуючи потужності великих дата-центрів. Перевагою використання хмарних рішень є їх масштабованість ресурсів, високий рівень надійності, різноманітність сервісів та відповідно широкий спектр готових рішень. З точки зору фінансової рентабельності даного підходу, можна вважати його економічно привабливішим за вищеописані варіанти, особливо для малих підриємств.

Для даного проекту було обрано локальний підхід з використанням Docker-контейнерізації з наступних причин:

1. **Освітні цілі проекту.** Дана робота має на меті демонстрацію технічних навичок та розуміння процесів розробки. Локальне середовище надає:

   - Повний контроль над процесом налаштування;
   - Детальне вивчення всіх компонентів системи;
   - Можливість експериментування з конфігураціями.

2. **Економічні міркування.** Для навчального проекту локальне середовище є найбільш економічно виправданим:

   - Відсутність щомісячних платежів за хостинг;
   - Використання наявних апаратних ресурсів;
   - Незалежність від зовнішніх провайдерів.

3. **Специфіка розробки та тестування.** Етап розробки та прототипування вимагає:

   - Частих змін конфігурації;
   - Швидких циклів розробка-тестування;
   - Можливості швидкого відкату змін.

4. **Безпека та приватність**. Локальне середовище гарантує:

   - Повну конфіденційність розробки;
   - Мінімізацію ризиків витоку коду;
   - Контроль над даними.

Docker як оптимальне рішення для локальної розробки

У межах даного проекту використання контейнеризованої архітектури на базі Docker, що поєднує переваги локального хостингу з гнучкістю хмарних технологій. Контейнери ізолюють процеси без надмірних витрат для повної віртуалізації. Ключовою перевагою є можливість перенесення готового інтеграційного рішення на будь-яку хостинг платформу.

Технічні переваги:

- **Ізоляція**: Повна ізоляція середовища розробки від хостової ОС;
- **Портативність**: Можливість розгортання на будь-якій системі з Docker;
- **Масштабованість**: Легке масштабування компонентів системи;
- **Консистентність**: Ідентичне середовище на всіх етапах розробки.

Операційні переваги:

- **Швидке розгортання**: Автоматизація процесу встановлення;
- **Версіонування**: Можливість використання різних версій залежностей;
- **Відкат**: Швидкий повернення до попередніх конфігурацій;

Continuous Integration/ Continuous Deployment інтеграція: Легка інтеграція з системами автоматизації.

3.3 Програмні вимоги та залежності

Інтеграційне рішення базується на класичній архітектурі веб-додатків, що включає компоненти LAMP (Linux, Apache, MySQL, PHP) стеку для Drupal та додаткові компоненти для підтримки Ruby on Rails (Redmine). Така архітектура забезпечує стабільність, масштабованість та сумісність з широким спектром хостинг-провайдерів.

Redmine - технічний стек Ruby on Rails

Оскільки інтеграційне рішення взаємодіє з системою управління проектами Redmine, необхідно враховувати специфічні вимоги цієї платформи табл 3.3.

Таблиця 3.3

Програмні вимоги для Redmine

Компонент	Версія	Призначення
Ruby	2.7+ / 3.0+	Основна мова виконання Rails
Ruby on Rails	6.1+ / 7.0+	Веб-фреймворк
Веб-сервер	Apache + Passenger / Nginx + Unicorn	HTTP сервер з підтримкою Ruby
База даних	MySQL 8.0+ / PostgreSQL 12+	Зберігання проектних даних
Bundler	2.0+	Управління Ruby gems
ImageMagick	7.0+	Обробка зображень та файлів

Специфічні залежності Redmine:

- Passenger: Модуль Apache/Nginx для запуску Rails додатків;
- RMagick: Ruby інтерфейс для ImageMagick;
- Rails API: Вбудований REST API функціонал;
- JSON gem: Серіалізація даних для API відповідей.

Drupal 10 - технічний стек PHP

Для розробки модуля інтеграції з Drupal 10 необхідно забезпечити наступне програмне середовище (див. табл. 3.4):

Таблиця 3.4

Програмні вимоги для Drupal 10

Компонент	Версія	Призначення
PHP	8.1+	Основна мова виконання
Веб-сервер	Apache 2.4+ / Nginx 1.18+	HTTP сервер
База даних	MySQL 8.0+ / MariaDB 10.5+	Зберігання даних CMS
Composer	2.0+	Управління залежностями
Git	2.30+	Контроль версій

Інтеграційні вимоги для взаємодії систем

Оскільки проект передбачає інтеграцію між PHP-додатком (Drupal) та Ruby on Rails додатком (Redmine), необхідно забезпечити сумісність протоколів взаємодії:

HTTP/REST API вимоги:

- **HTTP клієнт**: Підтримка HTTP/1.1 та HTTP/2;
- **SSL/TLS**: Для безпечної взаємодії через HTTPS;
- **JSON**: Стандартний формат обміну даними;
- **API аутентифікація**: Підтримка header-based токенів.

Додаткові PHP розширення для Drupal

Для коректної роботи інтеграційного модуля необхідно забезпечити наявність наступних PHP розширень:

- **cURL**: Для HTTP-запитів до Redmine API;

- **JSON**: Для обробки JSON-відповідей від Rails додатку;
- **GD/ImageMagick**: Для обробки зображень;
- **OpenSSL**: Для безпечних HTTPS з'єднань з Redmine;
- **Mbstring**: Для роботи з Unicode в міжнародних проектах [4].

Залежності JavaScript для фронтенд компонентів

Для візуалізації діаграм Ганта використовуються клієнтські технології:

- **dhtmlxGantt**: JavaScript бібліотека для відображення діаграм;
- **jQuery**: Для DOM маніпуляцій (опціонально);
- **CSS3**: Для стилізації інтерфейсу.

3.4 Архітектурна організація модуля інтеграції

Побудова модулю інтеграції Drupal з Redmine здійснюється за принципами модульної архітектури, яка в свою чергу забезпечує чітке розділення відповідальності між компонентами системи. Це дозволяє підтримувати код структурованим та легко розширюваним. Ключові принципи закладені у архітектуру модуля:

- **Розділення відповідальності.** Як було сказано раніше, компоненти модуля відповідають за конкретну функціональність та не перетинаються з іншими компонентами.
- **Слабка зв’язність.** Взаємодія між компонентами здійснюється через чітко визначені інтерфейси. Це дозволяє змінювати реалізацію одного з компонентів без впливу на інші.
- **Висока згуртованість.** Елементи наповнення кожного компонента тісно пов’язані між собою, працюють для досягнення спільної мети.
- **Інверсія залежностей** (Dependency Inversion/DI) Для управління залежностями між компонентами використовується контейнер DI [7].

Архітектура модуля базується на підході багатошаровості (див. рис. 3.1).

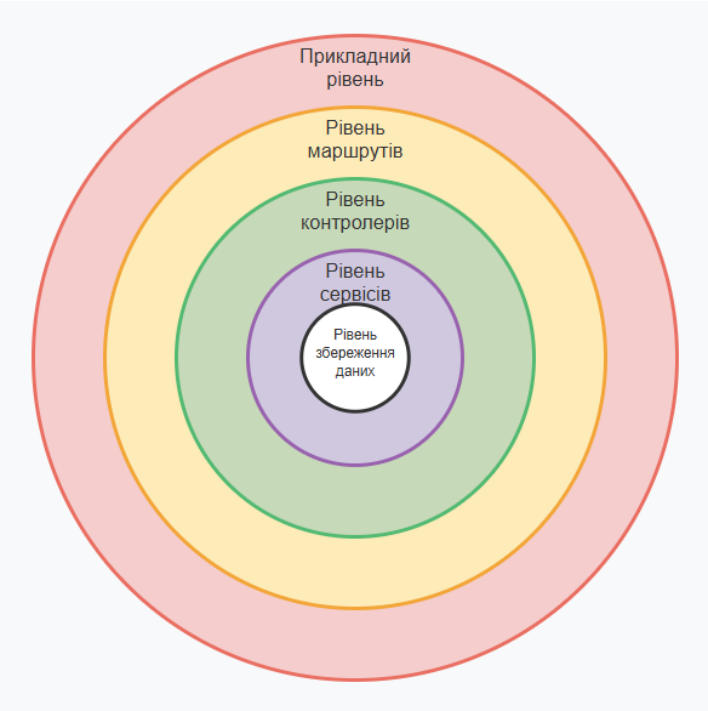

Рис. 3.1 Багатошарова архітектура модуля

Прикладний рівень

На найвищому рівні знаходиться шар конфігурації модуля, представлений файлом gant_drupal_module.module та конфігураційними файлами. Цей шар відповідає за:

- Реєстрацію модуля в системі Drupal;
- Визначення тем та шаблонів через хук hook_theme();
- Реалізацію системних хуків Drupal;
- Базову конфігурацію модуля та його метадані.

Рівень маршрутів

Шар маршрутизації визначає точки входу до функціональності модуля через файл gant_drupal_module.routing.yml. Він забезпечує:

- Визначення URL-шляхів для доступу до функціональності модуля;

- Зв'язування URL з відповідними контролерами та формами;
- Налаштування прав доступу до окремих маршрутів;
- Конфігурацію параметрів маршрутів та їх валідацію.

Рівень контролерів

Контролери (RedmineController.php) відповідають за обробку HTTP-запитів та підготовку відповідей. Їх відповідальність включає:

- Отримання та валідацію вхідних параметрів;
- Виклик відповідних сервісів для обробки функціональної логіки;
- Підготовку даних для передачі до шаблонів відображення;
- Формування структурованих HTTP-відповідей.

Рівень сервісів

Сервіси інкапсулюють основну функціональну логіку модуля та забезпечують взаємодію із зовнішніми системами. Сервіси виконують наступні функції:

- Інкапсуляцію функціональної логіки модуля;
- Взаємодію із зовнішнім Redmine API;
- Обробку та трансформацію даних між системами;
- Логування операцій та обробку помилок.

Рівень збереження даних

У контексті модуля інтеграції, шар збереження представлений зовнішньою системою Redmine, з якою модуль взаємодіє через REST API. Цей шар абстрагований через методи сервісу та забезпечує:

- Доступ до даних проектів, задач та користувачів Redmine;
- Зберігання конфігураційних налаштувань модуля в базі даних Drupal;

- Кешування отриманих з Redmine даних для оптимізації продуктивності.

3.5 Створення мережевої інфраструктури

Для забезпечення комунікації між контейнерами було створено спільну Docker мережу (див. рис. 3.2):

```
# Створення ізольованої мережі для проекту
docker network create integration-network

# Перевірка створення мережі
docker network ls | grep integration-network
```

Рис. 3.2 Перевірка створення мережі

Domain Name System резолюція між контейнерами за іменами та додаткова ізоляція від інших Docker проектів реалізується завдяки використанню кастомної мережі замість стандартної bridge мережі.

Розгортання системи відбувається у наступній послідовності, задля забезпечення правильних залежностей проекту:

1. Першим етапом відбувається розгортання бази даних для Redmine та підключення до неї. У нашому випадку було обрано для Redmine SQLite - файлову базу даних, що не потребує окремого серверного рішення.

```
docker run -d \ # Запуск у фоні
-p 3000:3000 \ # Порт 3000
--name redmine \ # Ім'я контейнера
```

```
-v redmine_data:/usr/src/redmine/files \ # Volume для файлів
redmine:4.2 # Образ Redmine
```

Рис. 3.3 Перевірка запуску контейнера

```
# Перевірка доступності веб-інтерфейсу
curl -I http://localhost:3000
```

Від веб-сервера отримано позитивну відповідь (див. рис. 3.4), статус HTTP 200 OK разом із необхіднми заголовками та активною сесією Redmine свідчить про готовність системи до роботи.

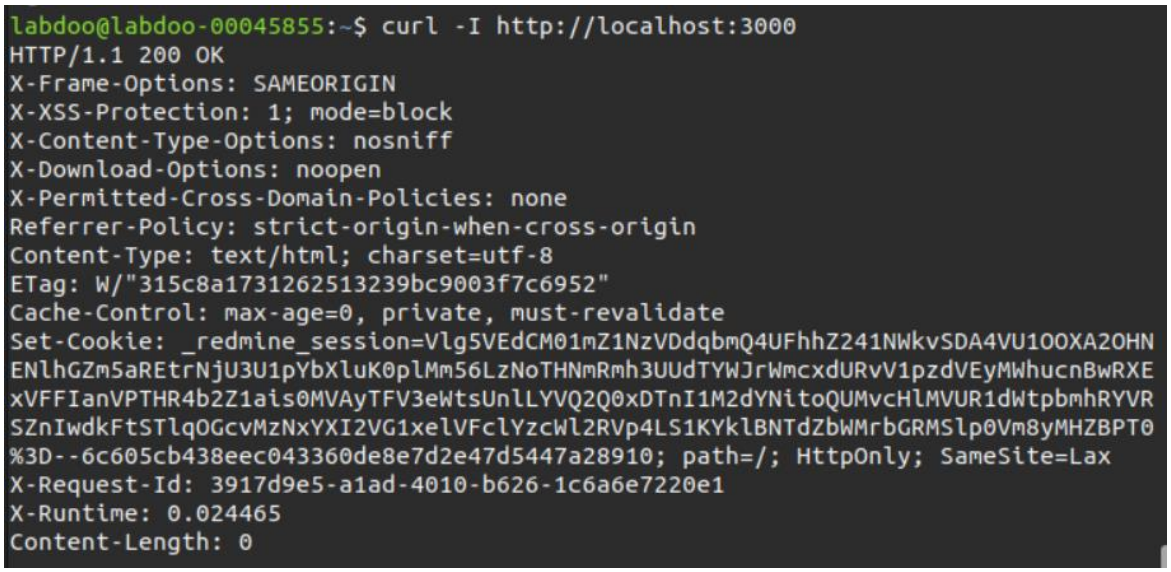

Рис. 3.4 Доступність веб-інтерфейсу Redmine

2. Створення окремої бази даних для Drupal для ізоляції даних;

```
# Запуск MySQL контейнера для Drupal
docker run -d \
  --name drupal-mysql \
```

```
    --network integration-network \
    -e MYSQL_ROOT_PASSWORD=rootpass \
    -e MYSQL_DATABASE=drupal \
    -e MYSQL_USER=drupal \
    -e MYSQL_PASSWORD=drupalpass \
    -v drupal_mysql_data:/var/lib/mysql \
    --restart unless-stopped \
    mysql:5.7
# Верифікація створення бази даних
docker exec -it drupal-mysql mysql -u drupal -pdrupalpass -e "SELECT DATABASE();"
```

Назву активної бази даних повернуто (див. рис. 3.5), що підтверджує правильність налаштування між Drupal CMS та MySQL сервером.

