Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та безпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ТА ОПТИМІЗАЦІЯ СЕРВЕРНОГО СЕРЕДОВИЩА ДЛЯ УПРАВЛІННЯ КОРПОРАТИВНОЮ СИСТЕМОЮ

DEVELOPMENT AND OPTIMIZATION OF THE SERVER ENVIRONMENT FOR CORPORATE SYSTEM MANAGEMENT

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІс-21
Тишко Микола Ярославович

(підпис)

Керівник:
к.т.н., доцент
Мельник Катерина Вікторівна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 10 » червня 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Т. Терлецький
« 10 » 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Тишку Миколі Ярославовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Розробка та оптимізація серверного середовища для управління корпоративною системою*

Керівник роботи *к.т.н., доцент Мельник Катерина Вікторівна*

затверджені наказом закладу вищої освіти від «04» січня 2025 року № 11/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 10.06.2025р.

3. Вихідні дані до роботи *джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування.*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Вивчення основ створення системи управління проектами
Аналіз технологій для створення серверного середовища
Реалізація серверного середовища
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Схема бази даних
Файли конфігурацій серверних налаштувань
Скрипти для резервного копіювання
Інтерфейс системи управління проєктами

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Вивчення основ створення системи управління проектами	*Мельник К.В., доцент*		
Аналіз технологій для створення серверного середовища	*Мельник К.В., доцент*		
Реалізація серверного середовища	*Мельник К.В., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,ст.викладач*		

7. Дата видачі завдання 10.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми, аналіз проблемної області на наявних рішень*	до 10.02.2025 р.	Виконано
2.	*Аналіз технологій для створення серверного середовища*	до 02.03.2025 р.	Виконано
3.	*Реалізація серверного середовища*	до 02.04.2025 р.	Виконано
4.	*Висновки та пропозиції*	до 10.04.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 15.04.2025 р.	Виконано
6.	*Формування додатків*	до 02.05.2025 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 10.05.2025 р.	Виконано
8.	*Представлення остаточного варіанту кваліфікаційної роботи керівникові*	до 15.05.2025 р.	Виконано
9.	*Нормоконтроль*	до 30.05.2025 р.	Виконано
10	*Інструментальна перевірка на академічний плагіат*	до 03.06.2025 р	Виконано
11.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ (підпис) Тишко М.Я. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Мельник К.В. (прізвище, ініціали)

АНОТАЦІЯ

Тишко М.Я. Розробка та оптимізація серверного середовища для управління корпоративною системою. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Перший розділ присвячено аналізу предметної області корпоративних інформаційних систем та управління проєктами, обґрунтуванню актуальності створення захищених і надійних рішень для автоматизації бізнес-процесів. Окрему увагу приділено огляду сучасних підходів до захисту даних, резервного копіювання та безперервності роботи систем.

У другому розділі описано вибір засобів розробки та обґрунтування архітектури системи. Розглянуто вибір стеку технологій (Node.js, PostgreSQL, React), особливості побудови захищеного API, підходи до організації зберігання та реплікації даних у PostgreSQL, а також принципи створення сучасного веб-інтерфейсу для користувачів і адміністраторів.

Третій розділ присвячено реалізації системи управління проєктами: детально описано структуру бази даних, особливості впровадження серверної й клієнтської частин, налаштування механізмів реплікації та резервного копіювання, а також проведено тестування функціональності, безпеки та відмовостійкості системи. Представлено результати впровадження та можливості подальшого розвитку розробленого програмного комплексу.

Ключові слова: корпоративна система, PostgreSQL, реплікація, Node.js, API, резервне копіювання.

ANNOTATION

Tyshko M. Development and optimization of the server environment for corporate system management. Manuscript.

Qualification work for bachelor's degree in Computer Engineering, specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

The qualification work consists of an introduction, three chapters, conclusions, a list of references and appendices.

The first section is devoted to the analysis of the subject area of corporate information systems and project management, justification of the relevance of creating secure and reliable solutions for business process automation. Special attention is paid to an overview of modern approaches to data protection, backup, and system continuity.

The second section describes the choice of development tools and the justification of the system architecture. It considers the choice of technology stack (Node.js, PostgreSQL, React), the features of building a secure API, approaches to organizing data storage and replication in PostgreSQL, and the principles of creating a modern web interface for users and administrators.

The third section is devoted to the implementation of the project management system: the structure of the database, the specifics of implementing the server and client parts, setting up replication and backup mechanisms, and testing the functionality, security, and fault tolerance of the system are described in detail. The results of implementation and possibilities for further development of the developed software system are presented.

Keywords: corporate system, PostgreSQL, replication, Node.js, API, backup.

ЗМІСТ

ВСТУП

Корпоративні інформаційні системи дозволяють автоматизувати бізнес-процеси, підвищити прозорість та оперативність прийняття управлінських рішень. Особливої актуальності набувають питання захисту даних та безперервності роботи IT-систем, зокрема шляхом впровадження реплікації та резервного копіювання баз даних. Створення надійної та масштабованої системи управління проєктами із сучасними механізмами захисту є важливим завданням для підприємств у різних галузях.