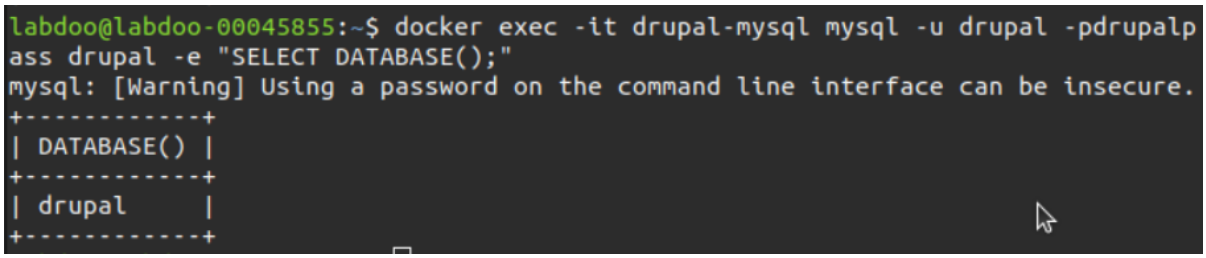

Рис. 3.5 Тестування з'єднання з базою даних MySQL через Docker-контейнер

```
# Запуск Drupal контейнера
docker run -d \
  --name drupal \
  --network integration-network \
  -p 8080:80 \
```

```
  -v drupal_modules:/opt/drupal/web/modules/custom \
  -v drupal_themes:/opt/drupal/web/themes/custom \
  -v drupal_sites:/opt/drupal/web/sites \
  --restart unless-stopped \
  drupal:10
# Перевірка доступності веб-інтерфейсу
curl -I http://localhost:8080
```

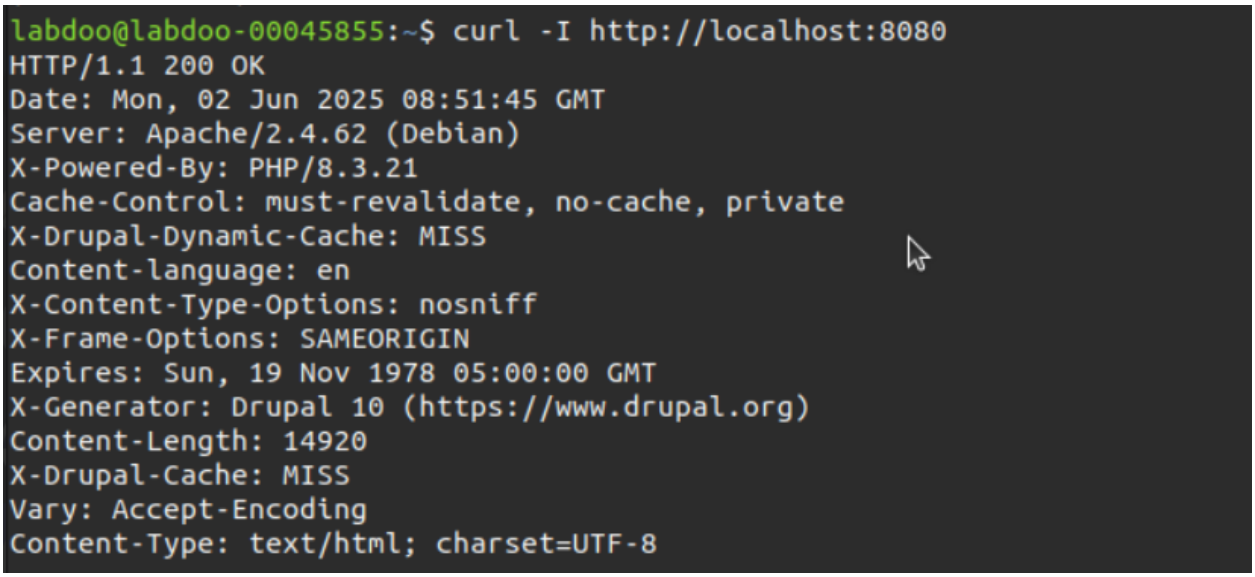

Рис. 3.6 Доступність веб-інтерфейсу Drupal

Кожен з контейнерів налаштовується з відповідними змінними середовища, мережевими підключеннями та томами для збереження даних.

3.6 Первинне налаштування через веб-інтерфейси

Конфігурація Redmine після розгортання

Після успішного розгортання контейнера Redmine необхідно здійснити первинне налаштування системи через веб-інтерфейс, доступний за адресою

http://localhost:3000, використовуючи стандартні облікові дані адміністратора (див. рис. 3.7).

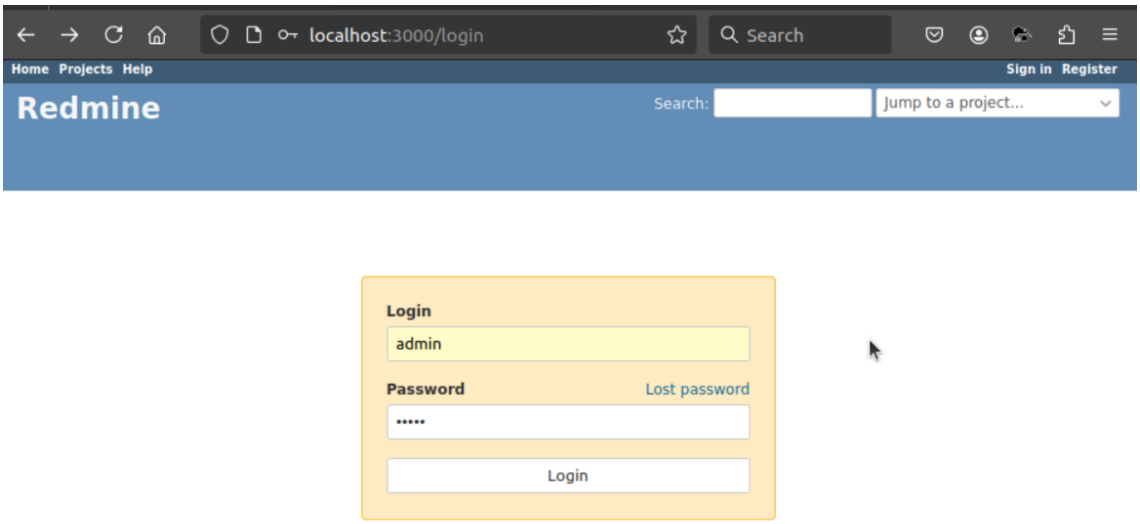

Рис. 3.7 Перша форма автентифікації адміністратора в системі Redmine

Першочерговим завданням є зміна стандартного пароля адміністратора з міркувань безпеки, що здійснюється через перехід до розділу "My account" у персональному профілі користувача, де обирається опція "Change password" для встановлення нового захищеного пароля відповідно до політики безпеки підприємства (див. рис. 3.8).

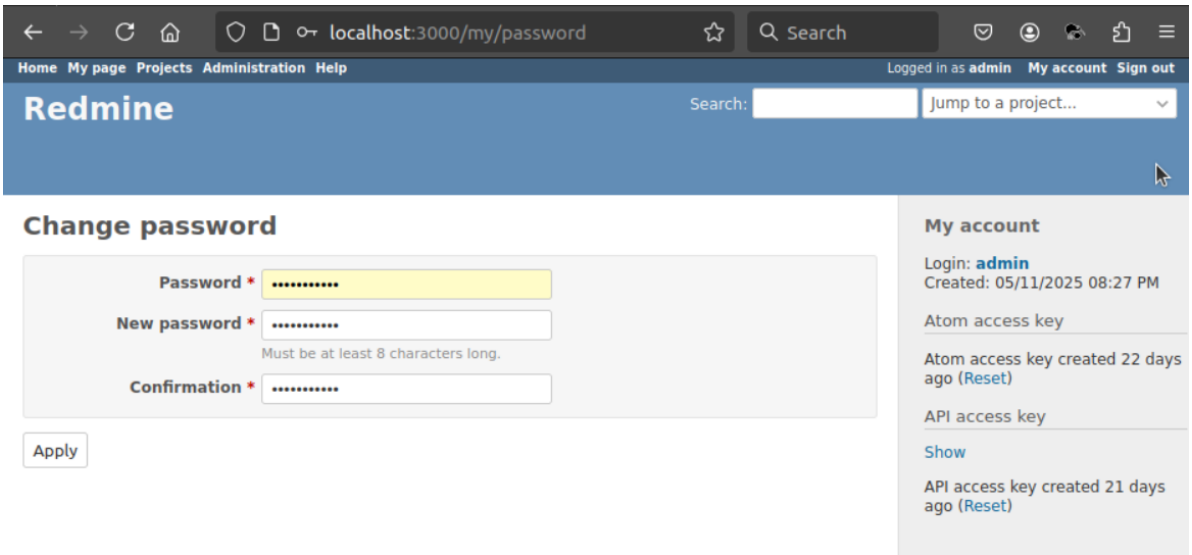

Рис. 3.8 Зміна пароля для користувача «admin»

Наступним етапом налаштування є генерація API ключа, необхідного для програмного доступу до функціональних можливостей Redmine з зовнішніх систем, що виконується через розділ "API access key" в особистому кабінеті користувача, де за допомогою кнопки "Show" створюється унікальний ключ доступу (див. рис. 3.9), який копіюється та зберігається для подальшого використання при розробці модуля інтеграції між системами управління проектами та контентом.

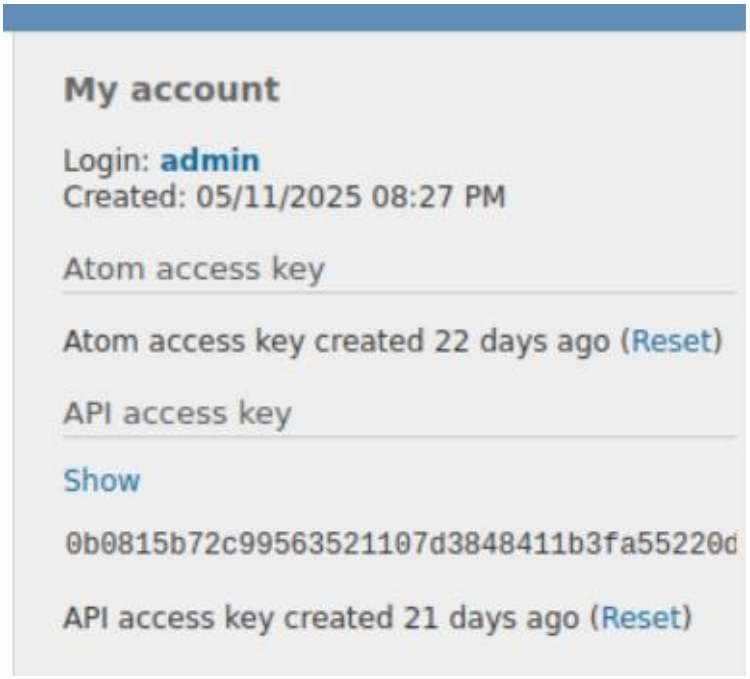

Рис. 3.9 API access key Redmine

Конфігурація Drupal після розгортання

Наступним етапом розгортання інфраструктури є конфігурація системи управління контентом Drupal через веб-інтерфейс, доступний за адресою http://localhost:8080, де запускається процес встановлення та первинного налаштування системи (див. рис. 3.10). У процесі налаштування бази даних необхідно вказати параметри підключення до MySQL сервера, а саме: тип бази даних MySQL, назву бази даних "drupal", ім'я користувача "drupal" з відповідним паролем "drupalpass", хост "drupal-mysql" та стандартний порт

3306, що забезпечує коректне з'єднання між контейнерами у Docker-мережі. Завершальним кроком встановлення є створення адміністративного облікового запису з налаштуванням назви сайту "Drupal-Redmine Integration", що відображає призначення системи для інтеграційних цілей, та введенням облікових даних адміністратора включаючи логін "admin", безпечний пароль та робочу електронну адресу, необхідні для подальшого управління системою та розробки модуля інтеграції.

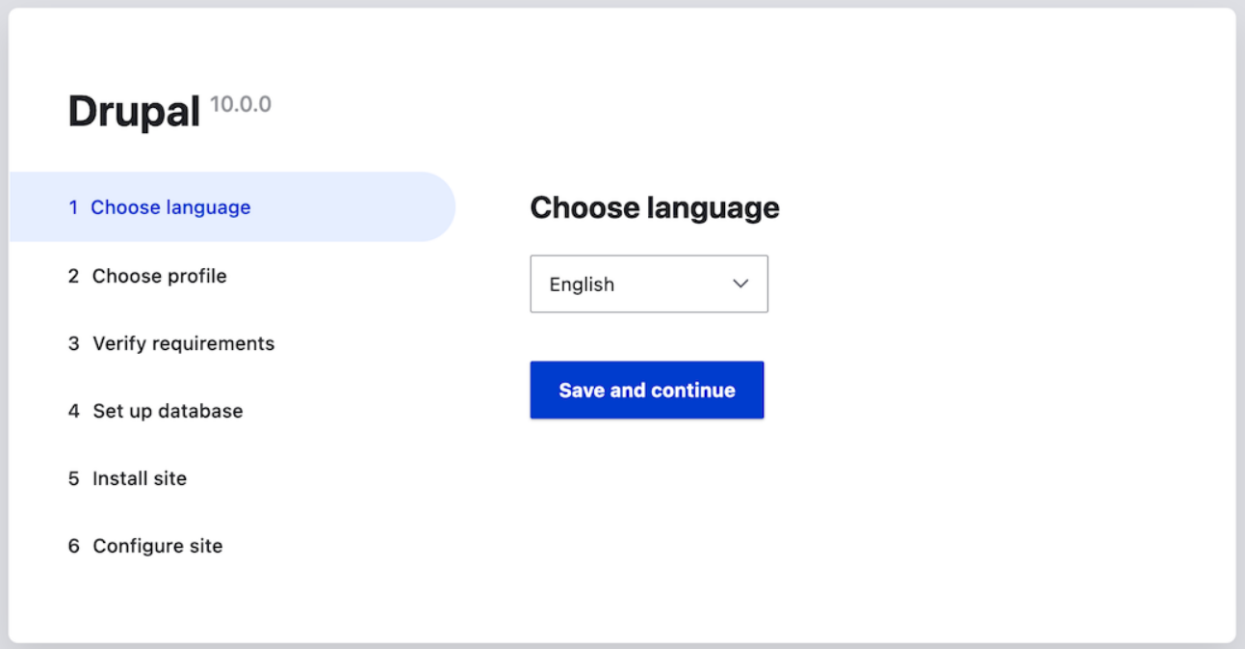

Рис. 3.10 Первинне налаштування системи Drupal 10

3.7 Розробка модуля “Gant Drupal Module”

Розробка модуля "Gant Drupal Module" розпочалася зі створення файлової структури, що відповідає стандартам Drupal 10.

Для забезпечення коректної організації коду було створено наступну ієрархію каталогів (див. рис. 3.11):

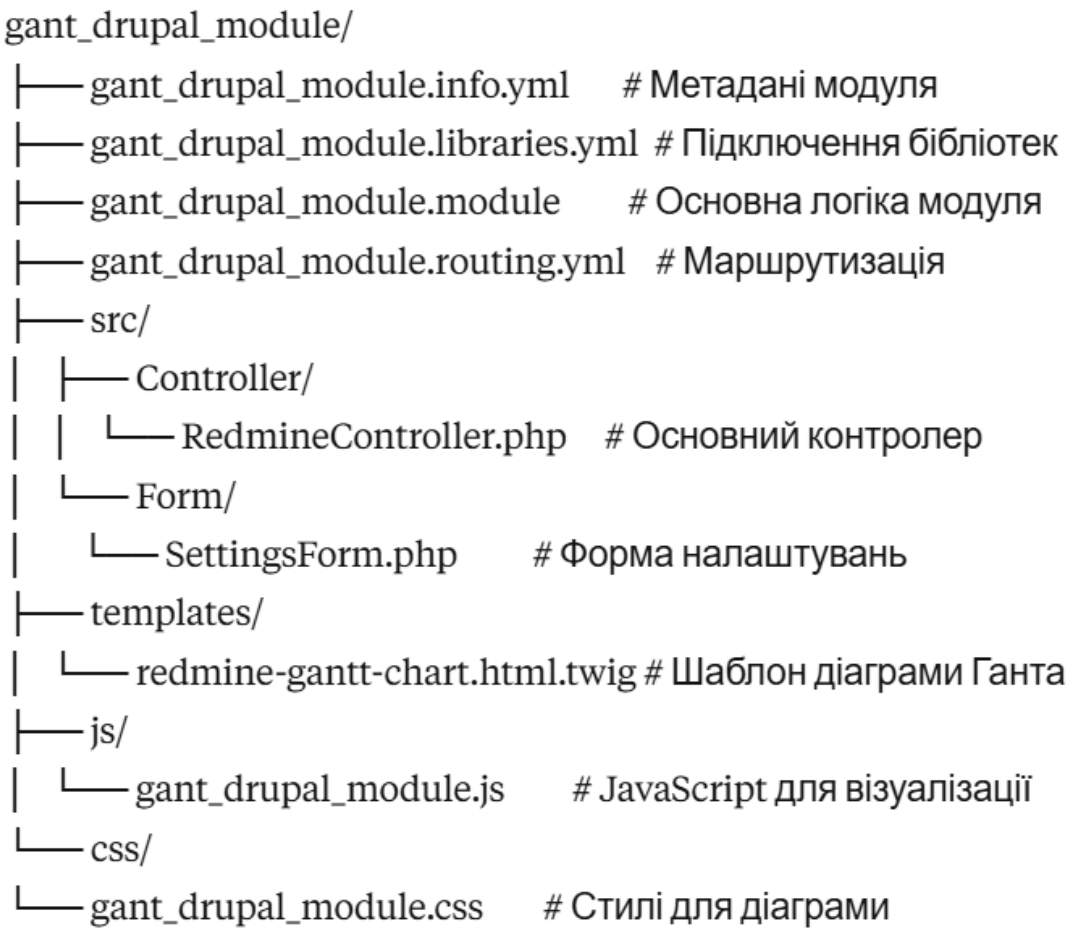

Рис. 3.11 Фізична структура модуля «Gant Drupal Module»

Для створення структури модуля необхідно підключитися до контейнера Drupal через команду:

```
# Підключення до контейнера Drupal
docker exec -it drupal bash
# Перехід до директорії кастомних модулів
cd /opt/drupal/web/modules/custom/
# Створення основної папки модуля
mkdir gant_drupal_module
cd gant_drupal_module
```

```
# Створення всіх необхідних піддиректорій [18]
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/src/Controller
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/src/Form
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/templates
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/js
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/css
```

3.7.1 Створення метаданих модуля

Файл `gant_drupal_module.info.yml` містить метадані модуля та визначає його залежності.

```
# Створення файлу метаданих
nano gant_drupal_module.info.yml
```

Вміст файлу:

```
name: Gant Drupal Module
type: module
description: 'Інтеграція діаграм Ганта з Redmine на сайті Drupal. '
package: Custom
core_version_requirement: ^9 || ^10
dependencies:
  - drupal:rest
```

Ключовими елементами конфігурації є сумісність з версіями Drupal 9-10 та залежність від модуля REST API.

Створення основного файлу модуля

Файл gant_drupal_module.module містить основні hook-функції модуля:

```
# Створіть основний файл модуля
cat > gant_drupal_module.module << 'EOL'
# Фрагмент коду (повний код див. Додаток А)
<?php
use Drupal\Core\Routing\RouteMatchInterface;

function gant_drupal_module_help($route_name, RouteMatchInterface $route_match) {
  // Довідкова інформація про модуль
}

function gant_drupal_module_theme() {
  return [
    'gant_chart' => [
      'variables' => ['projects' => [], 'chart_id' => 'gantt-chart'],
      'template' => 'gant-chart',
    ],
  ];
}
EOL
```

Цей файл реалізує два ключові hook-и: hook_help() забезпечує довідкову інформацію про модуль в адміністративному інтерфейсі, а hook_theme() реєструє кастомний шаблон gant-chart для відображення діаграми Ганта з передачею змінних проектів та ідентифікатора діаграми.

3.7.2 Розробка системи маршрутизації

У файлі gant_drupal_module.routing.yml визначено три основні маршрути системи (див. рис. 3.12):

1. **Маршрут налаштувань модуля** (gant_drupal_module.settings) - забезпечує доступ до адміністративної панелі конфігурації;
2. **Маршрут списку проектів** (gant_drupal_module.projects) - відображає всі доступні проекти з Redmine;
3. **Маршрут діаграми Ганта** (gant_drupal_module.project_gantt) - генерує діаграму для конкретного проекту.

Кожен маршрут має визначені права доступу та відповідний контролер для обробки запитів.

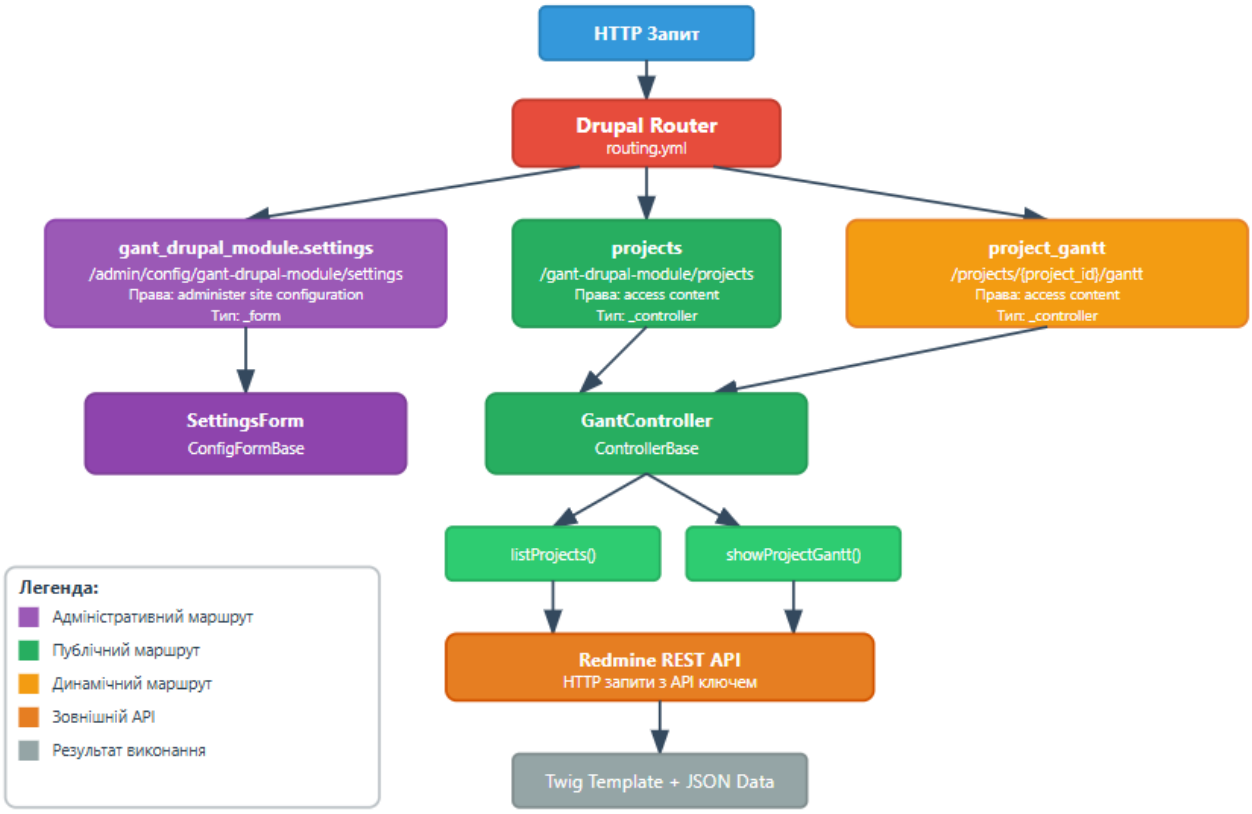

Рис. 3.12 Схема маршрутизації модуля Gant Drupal Module

Публічний маршрут списку проектів (**gant_drupal_module.projects**)

Маршрут /gant-drupal-module/projects призначений для відображення переліку доступних проектів з системи Redmine. На відміну від адміністративного маршруту, цей використовує параметр _controller, що вказує на метод listProjects класу GantController. Така архітектура дозволяє реалізувати складнішу логіку обробки даних, включаючи взаємодію з зовнішнім API, кешування результатів та формування структурованого виводу.

Динамічний маршрут діаграми Ганта (**gant_drupal_module.project_gantt**)

Найскладніший маршрут системи використовує динамічний параметр {project_id}, який автоматично передається до методу контролера як аргумент. Це дозволяє створювати URL вигляду /gant-drupal-module/projects/15/gantt, де число 15 автоматично стає доступним в методі showProjectGantt() для завантаження конкретних даних проекту.

Система прав доступу та безпека

Кожен маршрут має чітко визначені вимоги до прав доступу через секцію requirements. Адміністративний маршрут використовує право administer site configuration, що обмежує доступ лише користувачами з адміністративними привілеями. Публічні маршрути використовують базове право access content, що дозволяє перегляд авторизованим користувачам сайту. Така градація забезпечує належний рівень безпеки при збереженні зручності використання.