Метою роботи є розробка та впровадження серверного рішення для корпоративної системи управління проєктами, що забезпечує збереження, реплікацію та захист даних, а також інтеграцію із сучасним веб-інтерфейсом.

Об’єкт дослідження – корпоративні інформаційні системи управління проєктами.

Предмет дослідження – методи та засоби побудови серверної частини системи управління проєктами з підтримкою реплікації даних, безпеки та інтеграції із клієнтською частиною.

Завдання, які необхідно виконати:

– розробити архітектуру корпоративної системи управління проєктами;

– реалізувати серверну частину із захищеним API для взаємодії з базою даних;

– дослідити можливості впровадження реплікації та резервного копіювання у PostgreSQL;

– спроектувати структуру бази даних для зберігання інформації про задачі, користувачів та коментарі.

Результати роботи доповідалися на міжнародній науково-практичній конференції молодих вчених та студентів «Програмне та апаратне забезпечення в інформаційних технологіях», Луцьк, 6 травня 2025 р. [30].

РОЗДІЛ 1
ОСНОВИ СТВОРЕННЯ КОРПОРАТИВНИХ СИСТЕМ УПРАВЛІННЯ ПРОЕКТАМИ

1.1 Поняття та принципи управління проєктами

Управління проєктами є сучасною методологією організації діяльності, що дозволяє підприємствам, установам і командам ефективно реалізовувати стратегічні ініціативи, адаптуватися до динамічного середовища та підвищувати власну конкурентоспроможність [5, 24].

Проєкт, згідно з класичними визначеннями, це унікальна сукупність вхідних даних, таких як цілі чи проблеми, а також процесів і ресурсів, які спрямовані на те, щоб у визначений проміжок часу досягти очікуваних результатів, наприклад, розробити новий застосунок [27].

У найзагальнішому вигляді проєкт (від англ. project) – це будь-що, що замислюється або планується: від нової інформаційної системи для бухгалтерії до розробки складного інноваційного продукту. З точки зору системного підходу, проєкт розглядається як процес переходу з початкового стану в кінцевий результат, що досягається через виконання низки взаємопов'язаних дій з урахуванням обмежень і механізмів. Також проєкт можна визначити як задачу з конкретними вихідними даними та необхідними результатами (цілями), де спосіб досягнення обумовлюється специфікою поставленої задачі [28].

Застосування проєктного підходу дає змогу організаціям упорядковувати діяльність, чітко керувати ресурсами, досягати визначених цілей у межах заданих часових, бюджетних та інших обмежень, впроваджувати інновації та оперативно реагувати на зміни зовнішнього середовища [5].

Проєкти мають низку характерних особливостей, які вирізняють їх із-поміж рутинної операційної діяльності: по-перше, кожен проєкт має чітко визначену мету, а його реалізація означає послідовне досягнення проміжних і кінцевих цілей, формулювання яких визначає весь процес управління. По-друге, для проєктів характерна координованість і взаємозалежність дій, коли деякі

етапи виконуються паралельно, інші – послідовно, а порушення цього порядку може негативно вплинути на результативність [18, 25].

Важливою ознакою будь-якого проєкту є обмеженість у часі: проєкт має визначені початок і завершення, а оптимальний розподіл зусиль і ресурсів досягається за допомогою ретельного планування і впорядкування заходів та робіт у межах проєктної діяльності. Відмінність проєкту від виробничої або операційної діяльності також полягає в його одноразовості, тобто кожен проєкт є унікальним завданням, навіть якщо він пов'язаний з подібною діяльністю в минулому.

У сучасних організаціях проєктний підхід все частіше застосовується й до повторюваних процесів, наприклад, виконання замовлень із визначеними договірними термінами постачання [5, 24]. Серед ключових характеристик проєкту необхідно також виділити обмеженість ресурсів – бюджетних, людських, матеріальних, технологічних, часових, що прямо впливає на вибір інструментів та методів управління.

Ефективне управління проєктами базується на певних принципах, які забезпечують результативність проєктної діяльності та досягнення поставлених цілей. По-перше, це цілеспрямованість чітке визначення цілей проєкту та орієнтація всієї команди на їх досягнення. Раціональне використання обмежених ресурсів (часу, бюджету, персоналу) дозволяє досягти максимального результату за мінімальних витрат [27]. Наступний принцип – управління ризиками, тобто ідентифікація, аналіз та мінімізація потенційних загроз для проєкту, що можуть негативно вплинути на його успіх. Не менш важливим є принцип комунікації, який передбачає ефективний обмін інформацією між усіма учасниками проєкту для досягнення спільного розуміння та координації дій. Здатність проєктної команди до гнучкості та адаптивності, швидке реагування на зміни у зовнішньому й внутрішньому середовищі, а також орієнтація на кінцевий результат і задоволення потреб замовника є фундаментом для побудови сучасних систем управління проєктами [24, 27].