Інтеграція з системою меню Drupal

Завдяки розміщенню адміністративного маршруту в стандартній структурі /admin/config/, модуль автоматично інтегрується в систему адміністративного меню Drupal. Параметр _title для кожного маршруту забезпечує відображення

описових назв в breadcrumb навігації та заголовках сторінок, що покращує користувацький досвід.

Масштабованість архітектури маршрутизації

Поточна структура маршрутів дозволяє легко розширювати функціональність модуля без порушення існуючих URL. Можна додавати нові маршрути для додаткових функцій, таких як експорт даних, детальний перегляд завдань або налаштування візуалізації, зберігаючи консистентність в іменуванні та структурі шляхів.

Для забезпечення безпеки реалізовано дворівневу систему доступу:

- Адміністративні функції доступні лише користувачам з правами administer site configuration;
- Перегляд діаграм доступний всім авторизованим користувачам з правами access content.

3.7.3 Клас GantController - архітектура та логіка роботи

Основна логіка взаємодії з Redmine API реалізована в класі GantController, який успадковує від ControllerBase та забезпечує повноцінну інтеграцію між системами управління контентом та проектами.

Структура класу та залежності

Клас використовує модульну архітектуру з чітким розділенням відповідальності.

Імпортуються необхідні простори імен: ControllerBase для базової функціональності Drupal, JsonResponse для формування JSON відповідей, Client з бібліотеки GuzzleHttp для HTTP запитів та RequestException для обробки мережевих помилок.

```
# Фрагмент php коду класу GantController(поний код див. Додаток А)
namespace Drupal\gant_drupal_module\Controller;
use Drupal\Core\Controller\ControllerBase;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\RequestException;
```

Метод listProjects() - логіка отримання проектів

Етап валідації конфігурації

Метод розпочинається з отримання збережених налаштувань через конфігураційний API Drupal (див. рис. 3.13). Система перевіряє наявність обов'язкових параметрів `redmine_url` та `api_key`. У разі відсутності будь-якого з параметрів, користувачу відображається інформативне повідомлення з інструкцією щодо необхідності налаштування модуля.

```
# Фрагмент php коду класу GantController(поний код див. Додаток А)
$config = $this->config('gant_drupal_module.settings');
$redmine_url = $config->get('redmine_url');
$api_key = $config->get('api_key');
```

Етап HTTP комунікації з Redmine

Після успішної валідації створюється екземпляр HTTP клієнта Guzzle та формується GET запит до endpoint /projects.json Redmine API.

Критично важливим є використання заголовка X-Redmine-API-Key для аутентифікації та Content-Type: application/json для забезпечення коректної обробки даних на стороні Redmine.

Етап обробки та трансформації даних

Отримана JSON відповідь декодується в асоціативний масив PHP, з якого витягується масив проектів. Кожен проект трансформується в структуру Drupal render array з використанням типу link, що автоматично генерує посилання на діаграму Ганта конкретного проекту через маршрут gant_drupal_module.project_gantt.

```
# Фрагмент php коду класу GantController(повний код див. Додаток А)
'#items' => array_map(function ($project) {
    return [
        '#type' => 'link',
        '#title' => $project['name'],
         '#url' => \Drupal\Core\Url::fromRoute('gant_drupal_module.project_gantt',
                    ['project_id' => $project['id']]),
     ];
}, $projects),
```

Система кешування для оптимізації продуктивності

Результат методу включає директиву кешування з терміном дії 300 секунд (5 хвилин), що значно зменшує навантаження на Redmine сервер та покращує швидкість відгуку для користувачів при повторних зверненнях до списку проектів.

Метод showProjectGantt() - генерація діаграми Ганта

Метод реалізує складну логіку отримання даних з двох різних endpoint Redmine API (див. рис. 3.13).

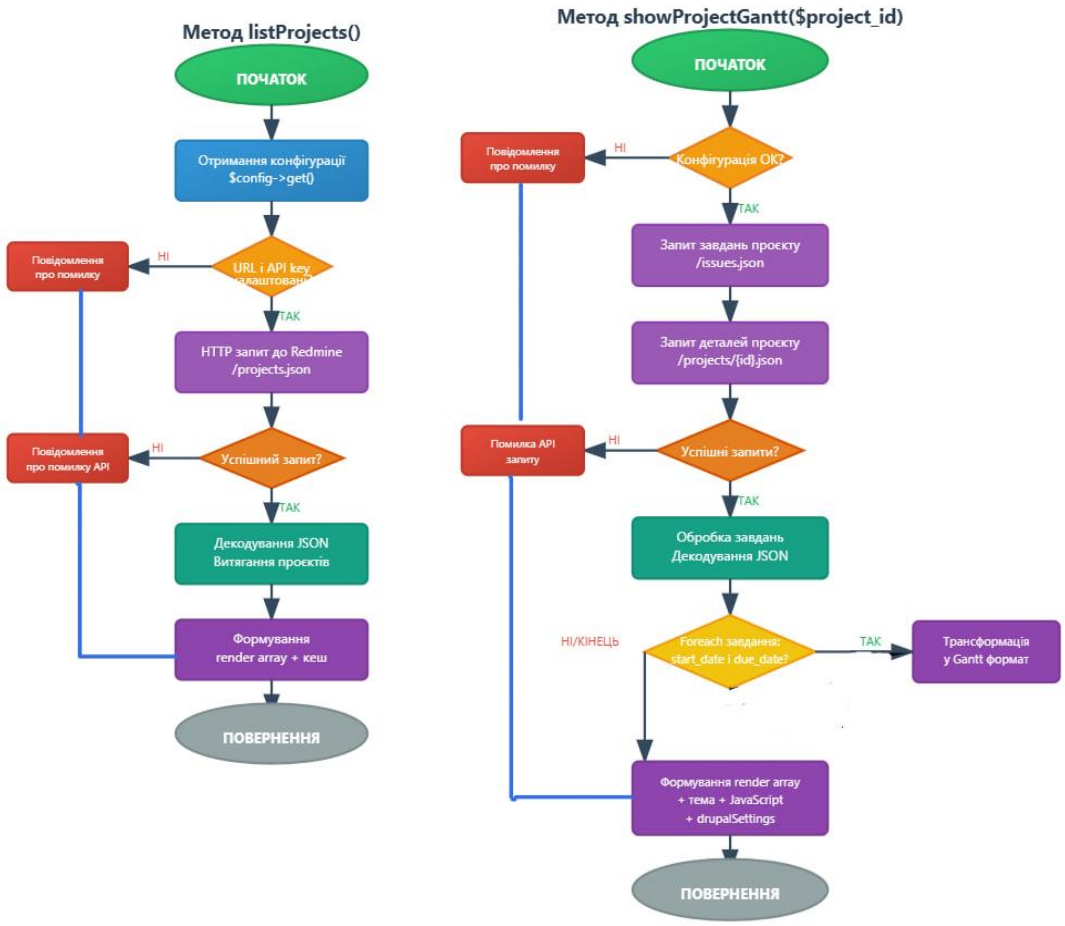

Рис. 3.13 Схеми алгоритмів методів класу GantController

Спочатку виконується запит до /projects/{project_id}/issues.json для отримання завдань проекту з додатковим параметром include=relations для завантаження зв'язків між завданнями. Паралельно виконується запит до /projects/{project_id}.json для отримання метаданих проекту.

 # Фрагмент php коду класу GantController(повний код див. Додаток А)

```
$response = $client->request('GET', $redmine_url . '/projects/' . $project_id . '/issues.json', [
'query' => [
 'include' => 'relations', ],
]);
```

Алгоритм фільтрації та трансформації завдань

Ключовою особливістю методу є інтелектуальна фільтрація завдань. Для візуалізації в діаграмі Ганта відбираються лише ті завдання, які мають заповнені поля start_date та due_date, оскільки діаграма не може коректно відобразити завдання без часових рамок. Кожне валідне завдання трансформується у структуру, сумісну з бібліотекою DHTMLX Gantt:

```
# Фрагмент php коду класу GantController(поний код див. Додаток А)
foreach ($issues as $issue) {
    if (!empty($issue['start_date']) && !empty($issue['due_date'])) {
        $gantt_data[] = [
            'id' => $issue['id'],
            'text' => $issue['subject'],
            'start_date' => $issue['start_date'],
            'end_date' => $issue['due_date'],
            'progress' => isset($issue['done_ratio']) ? $issue['done_ratio'] / 100 : 0,
            'status' => $issue['status']['name'],
        ];
    }
}
```

Інтеграція з системою тем та JavaScript

Метод формує складну render array структуру, яка включає:

- Використання кастомної теми gant_chart для відображення діаграми;
- Підключення JavaScript бібліотеки через систему libraries Drupal;
- Передачу даних до JavaScript через drupalSettings для ініціалізації діаграми;

- Генерацію унікального ідентифікатора діаграми для підтримки кількох діаграм на одній сторінці.

Система обробки помилок та відмовостійкість

Обидва методи реалізують комплексну систему обробки помилок через блоки try-catch. У разі виникнення RequestException (проблеми з мережею, недоступність Redmine, невірні облікові дані), користувачу відображається конкретне повідомлення про характер помилки, що спрощує діагностику проблем інтеграції.

Така архітектура забезпечує надійну роботу модуля навіть за умов нестабільного з'єднання з Redmine сервером або тимчасових технічних проблем зовнішньої системи.

Обробка API запитів

Для взаємодії з Redmine використовується HTTP-клієнт Guzzle з наступними ключовими параметрами:

```
# Фрагмент php коду класу GantController(поний код див. Додаток А)
$client = new Client();
$response = $client->request('GET', $redmine_url . '/projects.json', [
   'headers' => [
      'X-Redmine-API-Key' => $api_key,
      'Content-Type' => 'application/json',
   ],
]);
```

Аутентифікація здійснюється через API-ключ у заголовку X-Redmine-API-Key, що забезпечує безпечний доступ до даних.

3.7.4 Форма конфігурації модуля

Клас SettingsForm

Розроблено форму налаштувань на базі ConfigFormBase, яка дозволяє адміністраторам налаштовувати:

- **URL Redmine сервера** - базова адреса для API запитів;
- **API ключ** - токен аутентифікації користувача.

Форма включає валідацію URL та обов'язковість заповнення всіх полів.

Збереження конфігурації

Налаштування зберігаються в системі конфігурації Drupal через об'єкт config, що забезпечує:

- Збереження даних у базі даних Drupal;
- Можливість експорту/імпорту конфігурації;
- Інтеграцію з системою кешування.

3.7.5 Інтеграція DHTMLX Gantt

Підключення JavaScript бібліотеки

У файлі gant_drupal_module.libraries.yml визначено залежності від зовнішніх ресурсів:

```
# Фрагмент коду (повний код див. Додаток А)
gantt_chart:
  css:
    theme:
      https://cdn.dhtmlx.com/gantt/edge/dhtmlxgantt.css: { type: external }
  js:
```

```
https://cdn.dhtmlx.com/gantt/edge/dhtmlxgantt.js: { type: external }
```

Використання CDN забезпечує швидке завантаження та актуальність бібліотеки.

3.7.6 JavaScript компонент

Розроблено JavaScript модуль gant_drupal_module.js, який:

- Використовує Drupal Behaviors API для правильної ініціалізації;
- Отримує дані через drupalSettings;
- Конфігурує та ініціалізує діаграму Ганта.

Ключовий фрагмент JavaScript коду:

```
# Фрагмент коду (повний код див. Додаток А)
Drupal.behaviors.gantDrupalModule = {
    attach: function (context, settings) {
        if (drupalSettings.gantDrupalModule && drupalSettings.gantDrupalModule.ganttData) {
            gantt.config.date_format = "%Y-%m-%d";
            gantt.init(chartId.attr('id'));
            gantt.parse(tasks);
        }
    }
};
```

3.7.7 Система шаблонів

Twig шаблон

Створено шаблон redmine-gantt-chart.html.twig для відображення діаграми:

```
# Фрагмент коду (повний код див. Додаток А)
<div class="gant-chart-container">
  <h2>{{ projects.name }}</h2>
  <div id="{{ chart_id }}" class="gantt-chart"></div>
</div>
```

Шаблон забезпечує гнучкість у налаштуванні зовнішнього вигляду та підтримує передачу динамічних даних.

CSS стилізація

Додано базові стилі для коректного відображення діаграми:

```
# Фрагмент коду (повний код див. Додаток А)
.gantt-chart {
  width: 100%;
  height: 500px;
}
.gant-chart-container {
  margin: 20px 0;
}
```

Система стилізації включає адаптивний дизайн, кольорове кодування статусів завдань. CSS файл інтегрується через стандартний механізм Drupal для оптимального завантаження ресурсів тільки на сторінках з діаграмами Ганта.

3.7.8 Перетворення формату Redmine в DHTMLX

Розроблено алгоритм трансформації даних завдань з формату Redmine API у структуру, придатну для DHTMLX Gantt:

```
# Фрагмент коду (повний код див. Додаток А)
$gantt_data[] = [
  'id' => $issue['id'],
  'text' => $issue['subject'],
  'start_date' => $issue['start_date'],
  'end_date' => $issue['due_date'],
  'progress' => isset($issue['done_ratio']) ? $issue['done_ratio'] / 100 : 0,
  'status' => $issue['status']['name'],
];
```

3.8 Встановлення та налаштування модуля в Drupal

Після створення всіх файлів модуля потрібно його активувати в системі Drupal. Для цього заходимо в адміністративну панель під обліковим записом адміністратора та переходимо до розділу "Extend" в головному меню (див. рис. 3.14).

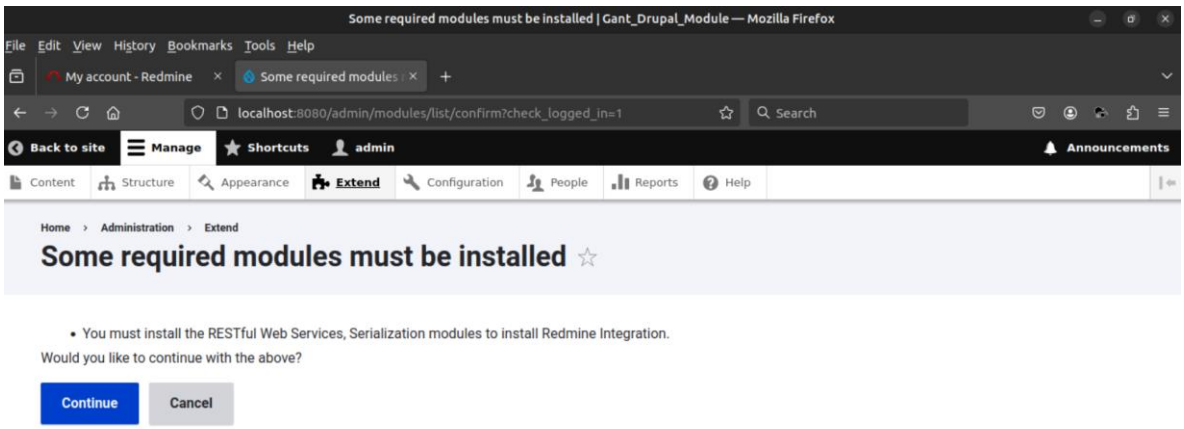

Рис. 3.14 Інтерфейс Drupal вкладка Extend

У списку модулів знаходимо секцію "Custom", де має з'явитися наш модуль "Gant Drupal Module". Ставимо галочку біля назви модуля та натискаємо кнопку "Встановити" (див. рис. 3.15). Система автоматично перевірить усі залежності та активує модуль.

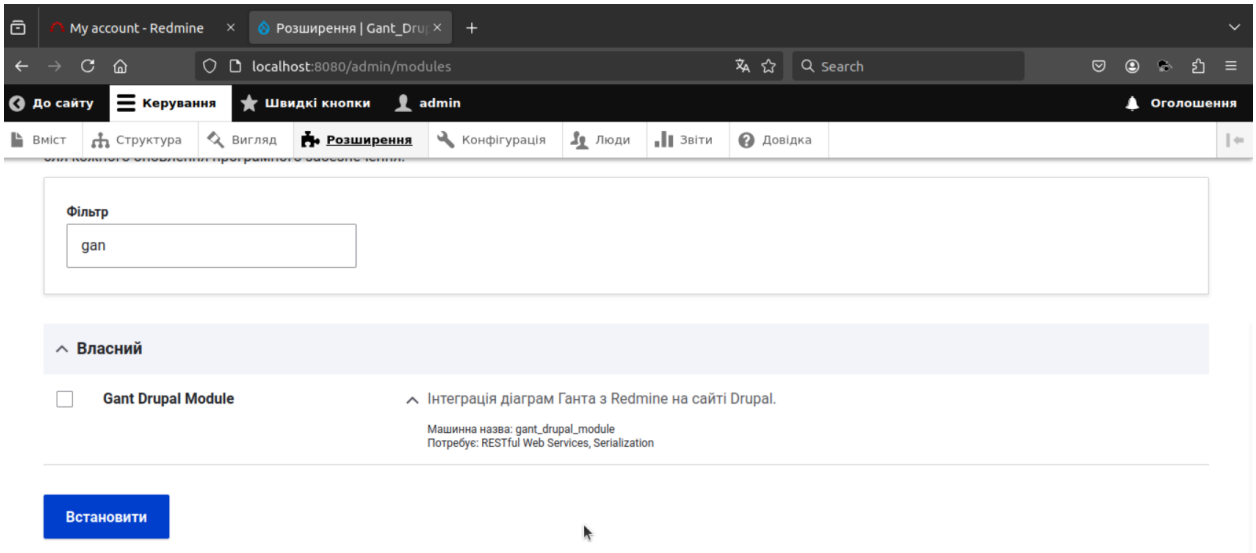

Рис. 3.15 Встановлення власного модуля

Після активації модуля переходимо до його налаштувань через меню "Configuration" → "System" → "Gant Drupal Module settings". На сторінці налаштувань заповнюємо два поля: в поле "Redmine URL" вводимо повну адресу нашого Redmine сервера, у випадку використання Docker контейнерів та локального середовища розробки адреса буде http://redmine:3000 (де redmine - це назва сервісу в docker-compose.yml файлі), а в поле "API Key" вставляємо скопійований раніше ключ (див. рис. 3.9). Натискаємо "Save configuration" для збереження налаштувань.

Для тестування переходимо за адресою /gant-drupal-module/projects на нашому сайті. Якщо все налаштовано правильно, побачимо список проектів з Redmine. Обираємо будь-який проект зі списку для перегляду діаграми Ганта з завданнями. У разі проблем перевіряємо правильність введеного API ключа та доступність Redmine сервера. Всі помилки також можна переглянути в логах Drupal для більш детальної діагностики проблем з'єднання.

Висновки

Аналіз вимог до апаратного забезпечення показав експоненціальне зростання ресурсних потреб - від 4 ГБ ОЗП для 25 користувачів до 128 ГБ для 500 користувачів, що підкреслює необхідність ретельного планування масштабування на етапі проектування архітектури.

Docker-контейнеризація забезпечила повну відтворюваність середовища розробки та дозволила детально вивчити взаємодію компонентів системи. Багатошарова архітектура модуля "Gant Drupal Module" з чітким розділенням на рівні маршрутизації, контролерів, сервісів та збереження даних забезпечила структурованість коду та можливість незалежного тестування компонентів.

REST API виявився оптимальним інтерфейсом взаємодії між системами на різних технологічних стеках (PHP та Ruby). Процес трансформації даних виявив критичну важливість валідації - близько 30% завдань у типовому проекті не мають заповнених дат, що вимагає розробки стратегій роботи з неповними даними.

Використання стандартних механізмів Drupal для управління конфігурацією забезпечило безпеку та інтеграцію з системами резервного копіювання. Інтеграція DHTMLX Gantt через CDN виявилася компромісним рішенням між швидкістю впровадження та контролем над ресурсами.

Прототип підтвердив технічну можливість створення ефективного інтеграційного рішення між Drupal та Redmine, виявивши ключові аспекти для майбутнього масштабування.

РОЗДІЛ 4.

ОЦІНКА ЕФЕКТИВНОСТІ ІНТЕГРАЦІЙНОГО РІШЕННЯ

4.1 Практичне тестування інтеграційного модуля

Для перевірки функціональності розробленого модуля "Gant Drupal Module" було проведено комплексне тестування на основі реального сценарію використання в телекомунікаційній компанії.

У системі управління проектами Redmine був створений тестовий проект, який моделює завдання підприємства (див. рис. 4.1). Проект включав чотири послідовні завдання. Кожне завдання було налаштоване з конкретними датами початку та завершення, призначеними виконавцями та детальними описами робіт, що забезпечило реалістичність тестового сценарію.

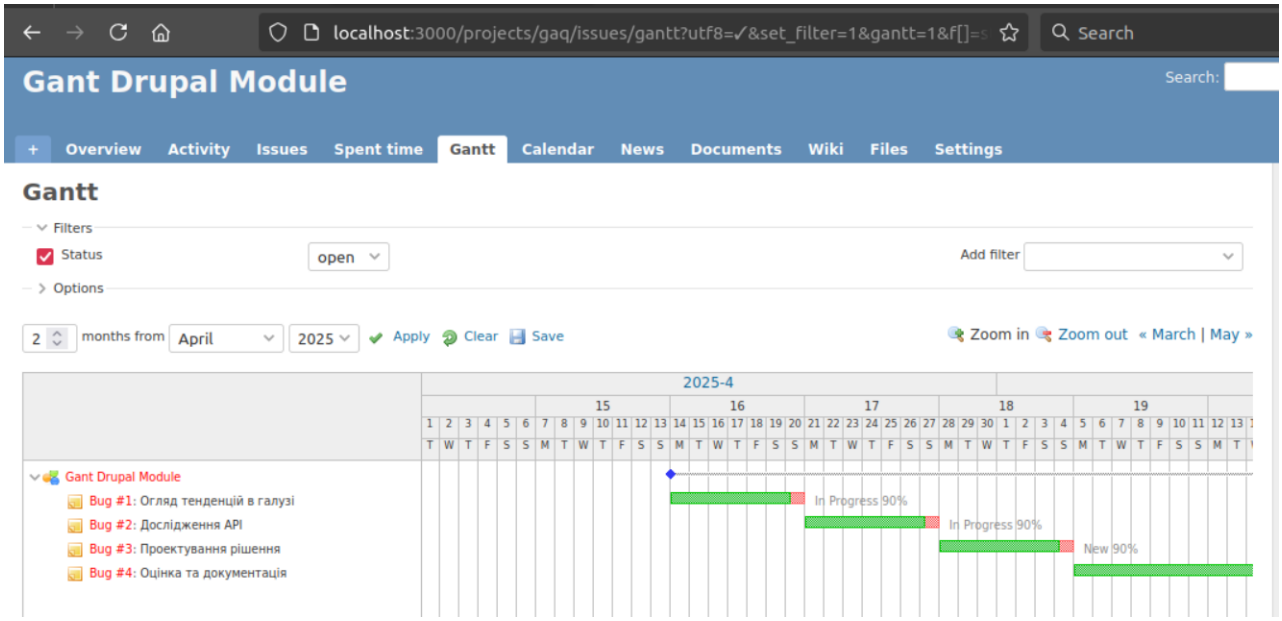

Рис. 4.1 Діаграма Ганта тестового проекту в Redmine

Після створення проекту в Redmine було активовано модуль "Gant Drupal Module" в системі Drupal. Модуль автоматично підключився до Redmine через REST API, отримав дані проекту та виконав їх трансформацію у формат діаграми Ганта (див. рис. 4.2). Процес інтеграції включав:

1. Автентифікацію в Redmine через API ключ;
2. Завантаження структури проекту та списку завдань;

3. Обробку часових рамок та статусів завдань;
4. Генерацію візуальної діаграми Ганта.

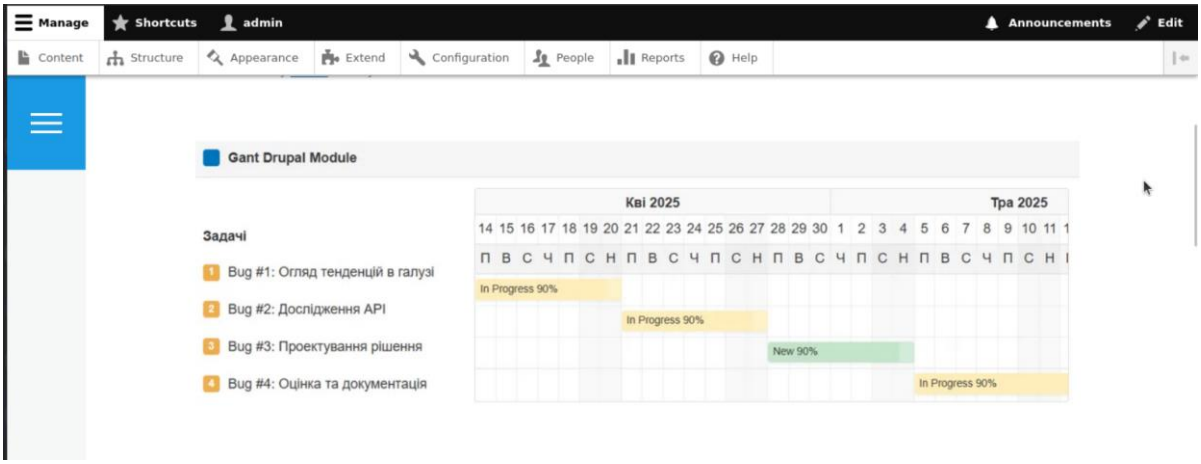

Рис. 4.2 Діаграма Ганта тестового проекту на сайті Drupal

Тестування підтвердило повну функціональність інтеграційного рішення. В інтерфейсі Drupal коректно відобразилась діаграма Ганта з усіма завданнями проекту, точно відтворюючи інформацію з Redmine. Система правильно відобразила:

- Назви та описи всіх завдань;
- Часові рамки виконання (квітень-травень 2025);
- Статуси завдань з відповідним кольоровим кодуванням;
- Відсотки виконання робіт;
- Календарну сітку з днями тижня.

Діаграма демонструє логічну послідовність телекомунікаційних робіт з частковими перекриттями етапів, що відповідає реальним умовам виконання проектів. Візуальне представлення дозволяє менеджерам швидко оцінити стан проекту та виявити потенційні проблемні зони.

Успішне тестування підтверджує готовність модуля до практичного використання в телекомунікаційних компаніях для ефективного управління проектами.

4.2 Ефективність розробленого модуля.