Ці принципи, що лежать в основі управління проєктами, втілені у міжнародних стандартах (PMBOK, ISO 21500) та є невід'ємною частиною сучасної професійної освіти у сфері менеджменту проєктів [24]. На основі вищевказаних підходів можна стверджувати, що проєктний менеджмент сьогодні – це інтегративна дисципліна, яка поєднує у собі елементи планування, організації, мотивації, контролю, комунікацій, управління змінами та ризиками, дозволяючи ефективно досягати унікальних цілей та розвивати потенціал організації в умовах постійних змін [5, 24, 25, 29].

1.2 Серверні рішення для корпоративних систем управління проєктами

Серверні рішення є основою корпоративних систем управління проєктами (КСУП), оскільки саме вони забезпечують централізоване зберігання даних, обробку інформації та доступ користувачів до необхідних ресурсів і функціоналу. Вибір відповідної серверної платформи, її архітектури та інструментарію визначає не лише ефективність реалізації проєктів, але й надійність підтримки, масштабованість і подальший розвиток всієї системи управління. КСУП у сучасному вигляді спираються на системний і процесний підходи до управління проєктами, які застосовуються незалежно від виду діяльності підприємства, та поєднують знання, навички, інструменти й методи для вирішення завдань у межах життєвого циклу проєкту. Впровадження КСУП передбачає визначення не лише організаційної структури, але й конкретних ролей і функцій персоналу, формалізованих взаємовідносин між учасниками, а також затверджених стандартів і методологій, що регламентують порядок взаємодії та прийняття рішень у межах проєкту [5].

Складовими елементами сучасної КСУП є методологічна база (стандарти, корпоративні регламенти, політики управління), персонал (кваліфіковані керівники проєктів, аналітики, виконавці, які діють у межах єдиних правил), а також потужні інструментальні засоби – спеціалізовані програмні й технічні

рішення, які створюють єдиний інформаційний простір для учасників проєктів, автоматизують процеси обміну даними, планування, моніторингу, контролю та звітності. Програмна інфраструктура КСУП дозволяє забезпечити прозорість і контроль за виконанням завдань, а також реалізовувати принципи централізованого керування ресурсами та ризиками, що є ключовими для досягнення успіху в сучасних конкурентних умовах [5].

Функціонування більшості корпоративних систем управління проєктами базується на архітектурі клієнт-сервер. Суть цієї архітектури полягає у взаємодії двох основних компонентів: клієнта (зазвичай це браузер, додаток або інший користувацький інтерфейс) і сервера (серверної частини, що обробляє запити, управляє даними, виконує бізнес-логіку тощо). Клієнт надсилає запити на сервер (наприклад, запит на отримання сторінки, створення або редагування задачі), а сервер у відповідь надає необхідний контент чи виконує операції над даними. Таким чином, сервер є ключовим елементом збереження, обробки й передачі інформації між користувачами та базою даних. Для розширення функціональних можливостей корпоративних web-систем до серверної частини інтегруються спеціалізовані програми: вони забезпечують обробку даних, валідацію, взаємодію з базами даних, підтримку авторизації та безпеки, проведення розрахунків і аналітики. Сукупність таких інструментів і методів складає поняття серверних технологій у сфері управління проєктами [25].

Для розробки сучасних серверних рішень у КСУП використовуються потужні інструментарії – фреймворки, які містять попередньо підготовлені, стабільно протестовані компоненти та структуру проєкту. Використання фреймворків (наприклад, Symfony 5+, Laravel 9 для PHP; NestJS, Express для NodeJS; Flask, Django для Python) значно скорочує час впровадження нових рішень, полегшує масштабування та подальшу підтримку. Крім того, сучасні фреймворки підтримують високий рівень безпеки, логування, тестування і забезпечують зручну інтеграцію з іншими корпоративними сервісами й інструментами [5].

Типова структура серверних рішень для КСУП включає декілька ключових компонентів. Сервер баз даних відповідає за зберігання, обробку та управління всіма корпоративними даними: інформацією про проєкти, завдання, ресурси, фінанси, строки, документи тощо. Сервер додатків реалізує бізнес-логіку системи, виконує обробку запитів користувачів, здійснює взаємодію між клієнтським інтерфейсом і базою даних. Веб-сервер забезпечує зручний доступ до функціоналу системи через web-інтерфейс із будь-якої точки світу, підтримуючи різні рівні доступу. Сервер автентифікації та авторизації контролює, які саме користувачі та з якими правами можуть працювати з корпоративною інформацією, що дозволяє забезпечити необхідний рівень захисту та конфіденційності даних.