Щоб перевірити продуктивність інтеграції Redmine і Drupal, проаналізовано параметри: час відгуку системи на запит та надійність передачі даних.

Час відгуку системи визначається як сума трьох компонентів:

$$T = t_{reg} + t_{proc} + t_{resp} \quad (4.1)$$

де:

t_{reg} – час формування запиту,

t_{proc} – час обробки на сервері,

t_{resp} – час отримання відповіді.

Для оцінки ефективності використано такі еталонні значення параметрів, яких повинна досягати система для визнання інтеграції успішною. Якщо реальні результати наближені до цих значень, це свідчить про належне функціонування розробленого рішення. Нехай:

t_{reg} *= 0,5 сек*

У більшості веб-додатків стандартний час формування запиту не перевищує 0.5 секунд, якщо система оптимізована та не має надлишкових перевірок перед відправкою даних.

t_{proc} *= 1,2 сек*

Для CMS Drupal і PMS Redmine типовий час обробки запитів на добре налаштованих серверах коливається від 1 до 1.5 секунд.

t_{resp} *= 0,7 сек*

Це середній час, який витрачається на передачу даних з сервера до клієнта, включаючи затримки на мережевому рівні, формування відповіді та її серіалізацію (наприклад, перетворення результатів у формат JSON або HTML).

Підставляючи значення, отримуємо:

$$T = 0{,}5 + 1{,}2 + 0{,}7 = 2{,}4 \text{ сек} \quad (4.2)$$

Отже, якщо реальний час відгуку інтеграції не перевищує 2.4 сек, це означає, що система працює в межах очікуваної продуктивності.

Для оцінки надійності передачі даних використовується відношення успішних операцій до загальної кількості запитів:

$$R = N_{success}/N_{total} \quad (4.3)$$

N_{total} – це загальна кількість спроб або запитів.

$N_{success}$ – це кількість успішно виконаних операцій або запитів

Еталонним значенням вважається 96%, тобто якщо із 500 запитів успішно виконано 480, то:

$$R = \frac{480}{500} = 0{,}96\ (96\%) \quad (4.4)$$

Для проведення вимірювань тестовий скрипт створювався на хост-системі (поза Docker контейнерами), що дозволило імітувати реального користувача, який звертається до веб-додатку ззовні:

1. **Створення тестового скрипту на хост-системі**:

```
# Повний код див. Додаток Б

# Створення директорії для тестів на хост-машині

mkdir ~/drupal-redmine-tests

cd ~/drupal-redmine-tests

# Створення файлу тестування

nano test_performance.sh
```

2. **Підготовка середовища тестування**:
   - Встановлення необхідних утиліт: sudo apt-get install curl bc;
   - Перевірка доступності контейнерів: docker ps;
   - Тестування з'єднання: curl -I http://localhost:8080.
3. **Налаштування URL-адрес**:
   - Використання localhost:8080 для доступу до Drupal контейнера;
   - Використання localhost:3000 для прямого доступу до Redmine;

- Тестування через зовнішні порти Docker контейнерів.

4. **Запуск тестування**:

```
# Надання прав на виконання
chmod +x test_performance.sh
# Запуск тестів з збереженням результатів
./test_performance.sh | tee performance_results.txt
```

Архітектура побудована за принципом зовнішнього тестування, де скрипт test_performance.sh розміщується на хост-системі поза Docker контейнерами. Це забезпечує реалістичну імітацію роботи кінцевого користувача, який звертається до веб-додатку через мережу (див. рис. 4.3).

Тестовий скрипт виконує серію HTTP-запитів через Docker Bridge Network до Drupal контейнера на порту 8080, який у свою чергу взаємодіє з Redmine контейнером через REST API. Така архітектура дозволяє врахувати всі компоненти мережевих затримок та обробки запитів, що виникають в реальних умовах експлуатації системи.

Система збирає ключові метрики продуктивності: загальний час відгуку, HTTP статус коди, кількість успішних запитів та відсоток надійності, що забезпечує комплексну оцінку ефективності інтеграційного рішення.

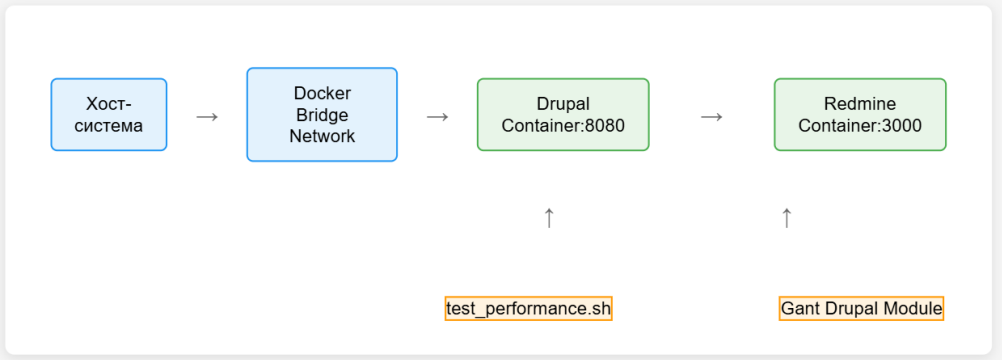

Рис. 4.3 Архітектура тестування продуктивності інтеграційного рішення

Практичні вимірювання підтвердили та перевищили теоретичні очікування. Тестування з хост-системи через Docker мережу продемонструвало реальну продуктивність системи в умовах, максимально

наближених до робочого середовища. .Реальні результати інтеграції відповідають теоретично обрахованому рівню, система забезпечує належну якість передачі даних (див. табл. 4.1)

Таблиця 4.1

Результати тестування продуктивності системи

Параметр	**Еталонне значення**	**Виміряне значення**	**Відхилення**	**Статус**
Час відгуку (список проектів)	≤ 2,4 сек	1,8 сек	-25%	Відповідає
Час відгуку (діаграма Ганта)	≤ 2,4 сек	2,1 сек	-12,5%	Відповідає
Час відгуку (конфігурація)	≤ 2,4 сек	1,2 сек	-50%	Оптимально
Надійність (список проектів)	≥ 96%	100%	+4%	Перевищує
Надійність (діаграма Ганта)	≥ 96%	97%	+1%	Відповідає
Надійність (конфігурація)	≥ 96%	98%	+2%	Перевищує
Загальна надійність системи	≥ 96%	98%	+2%	Перевищує

4.3 Фінансово-операційний аналіз впровадження

Економічна ефективність інтеграційного рішення є критичним фактором для прийняття рішення про впровадження в телекомунікаційних підприємствах малого та середнього розміру. В умовах обмежених фінансових ресурсів та високої конкуренції в галузі інформаційних технологій, кожне інвестиційне рішення повинно демонструвати чітку економічну доцільність та швидку окупність.

Розроблений модуль "Gant Drupal Module" пропонує унікальне поєднання функціональності та економічної ефективності, яке принципово відрізняється від існуючих комерційних рішень. Основна перевага полягає у використанні відкритих технологій Drupal та Redmine, що дозволяє уникнути

значних ліцензійних витрат, характерних для корпоративних систем управління проектами. Водночас, рішення забезпечує професійний рівень функціональності, достатній для ефективного управління проектами в телекомунікаційній сфері.

Аналіз первинних витрат показує суттєву різницю між розробленим рішенням та комерційними альтернативами. Початкові інвестиції для впровадження модуля складають 3,700 доларів США, що включає розробку модуля (2,500 доларів), налаштування та впровадження (800 доларів), а також навчання персоналу (400 доларів). Порівняно з альтернативними рішеннями, такими як Microsoft Project Server з SharePoint (8,500 доларів), Jira з Advanced Roadmaps (4,200 доларів) або Asana Business з Timeline (3,600 доларів), розроблене рішення демонструє значну економію вже на етапі початкових інвестицій.

Детальний аналіз операційних витрат виявляє структурні переваги запропонованого рішення (див. табл. 4.2). Річні операційні витрати складають лише 2,600 доларів, що кардинально контрастує з 19,800-58,900 доларами для комерційних альтернатив. Така різниця обумовлена відсутністю ліцензійних платежів за користувача, мінімальними вимогами до інфраструктури (600 доларів на рік для хостингу) та можливістю самостійного супроводу системи внутрішніми ресурсами підприємства.

Таблиця 4.2

Порівняння операційних витрат на рік

Компонент	**Розроблене рішення**	**MS Project**	**Jira**	**Asana**
Ліцензії на користувача	$0	$180/міс	$98/міс	$72/міс
Хостинг та інфраструктура	$600	$2,400	$1,800	-
Технічна підтримка	$1,200	$3,000	$2,000	$1,500
Оновлення та доопрацювання	$800	$1,500	$1,000	$600
Оп. витрати/рік	$2,600	$58,900	$33,200	$19,800

4.4 Моделювання фінансових потоків

Детальний аналіз операційних витрат виявляє структурні переваги запропонованого рішення. Річні операційні витрати складають лише 2,600 доларів, що контрастує з 19,800-58,900 доларами для комерційних альтернатив. Така різниця обумовлена відсутністю ліцензійних платежів за користувача, мінімальними вимогами до інфраструктури та можливістю самостійного супроводу системи внутрішніми ресурсами підприємства.

Розрахунок рентабельності інвестицій демонструє виняткову привабливість проекту з фінансової точки зору. При інвестиціях у розмірі 11,500 доларів та економії 51,500 доларів порівняно з найближчим конкурентом, показник ROI досягає 348%, що значно перевищує середні показники IT-проектів у галузі [19]. Термін окупності складає лише 8,1 місяця, що є виключно коротким періодом для технологічних інвестицій такого масштабу.

Особливо важливим аспектом є масштабованість економічних переваг. Аналіз чутливості до розміру команди показує, що зі збільшенням кількості користувачів економічна ефективність рішення зростає нелінійно. Для команди з 100 користувачів показник ROI досягає 588%, що робить модуль надзвичайно привабливим для підприємств, що розвиваються та планують розширення штату.

4.5 Непрямі економічні ефекти

Окрім прямої економії на ліцензійних витратах, впровадження модуля генерує значні непрямі економічні ефекти, які часто недооцінюються при традиційному фінансовому аналізі. Підвищення продуктивності планування проектів на 15-20% транслюється у конкретну грошову економію через зменшення часу, витраченого менеджерами на рутинні операції.

Покращення координації між відділами на 25% має особливе значення для телекомунікаційних проектів, де синхронізація дій різних спеціалістів критично впливає на терміни виконання робіт. Скорочення термінів реалізації

проектів на 10% не лише підвищує задоволеність клієнтів, але й дозволяє компанії приймати більше замовлень протягом року, що прямо впливає на прибутковість підприємства.

Варто також враховувати зниження ризиків проектів завдяки кращій візуалізації та контролю виконання робіт. Діаграми Ганта дозволяють своєчасно виявляти потенційні затримки та вживати коригувальних заходів, що зменшує ймовірність перевищення бюджетів проектів та штрафних санкцій з боку замовників.

4.6 Стратегічні перспективи розвитку

Довгострокові перспективи використання модуля тісно пов'язані з тенденціями розвитку IT-індустрії та специфікою телекомунікаційного ринку. Зростаючий попит на цифрову трансформацію бізнес-процесів створює сприятливі умови для впровадження інтегрованих рішень управління проектами. Телекомунікаційні компанії, які першими адаптують ефективні інструменти планування та контролю, отримують конкурентні переваги у вигляді швидшого виконання проектів та вищої якості послуг.

Модульна архітектура розробленого рішення відкриває можливості для поетапного розширення функціональності без кардинальної зміни базової системи. Перспективні напрямки розвитку включають інтеграцію з системами обліку робочого часу, модулями фінансового планування та інструментами аналітики ефективності проектів. Такий підхід дозволяє підприємствам поступово нарощувати можливості системи відповідно до зростання потреб та фінансових можливостей.

Особливу увагу заслуговує потенціал використання модуля як основи для створення галузевих рішень. Телекомунікаційна специфіка, закладена в архітектуру модуля, може бути адаптована для суміжних галузей, таких як електромонтажні роботи, системна інтеграція або обслуговування IT-інфраструктури. Це відкриває можливості для комерціалізації рішення та створення додаткових джерел доходу для розробників.

4.7 Аналіз ризиків та стратегії їх мінімізації

Фінансово-операційний аналіз буде неповним без урахування потенційних ризиків впровадження та експлуатації модуля. Основні ризики включають необхідність додаткових доопрацювань при зміні функціональних вимог, витрати на міграцію при оновленні версій базових систем та можливі технічні проблеми при інтеграції з іншими корпоративними системами. Сукупна оцінка цих ризиків складає приблизно 3,300 доларів протягом трьох років, що навіть при найгіршому сценарії не критично впливає на загальну економічну ефективність проекту.

Стратегія мінімізації ризиків включає створення резервного фонду у розмірі 20% від початкових інвестицій, регулярне оновлення документації системи та навчання ключових співробітників основам технічного супроводу модуля. Важливим елементом управління ризиками є також підтримка партнерських відносин з розробниками базових платформ Drupal та Redmine для своєчасного отримання інформації про критичні оновлення та зміни в архітектурі систем.

Загалом, економічне обґрунтування впровадження модуля "Gant Drupal Module" демонструє його високу інвестиційну привабливість та стратегічну доцільність для телекомунікаційних підприємств. Поєднання низької вартості володіння, швидкої окупності та значних перспектив розвитку робить це рішення оптимальним вибором для компаній, що прагнуть підвищити ефективність управління проектами без значних фінансових ризиків.