У сучасній практиці управління проєктами використовуються різноманітні серверні рішення, серед яких особливої популярності набули такі платформи, як OpenProject, RedmineUP, PLANTA Project та Azure DevOps Server. OpenProject – це відкритий серверний продукт для управління проєктами, який може працювати як з класичними, так і з гнучкими методологіями, підтримує діаграми Ганта, Kanban-дошки, систему відстеження задач, управління часом та ресурсами, може бути розгорнутий на власних серверах організації для максимального контролю над даними і забезпечення відповідності корпоративним стандартам безпеки [18]. RedmineUP – це розширена версія Redmine з додатковими плагінами для роботи з проєктами, CRM, Helpdesk та іншими корпоративними сервісами. Вона підтримує як хмарну, так і локальну інсталяцію, що дозволяє вибрати оптимальний сценарій під потреби підприємства. PLANTA Project орієнтований на великі компанії, що одночасно ведуть кілька десятків або сотень проєктів, і дає можливість планувати ресурси, контролювати портфелі проєктів, впроваджувати гібридні методології, адаптувати систему під вимоги користувачів. Azure DevOps Server – рішення від Microsoft для повного управління життєвим циклом програмного забезпечення, підтримує інтеграцію з іншими корпоративними продуктами Microsoft, такими

як SharePoint, SQL Server, надає можливості гнучкого налаштування робочих процесів і ролей у великих командах [9, 19].

Вибір і впровадження серверних рішень у корпоративних системах управління проєктами дозволяє забезпечити низку критично важливих переваг. Передусім, це централізація даних: всі корпоративні та проєктні дані зберігаються в єдиному сховищі, що підвищує їхню цілісність, знижує ймовірність дублювання або втрати, спрощує контроль версій та резервне копіювання.

Масштабованість сучасних серверних платформ дозволяє ефективно обслуговувати велику кількість користувачів, команд і проєктів без втрати продуктивності, оперативно реагувати на зміну навантаження. Важливою перевагою є безпека – серверні рішення передбачають можливість реалізації комплексних заходів захисту: шифрування даних, налаштування багаторівневої автентифікації, аудит дій користувачів, контроль доступу, регулярне резервне копіювання та відновлення. Інтеграція з іншими корпоративними системами (ERP, CRM, бухгалтерія тощо) дозволяє автоматизувати потоки даних, уникати помилок, підвищувати загальну ефективність діяльності організації. Ще одним важливим аспектом є гнучкість у налаштуванні бізнес-процесів: сучасні серверні рішення підтримують кастомізацію під потреби конкретної компанії, дозволяють впроваджувати як класичні водоспадні, так і гнучкі (Agile, Scrum, Kanban) або гібридні методології управління [5].

Підсумовуючи, можна відзначити, що серверні рішення є невід'ємною й ключовою складовою корпоративних систем управління проєктами. Їх впровадження дозволяє не лише централізувати й захистити дані, але й забезпечити високу продуктивність, масштабованість, безпеку й адаптивність системи до потреб підприємства. Сучасні серверні платформи дають змогу організувати ефективну взаємодію між усіма учасниками проєкту, автоматизувати рутинні процеси, підвищити рівень контролю та прозорості, що в кінцевому підсумку позитивно впливає на досягнення стратегічних цілей організації. Вибір і грамотне налаштування серверних рішень у складі КСУП є

одним із головних факторів успішної цифрової трансформації бізнесу в умовах сучасного ринку.

1.3 Роль реляційних баз даних у корпоративних системах управління проєктами

У корпоративних системах управління проєктами (КСУП) реляційні бази даних відіграють центральну роль як основа для надійного зберігання, обробки та аналізу даних. Саме реляційні бази даних (РБД) забезпечують структуроване представлення інформації, підтримку цілісності даних, гнучкість масштабування та надають змогу ефективно керувати великими обсягами інформації, що є критично важливим для успішного виконання складних корпоративних проєктів. База даних – це набір логічно пов'язаних даних спільного використання (і опис цих даних), призначений для задоволення інформаційних потреб різних підрозділів організації. Ключовими ознаками бази даних є централізоване сховище даних, що визначається одноразово, а потім одночасно використовується багатьма користувачами; загальний корпоративний ресурс; зберігає не лише дані, а і їх опис [26].

Використання реляційних СУБД у КСУП дає змогу отримати цілий ряд суттєвих переваг, які підвищують продуктивність і надійність роботи систем. Серед них: контроль за надмірністю даних, несуперечність даних, більше корисної інформації при тому ж обсязі збережених даних, спільне використання даних, підтримка цілісності, підвищена безпека, застосування стандартів, ефективність при масштабуванні, підвищення готовності та доступності даних, покращення продуктивності, спрощення супроводу системи за рахунок незалежності від даних, покращене керування паралельністю, розвинені служби резервного копіювання та відновлення [26]. Завдяки цьому КСУП можуть підтримувати актуальність, цілісність та безпеку корпоративних даних на всіх етапах життєвого циклу проєкту.

РБД гарантують підтримку транзакцій і ACID-властивостей, тобто забезпечують, що всі операції з даними будуть виконані повністю або не виконані зовсім, що підвищує надійність і передбачуваність системи. Сучасні реляційні бази даних підтримують як горизонтальне, так і вертикальне масштабування, що дозволяє ефективно обробляти великі й зростаючі обсяги даних без втрати продуктивності. До того ж мова SQL, яка використовується у більшості РБД, дозволяє формулювати складні запити для аналітики, звітності та прийняття рішень, що особливо цінно в управлінні проєктами.

Слід зауважити, що реляційна база даних – це набір нормалізованих відношень, тобто таблиць, пов’язаних між собою через ключі та правила цілісності [26]. У великих організаціях також встановлюються корпоративні обмеження цілісності, які визначаються користувачами або адміністраторами бази даних і забезпечують відповідність корпоративним політикам та стандартам обробки інформації [26].

Реляційні бази даних є невід’ємною частиною багаторівневих і багатокористувацьких систем. Зазвичай такі системи мають щонайменше чотири основні компоненти: базу даних, логіку транзакцій, логіку додатка та інтерфейс користувача. У сучасному web-середовищі дворівнева модель “клієнт/сервер” замінюється трирівневою, що включає рівень інтерфейсу користувача (тонкий клієнт), рівень бізнес-логіки (сервер додатку) і рівень СУБД (сервер бази даних), які можуть розподілятися між різними комп’ютерами, фізичними або віртуальними серверами [26].

Прикладом сучасної реляційної СУБД, яка повністю відповідає вимогам корпоративних систем управління проєктами, є PostgreSQL. Це потужна об'єктно-реляційна система управління базами даних із відкритим кодом, що підтримує численні типи даних (числові, символьні, часові, геометричні тощо), дозволяє зберігати різноманітну інформацію про проєкти, легко розширюється через створення власних функцій, типів та операторів, підтримує процедурне програмування за допомогою мови PL/pgSQL і забезпечує високий рівень доступності та захисту даних завдяки механізмам реплікації і резервного

копіювання [17]. PostgreSQL успішно використовується для підтримки складних бізнес-процесів, реалізації аналітики, інтеграції з іншими корпоративними системами та побудови гнучких рішень, що масштабуються відповідно до зростання організації.

Підсумовуючи, реляційні бази даних – це не лише інструмент зберігання інформації, а й ключовий компонент, що забезпечує надійність, цілісність, безпеку та доступність корпоративних даних у складних і багатокористувацьких системах управління проєктами. Використання сучасних СУБД, зокрема PostgreSQL, дає змогу організаціям підвищити ефективність виконання проєктів, забезпечити відповідність корпоративним і галузевим стандартам, а також досягати стратегічних бізнес-цілей у динамічному середовищі.

РОЗДІЛ 2
АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СЕРВЕРНОГО СЕРЕДОВИЩА

2.1 Аналіз вимог до серверного середовища

Серверне середовище корпоративної системи управління проєктами має відповідати низці функціональних і нефункціональних вимог, які визначають ефективність, продуктивність та надійність усієї інформаційної системи. З-поміж ключових функціональних вимог насамперед виділяється обробка сервером запитів від клієнтської частини (frontend), виконання операцій над даними у базі даних і повернення результатів у структурованому вигляді, наприклад, у форматі JSON.

Необхідною умовою є реалізація REST API, що дозволяє уніфікувати програмний інтерфейс для виконання CRUD-операцій (створення, читання, оновлення, видалення) над такими сутностями, як задачі, користувачі та інші корпоративні об'єкти. REST-архітектура робить можливим просту інтеграцію із різними клієнтськими застосунками, забезпечує гнучкість і масштабованість розробки, а також дає змогу швидко впроваджувати новий функціонал відповідно до змін бізнес-процесів [10].

Наприклад, створення нової задачі або зміна статусу відбуваються через стандартний POST/PUT-запит до REST API, результат якого миттєво відображається у клієнтському інтерфейсі. Крім того, серверна частина повинна підтримувати сучасні механізми аутентифікації та авторизації користувачів. Йдеться про можливість реєстрації, входу користувача, перевірки ролей і доступу до певних даних чи дій залежно від призначеної ролі. Реалізація аутентифікації часто здійснюється за допомогою JWT (JSON Web Tokens), які передаються у заголовках HTTP-запитів, а зберігання паролів – лише у хешованому вигляді із використанням стійких алгоритмів, таких як bcrypt [1].

Для багатокористувацьких систем важливо забезпечити одночасну роботу кількох користувачів із захистом від конфліктів при паралельному доступі до

даних; для цього застосовують механізми блокування, черги, транзакції. Вся активність у системі має бути задокументована через систему логування: події аутентифікації, зміни у даних, помилки, підозріла активність тощо, що дозволяє оперативно виявляти та локалізувати проблеми, а також підвищує рівень безпеки системи [13]. Наприклад, при спробі неавторизованого доступу до задачі система не лише блокує дію, а й залишає запис у журналі подій.

Нефункціональні вимоги визначають якісний рівень роботи серверної частини корпоративної системи та охоплюють питання продуктивності, масштабованості, безпеки, відмовостійкості та зручності адміністрування. Продуктивність полягає у здатності сервера обробляти запити із мінімально можливою затримкою, навіть у пікові періоди навантаження. Досягається це завдяки використанню сучасних неблокуючих технологій, наприклад, Node.js, оптимізації роботи із базою даних (ефективні індекси, пул з'єднань), а також кешуванню повторюваних запитів, наприклад, за допомогою Redis [11].