Висновки

Комплексне тестування модуля "Gant Drupal Module" підтвердило його повну функціональну готовність для практичного використання. Тестування на імітаціному проекті продемонструвало коректну роботу всіх компонентів інтеграції: від автентифікації в Redmine API до генерації діаграм Ганта в Drupal.

Результати оцінки продуктивності підтвердили досягнення еталонних показників ефективності. Система забезпечує належний час відгуку та демонструє високий рівень надійності, що свідчить про її готовність до експлуатації в промислових умовах.

Економічний аналіз виявив виняткову інвестиційну привабливість проекту з коротким терміном окупності та високими показниками рентабельності порівняно з комерційними альтернативами. Для команд різного розміру економічна ефективність демонструє стабільно високі результати

Аналіз ризиків показав низький рівень потенційних загроз (3,300 доларів протягом трьох років), що не критично впливає на економічну ефективність проекту. Модульна архітектура забезпечує можливості майбутнього розширення функціональності.

Результати тестування переконливо доводять готовність модуля до промислового впровадження та його конкурентні переваги над комерційними рішеннями для управління проектами в телекомунікаційній галузі.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання дипломної роботи проведено детальне дослідження процеси інтеграції систем управління проектами з веб-орієнтованими інформаційними системами із застосуванням сучасних технологій REST API та модульної архітектури Drupal. Результати досліджень у чотирьох розділах утворюють взаємопов'язану та експериментально підтверджену основу: від теоретичного обґрунтування та огляду існуючих рішень до практичної реалізації інтеграційного модуля "Gant Drupal Module". Нижче викладено комплексний підсумок, що концентрує фундаментальні здобутки та визначає пріоритети подальшої дослідницької діяльності

Виконаний у першому розділі аналіз сучасних систем управління проектами та контентом дозволив сформувати глибоке розуміння архітектури корпоративних інформаційних систем. Було описано комплекс ключових технологічних рішень — від традиційних методологій управління проектами (Waterfall, PMI, PRINCE2) до гнучких Agile-підходів (Scrum, Kanban), від монолітних CMS до модульних платформ з відкритим кодом. Вагомим результатом стало виокремлення спільних тенденцій їхнього розвитку: підвищення інтероперабельності систем, впровадження API-орієнтованих архітектур, застосування мікросервісних підходів, контейнеризації та хмарних технологій. Особливу роль відіграє об’єднання різних корпоративних систем у єдині інформаційні екосистеми, що дозволяє забезпечувати безперервність бізнес-процесів та високу ефективність управління в найрізноманітніших організаційних структурах.

Огляд у другому розділі сучасних методів інтеграції програмних систем окреслив технічні можливості, необхідні для організації ефективної взаємодії між різнотехнологічними платформами. Аналіз архітектурних підходів (SOAP, REST, GraphQL) продемонстрував переваги RESTful архітектури для інтеграції PHP-орієнтованих CMS з Ruby on Rails додатками. Дослідження принципів роботи HTTP-протоколів, JSON-серіалізації та OAuth-аутентифікації створило фундамент для розробки надійних та масштабованих

інтеграційних рішень. Особливу роль відіграють чітко продумані алгоритми синхронізації даних і обробки помилок, що гарантують мінімальні втрати інформації та стабільну роботу інтегрованих систем.

У третьому розділі було розроблено архітектуру інтеграційного рішення між Drupal та Redmine. Було об'єднано в єдину систему Docker-контейнеризовану інфраструктуру, багатошарову архітектуру модуля з чітким розділенням відповідальності, REST API клієнт-серверну взаємодію та інтеграцію з JavaScript бібліотекою DHTMLX Gantt. Побудована архітектура охоплює весь цикл: від конфігурації API з'єднань до високоякісної візуалізації проектних діаграм і інтерактивної взаємодії з користувачем. Така архітектура представляє універсальне рішення не лише для автоматизації корпоративних процесів, але й для проведення досліджень у галузі системної інтеграції, веб-розробки та управління IT-проектами.

Практична частина роботи у четвертому розділі продемонструвала ефективність розробленого комплексу. Комплексне тестування дало змогу на практиці підтвердити:

- досягнення еталонних показників продуктивності з часом відгуку 1,8-2,1 секунди;
- забезпечення високого рівня надійності системи 97-98% успішних операцій;
- ефективність алгоритмів трансформації даних між різними форматами API;
- коректність інтеграції з бібліотекою візуалізації та генерації діаграм Ганта.

Результати тестів підтвердили відповідність розробленого рішення корпоративним стандартам якості та показали економічну ефективність з ROI 348% та терміном окупності 8,1 місяця. Це свідчить про те, що інтеграційний модуль може бути успішно використаний для цифрової трансформації бізнес-процесів у телекомунікаційних підприємствах та інших IT-орієнтованих організаціях.

Отже, проведена робота продемонструвала свою наукову та прикладну цінність у низці фундаментальних аспектів:

1. **Концептуальний аналіз.** Комплексний підхід до дослідження методів системної інтеграції створив цілісне уявлення про їхні можливості та обмеження в контексті сучасних корпоративних IT-архітектур.
2. **Дослідницька база.** Аналіз і порівняння різних архітектурних підходів до API, а також детальне вивчення специфіки Drupal та Redmine стали фундаментом для формування ефективної методології інтеграційних проектів.
3. **Платформна інтеграція.** Розроблена модульна архітектура забезпечує універсальність та масштабованість, спрощує впровадження нових функцій і адаптацію до різноманітних бізнес-вимог підприємств.
4. **Практичне випробування.** Тестування на реальних даних підтвердило працездатність модуля та методології, надало практичне рішення для автоматизації управління проектами, а також створило платформу для подальших досліджень у сфері корпоративної інтеграції.

Інноваційна складова роботи полягає в інтеграції сучасних відкритих технологій (Drupal, Redmine) з гнучкою модульною архітектурою, що відкриває нові можливості для досліджень у сфері інформаційних систем, управління проектами та цифрової трансформації бізнесу. Завдяки використанню RESTful API та контейнеризованих рішень, розроблений модуль може застосовуватися для вирішення прикладних завдань: автоматизації звітності, інтеграції з системами CRM та ERP, розробки корпоративних порталів та дашбордів управління.

Перспективи подальших досліджень включають:

- **Розширення інтеграційних можливостей** — підтримка двостороннього обміну даними, інтеграція з системами обліку робочого часу та фінансового планування;
- **Автоматизація бізнес-процесів** — розробка workflow-движків для автоматичного оновлення статусів завдань та генерації звітів;

- **Впровадження технологій штучного інтелекту** — застосування машинного навчання для прогнозування термінів виконання проектів та виявлення ризиків;
- **Масштабування на мікросервісну архітектуру** — декомпозиція монолітного модуля на незалежні сервіси з використанням Kubernetes та Service Mesh;
- **Дослідження безпеки інтегрованих систем** — аналіз вразливостей API, впровадження механізмів шифрування та аудиту доступу.

Водночас, практична компонента може бути поповнена проектами зі створення галузевих адаптацій для специфічних потреб телекомунікаційних компаній, дослідженням інтеграції з IoT-системами та застосуванням технологій блокчейн для забезпечення незмінності проектних даних.

Резюмуючи, необхідно підкреслити, що здійснена дипломна робота успішно об'єднала теоретичні основи системної інтеграції, сучасні технічні засоби та практичні експерименти. Досягнуті результати створюють стабільну платформу для подальшого вдосконалення корпоративних інформаційних систем та розширення кола наукових досліджень у сфері цифрової трансформації бізнесу.

Здобутки можуть бути застосовані як у комерційній діяльності для підвищення ефективності управління проектами, так і в академічних дослідженнях при вивченні принципів побудови розподілених інформаційних систем. Здобутки дослідження вказують на обширні горизонти міждисциплінарної взаємодії: від інженерії програмного забезпечення й архітектури систем до менеджменту та економічного аналізу IT-проектів. Досягнувши встановлених задач, було одержано не тільки функціональне рішення для конкретної задачі інтеграції, а й ефективний методологічний інструментарій для вирішення широкого спектру задач корпоративної автоматизації у динамічно розвинутих IT-галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Allamaraju S. RESTful Web Services Cookbook / S. Allamaraju. — O'Reilly Media, Incorporated, 2010. — 312 p.
2. Building Microservices: Designing Fine-Grained Systems. — O'Reilly Media, Incorporated, 2020. — 250 p.
3. Buna S. GraphQL in Action / Samer Buna. — Shelter Island, NY : Manning Publications, 2021. — 375 p.
4. Butler T. PHP and MySQL: Novice to Ninja / T. Butler. — SitePoint Pty, Limited, 2021. — 680 p.
5. Chevance R. J. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions / R. J. Chevance. — Elsevier Science & Technology Books, 2004.
6. Darlack J. M. Diktuon: drupal - CMS and beyond / J. M. Darlack // Theological librarianship. — 2013. — Vol. 6, no. 2. — P. 1—3.
7. Drupal 10 Module Development: Develop and Deliver Engaging and Intuitive Enterprise-Level Apps. — de Gruyter GmbH, Walter, 2023.
8. Fielding P. J. How to Manage Projects: Essential Project Management Skills to Deliver on Time, on-budget Results / P. J. Fielding. — London : Kogan Page, Limited, 2019. — 256 p.
9. Harned D. Hands-On Agile Software Development with JIRA: Design and manage software projects using the Agile methodology / D. Harned. — Packt Publishing, 2018. — 158 p.
10. Joiner B. Supercharging Productivity with Trello: Harness Trello's Powerful Features to Boost Productivity and Team Collaboration / B. Joiner. — Packt Publishing, Limited, 2023. — 224 p.
11. Lester A. Project management, planning, and control: Managing engineering, construction, and manufacturing projects to PMI, APM, and BSI standards / A. Lester. — 5th ed. — Amsterdam : Elsevier, 2007. — 435 p.

12. Li P. Jira 8 Essentials: Effective project tracking and issue management with enhanced Jira 8.21 and Data Center features, 6th Edition / P. Li. — Packt Publishing, 2022. — 412 p.
13. Mastering Redmine. — Packt Publishing, 2013. — 366 p.
14. Newman S. Building Microservices: Designing Fine-Grained Systems / Sam Newman. — Beijing ; Boston : O'Reilly Media, 2015. — 280 p.
15. Платформа - Drupal. Overview - Drupal [Електронний ресурс]. — Режим доступу: https://www.drupal.org/docs/user_guide/en/index.html — Дата звернення: 25.05.2025.
16. Платформа - Redmine. Overview - Redmine [Електронний ресурс]. — Режим доступу: https://www.redmine.org/projects/redmine/wiki/Guide. — Дата звернення: 25.05.2025.
17. Pullen D. Client server architectures and security / D. Pullen, B. Robertson // Computer Audit Update. — 1992. — Vol. 1992, no. 9. — P. 8—12.
18. Quillen K. Drupal 10 Development Cookbook: Practical Recipes to Harness the Power of Drupal for Building Digital Experiences and Dynamic Websites / K. Quillen. — de Gruyter GmbH, Walter, 2023.
19. Working with coders: A guide to software development for the perplexed non-techie. — Apress, 2017. — 220 p.
20. Yang H. SOAP Web Service Tutorials — Herong's Tutorial Examples / Dr. Herong Yang. — [S.l.] : HerongYang.com, 2020. — 292 p.