Масштабованість забезпечується проєктуванням архітектури, яка дозволяє швидко збільшувати потужності – як на рівні програмного забезпечення (горизонтальне або вертикальне масштабування серверів, контейнеризація), так і на рівні бази даних, через кластеризацію та розподілення навантаження між основною базою та репліками [11]. Наприклад, додавання нового вузла до пулу серверів дозволяє розподіляти трафік без простоїв для користувачів.

Безпека – один із найважливіших критеріїв. Усі персональні та службові дані мають бути захищені, передача даних здійснюється тільки захищеними протоколами (HTTPS), усі конфіденційні дані зберігаються у зашифрованому вигляді. Для захисту від типових атак (SQL-ін'єкції, XSS, CSRF) застосовуються валідація вхідних даних, механізми CORS, регулярне оновлення залежностей і аудит коду [12]. Наприклад, при спробі SQL-ін'єкції система одразу ідентифікує некоректний запит і блокує його, не допускаючи зміни або зчитування чужих даних.

Відмовостійкість означає, що система повинна залишатися працездатною навіть у разі відмови окремих компонентів. Досягається це впровадженням

реплікації бази даних, регулярним резервним копіюванням, використанням моніторингових систем для швидкого виявлення та ліквідації проблем [16]. Наприклад, якщо один сервер бази даних виходить із ладу, інший репліка-сервер одразу підхоплює обслуговування клієнтів, а користувачі не помічають жодних перебоїв у роботі.

Вимоги до зберігання і реплікації даних також відіграють важливу роль. У сучасних корпоративних системах доцільно використовувати реляційні бази даних, наприклад, PostgreSQL, яка забезпечує потужні засоби транзакційної роботи, підтримує складні запити, гарантує цілісність даних навіть під великими навантаженнями [17]. Для підвищення надійності та доступності використовується реплікація PostgreSQL, що дозволяє мати кілька копій бази даних на різних серверах, розподіляти навантаження, забезпечувати відновлення у разі збою основного вузла, і мінімізувати час простою [16]. Кожна транзакція у системі має виконуватись із забезпеченням узгодженості даних у всіх репліках за допомогою вбудованих механізмів, таких як WAL (Write Ahead Log) [16]. Для запобігання втраті даних у випадку збоїв або атак передбачається регулярне створення резервних копій (бекапів) та автоматичне їх тестування.

Масштабування на рівні даних реалізується за рахунок додавання нових таблиць, індексів, оптимізації структури БД, а також підтримки великої кількості одночасних підключень [19]. Наприклад, під час розширення системи для нового відділу можна додати відповідні таблиці та налаштувати додаткові репліки, не зупиняючи основну роботу всієї КСУП.

Таким чином, грамотне поєднання функціональних, нефункціональних вимог і вимог до зберігання даних дозволяє побудувати надійну, продуктивну й масштабовану серверну частину корпоративної системи управління проєктами, яка відповідає сучасним стандартам та легко адаптується до майбутніх потреб організації.

2.2 Вибір технологій для розробки та налаштування серверного середовища

Вибір технологій для створення серверного середовища системи управління проєктами здійснювався на основі детального аналізу функціональних і нефункціональних вимог, а також сучасних тенденцій у сфері розробки web-додатків. Нижче наведено докладний огляд ключових елементів обраного стеку технологій та їх обґрунтування.

В якості основної мови програмування та середовища виконання для серверної частини було обрано JavaScript у поєднанні з Node.js. Основними причинами такого вибору є асинхронна та неблокуюча модель обробки запитів у Node.js, що дає змогу ефективно працювати з великою кількістю одночасних з'єднань – це критично для багатокористувацьких корпоративних систем управління проєктами [11].

Наприклад, у ситуації, коли десятки користувачів одночасно надсилають запити до сервера (створення задач, зміна статусу, коментарі), Node.js дозволяє обробляти їх паралельно без затримок. Додатковою перевагою є велика екосистема npm-пакетів, яка спрощує інтеграцію нових функцій, таких як автентифікація, логування чи робота з базами даних, а також можливість використовувати одну мову (JavaScript) на клієнті й сервері, що підвищує уніфікованість проекту та спрощує його підтримку. На сьогодні Node.js є однією з найпопулярніших платформ для розробки високопродуктивних веб-сервісів, REST API й мікросервісної архітектури [11].

Серверним фреймворком було обрано Express.js, який є стандартом де-факто для побудови REST API на платформі Node.js. Серед переваг Express.js варто відзначити легкість і гнучкість у створенні маршрутизованих додатків, підтримку middleware, що дозволяє додавати фільтри, логування, автентифікацію та інші функції до обробки кожного запиту. Фреймворк має широку документацію, потужну спільноту та підтримує модульність, що дає змогу легко інтегрувати сторонні бібліотеки для захисту, роботи з файлами,

аналітики чи оптимізації продуктивності [6]. Практично це означає, що будь-які зміни чи додавання нового API-методу відбуваються швидко і прозоро для всієї команди розробників.

Для забезпечення захисту API, гнучкої конфігурації та зручності адміністрування використовуються додаткові бібліотеки та інструменти. Jsonwebtoken (JWT) дає змогу створювати і перевіряти токени автентифікації, що забезпечує безпечну і масштабовану авторизацію для багатьох користувачів і різних рівнів доступу [9]. Бібліотека bcryptjs використовується для хешування паролів, що суттєво знижує ризик компрометації навіть у разі витоку бази даних, адже паролі ніколи не зберігаються у відкритому вигляді [2]. Dotenv дозволяє конфіденційно зберігати налаштування проекту, такі як паролі, ключі чи конфігурації, у зовнішньому файлі .env. Для захисту API від несанкціонованих запитів використовується бібліотека CORS, яка дозволяє обмежити доступ лише до дозволених джерел [3]. Нарешті, для інтеграції Node.js з PostgreSQL застосовується офіційний пакет pg, який підтримує роботу з пулом з'єднань, транзакціями, а також параметризованими запитами для запобігання SQL-ін'єкціям [4].

Завдяки такому підходу обраний стек технологій дає змогу створити сучасну, масштабовану, надійну і захищену серверну частину корпоративної системи управління проєктами, що повністю відповідає вимогам як корпоративного сектору, так і динамічним стандартам індустрії.

2.3 Засоби розгортання, налаштування та обслуговування серверної інфраструктури

У сучасних корпоративних інформаційних системах ефективне розгортання, налаштування та обслуговування серверної інфраструктури є однією з ключових складових успішної експлуатації й масштабування проєкту. Вибір інструментів та підходів здійснюється на основі порівняльного аналізу, врахування потреб проєкту, рекомендацій галузевих експертів і тенденцій

розвитку індустрії розробки веб-систем на основі Node.js та PostgreSQL. В цьому розділі розглядаються основні альтернативи для кожного аспекту інфраструктури, подано аргументацію вибору оптимальних рішень та наведено посилання на сучасні джерела.

Першочерговою задачею при створенні серверного середовища є організація керування залежностями. Серед найпопулярніших менеджерів пакетів для Node.js-проектів –npm, Yarn і pnpm. npm є стандартом де-факто у JavaScript-екосистемі, має широку підтримку, детальну документацію, активну спільноту та інтеграцію з більшістю інструментів CI/CD. Yarn розвивався як альтернатива з акцентом на підвищену швидкість та зручність роботи з великими монорепозиторіями. Останні версії npm скоротили різницю у продуктивності, запровадили підтримку lock-файлів, workspace та інші сучасні можливості, що раніше були перевагами Yarn [21]. pnpm відомий ефективним використанням диску завдяки механізму жорстких посилань, однак має меншу популярність та обмежену кількість готових інструкцій для інтеграції з CI/CD. Аналізуючи сучасні рекомендації експертів і порівняльні огляди, npm залишився найбільш універсальним рішенням, яке легко впроваджувати у будь-які проекти незалежно від їх масштабу [7]. Саме тому для розгортання Node.js-проекту у межах цієї системи управління проєктами було обрано npm.

Наступною важливою задачею є захист та гнучке керування конфіденційними даними, такими як паролі, ключі доступу до БД, API-токени. Основні підходи тут – використання явних конфігураційних файлів, корпоративних систем керування секретами (Vault, AWS Secrets Manager) або локальних файлів середовища з підвантаженням змінних через dotenv. Корпоративні системи (Vault, AWS Secrets Manager) надають найвищий рівень безпеки, підтримують аудит та автоматичну ротацію ключів, але відрізняються складністю налаштування і зазвичай застосовуються у великих хмарних чи мульти-регіональних проєктах [7]. Для невеликих та середніх команд найчастіше застосовується бібліотека dotenv – вона дозволяє розділити налаштування для різних середовищ, легко інтегрується з CI/CD і підтримується практично всіма

платформами для автоматизації розгортання. Змінні середовища зберігаються у .env-файлі, який виключається з публічних репозиторіїв (через .gitignore), що запобігає випадковому витоку конфіденційної інформації [3]. Для даного проекту такий підхід забезпечує оптимальне співвідношення безпеки, простоти та гнучкості.

Сучасні вимоги до DevOps практик диктують необхідність автоматизації розгортання й масштабування серверної інфраструктури. Історично такі задачі вирішувалися за допомогою Bash-скриптів або ручного адміністрування, проте зі зростанням складності систем такі підходи виявилися неефективними. Для комплексних проєктів застосовують системи конфігураційного менеджменту (Ansible, Chef, Puppet), але вони орієнтовані на великі компанії й потребують тривалого навчання персоналу [4]. Найбільш ефективним рішенням для Node.js та PostgreSQL-проектів визнано використання Docker або альтернативної технології контейнеризації. Docker дозволяє розгортати застосунки в ізольованих контейнерах з власними залежностями, мережевими налаштуваннями й середовищем виконання, що дає змогу уникати конфліктів версій та швидко масштабувати систему [23]. Всі конфігурації зберігаються у вигляді коду (наприклад, docker-compose.yml), що сприяє повторюваності процесів і спрощує інтеграцію з CI/CD-системами (GitHub Actions, GitLab CI). За даними сучасних оглядів, понад 80 % компаній, що працюють з Node.js, впровадили Docker як стандартний інструмент автоматизації [23].