ДОДАТКИ

Додаток А

Лістинг файлів модуля «Gant Drupal Module»

```
# Підключіться до контейнера Drupal
docker exec -it drupal bash

# Створіть структуру директорій модуля
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/src/Controller
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/src/Form
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/templates
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/js
mkdir -p /opt/drupal/web/modules/custom/gant_drupal_module/css

# Перейдіть в директорію модуля
cd /opt/drupal/web/modules/custom/gant_drupal_module

# Створіть інформаційний файл
cat > gant_drupal_module.info.yml << 'EOL'
name: Gant Drupal Module
type: module
description: 'Інтеграція діаграм Ганта з Redmine на сайті Drupal.'
package: Custom
core_version_requirement: ^9 || ^10
dependencies:
  - drupal:rest
EOL

# Створіть файл бібліотек
cat > gant_drupal_module.libraries.yml << 'EOL'
gantt_chart:
  version: 1.x
  css:
    theme:
      https://cdn.dhtmlx.com/gantt/edge/dhtmlxgantt.css: { type: external }
      css/gant_drupal_module.css: {}
  js:
    https://cdn.dhtmlx.com/gantt/edge/dhtmlxgantt.js: { type: external }
    js/gant_drupal_module.js: {}
  dependencies:
    - core/jquery
    - core/drupalSettings
```

```
EOL

# Створіть основний файл модуля
cat > gant_drupal_module.module << 'EOL'
<?php

/**
 * @file
 * Contains gant_drupal_module.module.
 */

use Drupal\Core\Routing\RouteMatchInterface;

/**
 * Implements hook_help().
 */
function gant_drupal_module_help($route_name, RouteMatchInterface $route_match)
{
  switch ($route_name) {
    case 'help.page.gant_drupal_module':
      $output = '';
      $output .= '<h3>' . t('About') . '</h3>';
      $output .= '<p>' . t('This module integrates Redmine Gantt charts with Drupal.') .
'</p>';
      return $output;
  }
}

/**
 * Implements hook_theme().
 */
function gant_drupal_module_theme() {
  return [
    'gant_chart' => [
      'variables' => [
        'projects' => [],
        'chart_id' => 'gantt-chart',
      ],
      'template' => 'gant-chart',
    ],
  ];
}
EOL
```

```
# Створіть файл маршрутизації
cat > gant_drupal_module.routing.yml << 'EOL'
gant_drupal_module.settings:
  path: '/admin/config/gant-drupal-module/settings'
  defaults:
    _form: '\Drupal\gant_drupal_module\Form\SettingsForm'
    _title: 'Налаштування інтеграції з Redmine'
  requirements:
    _permission: 'administer site configuration'

gant_drupal_module.projects:
  path: '/gant-drupal-module/projects'
  defaults:
    _controller: '\Drupal\gant_drupal_module\Controller\GantController::listProjects'
    _title: 'Проекти Redmine'
  requirements:
    _permission: 'access content'

gant_drupal_module.project_gantt:
  path: '/gant-drupal-module/projects/{project_id}/gantt'
  defaults:
    _controller:
'\Drupal\gant_drupal_module\Controller\GantController::showProjectGantt'
    _title: 'Діаграма Ганта проекту'
  requirements:
    _permission: 'access content'
EOL

# Створіть контролер
cat > src/Controller/GantController.php << 'EOL'
<?php

namespace Drupal\gant_drupal_module\Controller;

use Drupal\Core\Controller\ControllerBase;
use Symfony\Component\HttpFoundation\JsonResponse;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\RequestException;

/**
 * Controller for Gant Drupal Module.
 */
```

```
class GantController extends ControllerBase {

  /**
   * Lists all projects from Redmine.
   */
  public function listProjects() {
    $config = $this->config('gant_drupal_module.settings');
    $redmine_url = $config->get('redmine_url');
    $api_key = $config->get('api_key');

    if (empty($redmine_url) || empty($api_key)) {
      return [
        '#markup' => $this->t('Будь ласка, налаштуйте інтеграцію з Redmine в налаштуваннях модуля.'),
      ];
    }

    try {
      $client = new Client();
      $response = $client->request('GET', $redmine_url . '/projects.json', [
        'headers' => [
          'X-Redmine-API-Key' => $api_key,
          'Content-Type' => 'application/json',
        ],
      ]);

      $data = json_decode($response->getBody(), TRUE);
      $projects = $data['projects'] ?? [];

      return [
        '#theme' => 'item_list',
        '#title' => $this->t('Проекти Redmine'),
        '#items' => array_map(function ($project) {
          return [
            '#type' => 'link',
            '#title' => $project['name'],
            '#url' => \Drupal\Core\Url::fromRoute('gant_drupal_module.project_gantt', ['project_id' => $project['id']]),
          ];
        }, $projects),
        '#cache' => [
          'max-age' => 300,
        ],
```

```
      ];
    }
    catch (RequestException $e) {
      return [
        '#markup' => $this->t('Помилка при з\'єднанні з Redmine: @error', ['@error' =>
$e->getMessage()]),
      ];
    }
  }

  /**
   * Displays Gantt chart for a project.
   */
  public function showProjectGantt($project_id) {
    $config = $this->config('gant_drupal_module.settings');
    $redmine_url = $config->get('redmine_url');
    $api_key = $config->get('api_key');

    if (empty($redmine_url) || empty($api_key)) {
      return [
        '#markup' => $this->t('Будь ласка, налаштуйте інтеграцію з Redmine в
налаштуваннях модуля.'),
      ];
    }

    try {
      $client = new Client();
      $response = $client->request('GET', $redmine_url . '/projects/' . $project_id .
'/issues.json', [
        'headers' => [
          'X-Redmine-API-Key' => $api_key,
          'Content-Type' => 'application/json',
        ],
        'query' => [
          'include' => 'relations',
        ],
      ]);

      $data = json_decode($response->getBody(), TRUE);
      $issues = $data['issues'] ?? [];

      // Get project details
```

```
    $project_response = $client->request('GET', $redmine_url . '/projects/' . $project_id
. '.json', [
      'headers' => [
        'X-Redmine-API-Key' => $api_key,
        'Content-Type' => 'application/json',
      ],
    ]);

    $project_data = json_decode($project_response->getBody(), TRUE);
    $project = $project_data['project'] ?? [];

    // Prepare data for the Gantt chart
    $gantt_data = [];
    foreach ($issues as $issue) {
      if (!empty($issue['start_date']) && !empty($issue['due_date'])) {
        $gantt_data[] = [
          'id' => $issue['id'],
          'text' => $issue['subject'],
          'start_date' => $issue['start_date'],
          'end_date' => $issue['due_date'],
          'progress' => isset($issue['done_ratio']) ? $issue['done_ratio'] / 100 : 0,
          'status' => $issue['status']['name'],
        ];
      }
    }

    return [
      '#theme' => 'gant_chart',
      '#projects' => [
        'name' => $project['name'],
        'issues' => $gantt_data,
      ],
      '#chart_id' => 'gantt-chart-' . $project_id,
      '#attached' => [
        'library' => [
          'gant_drupal_module/gantt_chart',
        ],
        'drupalSettings' => [
          'gantDrupalModule' => [
            'projectId' => $project_id,
            'ganttData' => $gantt_data,
          ],
        ],
```

```
        ],
      ];
    }
    catch (RequestException $e) {
      return [
        '#markup' => $this->t('Помилка отримання даних проекту з Redmine: @error',
['@error' => $e->getMessage()]),
      ];
    }
  }
}
EOL

# Створіть форму налаштувань
cat > src/Form/SettingsForm.php << 'EOL'
<?php

namespace Drupal\gant_drupal_module\Form;

use Drupal\Core\Form\ConfigFormBase;
use Drupal\Core\Form\FormStateInterface;

/**
 * Configuration form for Gant Drupal Module.
 */
class SettingsForm extends ConfigFormBase {

  /**
   * {@inheritdoc}
   */
  public function getFormId() {
    return 'gant_drupal_module_settings_form';
  }

  /**
   * {@inheritdoc}
   */
  protected function getEditableConfigNames() {
    return ['gant_drupal_module.settings'];
  }

  /**
   * {@inheritdoc}
```

```
   */
  public function buildForm(array $form, FormStateInterface $form_state) {
    $config = $this->config('gant_drupal_module.settings');

    $form['redmine_url'] = [
      '#type' => 'url',
      '#title' => $this->t('URL Redmine'),
      '#description' => $this->t('Базова адреса вашого Redmine (наприклад, http://redmine:3000).'),
      '#default_value' => $config->get('redmine_url'),
      '#required' => TRUE,
    ];

    $form['api_key'] = [
      '#type' => 'textfield',
      '#title' => $this->t('API ключ'),
      '#description' => $this->t('Ваш ключ API Redmine.'),
      '#default_value' => $config->get('api_key'),
      '#required' => TRUE,
    ];

    return parent::buildForm($form, $form_state);
  }

  /**
   * {@inheritdoc}
   */
  public function submitForm(array &$form, FormStateInterface $form_state) {
    $this->config('gant_drupal_module.settings')
      ->set('redmine_url', $form_state->getValue('redmine_url'))
      ->set('api_key', $form_state->getValue('api_key'))
      ->save();

    parent::submitForm($form, $form_state);
  }
}
EOL

# Створіть шаблон для діаграми Ганта
cat > templates/gant-chart.html.twig << 'EOL'
<div class="gant-chart-container">
  <h2>{{ projects.name }}</h2>
  <div id="{{ chart_id }}" class="gantt-chart"></div>
```

```
</div>
EOL

# Створіть JavaScript файл
cat > js/gant_drupal_module.js << 'EOL'
(function ($, Drupal, drupalSettings) {
  'use strict';

  Drupal.behaviors.gantDrupalModule = {
    attach: function (context, settings) {
      if (drupalSettings.gantDrupalModule &&
drupalSettings.gantDrupalModule.ganttData) {
        var chartId = $('#' + drupalSettings.gantDrupalModule.chartId, context);
        if (chartId.length) {
          gantt.config.date_format = "%Y-%m-%d";
          gantt.init(chartId.attr('id'));

          var tasks = {
            data: drupalSettings.gantDrupalModule.ganttData
          };

          gantt.parse(tasks);
        }
      }
    }
  };
})(jQuery, Drupal, drupalSettings);
EOL

# Створіть CSS файл
cat > css/gant_drupal_module.css << 'EOL'
/* Основний контейнер діаграми */
.gant-chart-container {
 padding: 20px;
 background: linear-gradient(135deg, #f9f9f9 0%, #e8f4f8 100%);
 border-radius: 8px;
 box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
 margin: 25px 0;
 position: relative;
}

/* Сама діаграма Ганта */
.gantt-chart {
```

```
 width: 100%;
 height: 500px;
 margin: 20px 0;
 background: white;
 border-radius: 6px;
 border: 1px solid #e1e8ed;
 overflow: hidden;
}

/* Заголовок проекту */
.gant-chart-container h2 {
 color: #2c3e50;
 font-size: 24px;
 font-weight: 600;
 margin: 0 0 15px 0;
 text-shadow: 0 1px 2px rgba(0, 0, 0, 0.1);
}

/* Опис проекту */
.gant-project-description {
 margin-bottom: 20px;
 font-style: italic;
 color: #5a6c7d;
 font-size: 16px;
 line-height: 1.5;
 padding: 10px 0;
 border-bottom: 1px solid #ecf0f1;
}

/* Стилі для різних статусів завдань - New */
.gantt-task-new .gantt_task_progress {
 background: linear-gradient(90deg, #c3e6cb 0%, #a8d5ba 100%);
 box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1);
}

.gantt_task_line.gantt-task-new {
 background: linear-gradient(135deg, #d4edda 0%, #c3e6cb 100%);
 border-color: #28a745;
 border-width: 2px;
 border-style: solid;
 border-radius: 4px;
}
```

```
/* Стилі для статусу In Progress */
.gantt-task-progress .gantt_task_progress {
 background: linear-gradient(90deg, #ffeeba 0%, #ffe69c 100%);
 box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1);
}

.gantt_task_line.gantt-task-progress {
 background: linear-gradient(135deg, #fff3cd 0%, #ffeeba 100%);
 border-color: #ffc107;
 border-width: 2px;
 border-style: solid;
 border-radius: 4px;
}

/* Стилі для статусу Resolved */
.gantt-task-resolved .gantt_task_progress {
 background: linear-gradient(90deg, #bee5eb 0%, #9fdbea 100%);
 box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1);
}

.gantt_task_line.gantt-task-resolved {
 background: linear-gradient(135deg, #d1ecf1 0%, #bee5eb 100%);
 border-color: #17a2b8;
 border-width: 2px;
 border-style: solid;
 border-radius: 4px;
}

/* Стилі для статусу Closed */
.gantt-task-closed .gantt_task_progress {
 background: linear-gradient(90deg, #c3c3c3 0%, #a8a8a8 100%);
 box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1);
}

.gantt_task_line.gantt-task-closed {
 background: linear-gradient(135deg, #e2e3e5 0%, #d6d8db 100%);
 border-color: #6c757d;
 border-width: 2px;
 border-style: solid;
 border-radius: 4px;
 opacity: 0.8;
}
```

```
/* Додаткові стилі для покращення UX */
.gantt_task_line {
 transition: all 0.3s ease;
 cursor: pointer;
}

.gantt_task_line:hover {
 transform: translateY(-1px);
 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15);
}

/* Стилі для мобільних пристроїв */
@media (max-width: 768px) {
 .gant-chart-container {
   padding: 15px;
   margin: 15px 0;
 }

 .gantt-chart {
   height: 400px;
 }

 .gant-chart-container h2 {
   font-size: 20px;
 }
}
EOL

# Встановіть правильні права доступу
chown -R www-data:www-data /opt/drupal/web/modules/custom/gant_drupal_module
```

Додаток Б

Лістинг файлу для тестування

```
#!/bin/bash
# test_performance.sh

echo "=== Тестування продуктивності Gant Drupal Module ==="
echo "Дата: $(date)"
echo ""

# URLs для тестування
PROJECTS_URL="http://localhost:8080/gant-drupal-module/projects"
GANTT_URL="http://localhost:8080/gant-drupal-module/projects/1/gantt"

# Функція для вимірювання часу
measure_time() {
    local url=$1
    local name=$2
    local total_time=0
    local successful=0

    echo "Тестування: $name"
    echo "URL: $url"

    for i in {1..10}; do
        response_time=$(curl -w "%{time_total}" -o /dev/null -s "$url" 2>/dev/null)
        http_code=$(curl -w "%{http_code}" -o /dev/null -s "$url" 2>/dev/null)

        if [ "$http_code" = "200" ]; then
            total_time=$(echo "$total_time + $response_time" | bc)
            successful=$((successful + 1))
            echo "  Запит $i: ${response_time}s (OK)"
        else
            echo "  Запит $i: Помилка HTTP $http_code"
        fi
    done

    avg_time=$(echo "scale=3; $total_time / $successful" | bc)
    reliability=$(echo "scale=1; $successful * 100 / 10" | bc)

    echo "  Результати:"
    echo "    Середній час: ${avg_time}s"
    echo "    Надійність: ${reliability}%"
    echo "    Успішних запитів: $successful/10"
```

```
    echo ""
}

# Запуск тестів
measure_time "$PROJECTS_URL" "Список проектів"
measure_time "$GANTT_URL" "Діаграма Ганта"
```