Операційна система, що обслуговує серверне середовище, суттєво впливає на стабільність, безпеку й можливості подальшої підтримки проєкту. Серед найпопулярніших серверних дистрибутивів Linux – Ubuntu Server, Debian, CentOS/AlmaLinux/Rocky Linux, Red Hat Enterprise Linux. Ubuntu Server вирізняється регулярними оновленнями, зручністю встановлення й налаштування, активною спільнотою, багатою документацією та широкою підтримкою сучасних серверних технологій (PostgreSQL, Node.js, Docker тощо) [20]. Debian, хоча й дуже стабільний, має повільніший цикл оновлень і менш зручний для інтеграції сучасного ПЗ. CentOS/AlmaLinux/Rocky Linux були

популярні серед корпоративних користувачів, але після зміни політики підтримки CentOS дедалі більше компаній переходять на Ubuntu Server [22]. Red Hat Enterprise Linux надає розширену підтримку й SLA, але є платним рішенням, що не завжди виправдано для проєктів із обмеженим бюджетом [20]. Враховуючи співвідношення стабільності, простоти адміністрування, актуальності пакетів та інтеграції з DevOps-інструментами, для даного проекту було обрано Ubuntu Server як платформу для розгортання серверної частини.

Організація резервного копіювання та відновлення даних є одним із основних аспектів забезпечення безпеки та надійності корпоративної інформаційної системи. Серед сучасних підходів можна виділити ручне копіювання за допомогою інструментів pg_dump або pg_basebackup, використання спеціалізованих утиліт, таких як pgBackRest чи Barman, а також автоматизовані бекапи на стороні хмарних провайдерів (наприклад, Amazon RDS, Google Cloud SQL). Ручне копіювання залишається актуальним для невеликих проектів, однак для середніх і великих систем цей підхід обмежений можливостями щодо швидкості відновлення та безперервності роботи. Спеціалізовані утиліти (pgBackRest, Barman) дозволяють налаштувати регулярне повне та інкрементальне резервне копіювання, перевірку цілісності резервних даних, зберігання бекапів у різних сховищах (локальних чи хмарних), а також підтримують відновлення даних до певного моменту часу (Point-In-Time Recovery) [16].

Хмарні сервіси автоматизують процес резервного копіювання та підвищують рівень захисту, однак залежать від сторонньої інфраструктури та можуть мати вищу вартість для малих і середніх команд. У результаті аналізу сучасних інструментів, для проектів із використанням Node.js та PostgreSQL найчастіше рекомендується впровадження pgBackRest або Barman завдяки їхній гнучкості, надійності, простоті інтеграції та підтримці роботи у контейнеризованому середовищі.

Моніторинг і підтримка працездатності інфраструктури є важливою складовою сучасного серверного середовища. Існує цілий ряд класичних і

сучасних інструментів для збору метрик, логування та сповіщення: Nagios, Zabbix, Prometheus, Grafana, ELK Stack. Згідно з оглядами і практикою лідерів індустрії, оптимальною комбінацією для контейнеризованих Node.js-проектів є використання Prometheus (збір метрик) та Grafana (візуалізація) [8, 14, 15]. Це рішення дозволяє відслідковувати продуктивність сервісів, навантаження на сервери, стан баз даних і час реакції API, а також оперативно інформувати адміністраторів про критичні події. Для зберігання й аналізу логів можуть додатково використовуватись інструменти типу ELK Stack (ElasticSearch, Logstash, Kibana) або спеціалізовані сервіси (Papertrail, Loggly).

Підсумовуючи, для кожного ключового аспекту розгортання, налаштування й обслуговування серверної інфраструктури було проведено порівняльний аналіз сучасних технологій, розглянуто їхні переваги та недоліки у контексті Node.js та PostgreSQL-проекту, та аргументовано обрано оптимальні рішення. Менеджером залежностей обрано npm як стандарт JavaScript-екосистеми, для безпечної роботи з конфіденційними даними – бібліотеку dotenv. Як операційну систему обрано Ubuntu Server завдяки її стабільності, регулярним оновленням, гнучкості й підтримці DevOps-інструментів. Резервне копіювання та відновлення даних організовано за допомогою спеціалізованих утиліт pgBackRest або Barman. Така комбінація технологій забезпечує надійність, масштабованість, безпеку та зручність супроводу серверної частини корпоративної системи управління проєктами.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СЕРВЕРНОГО СЕРЕДОВИЩА

3.1 Розгортання серверної інфраструктури

Система корпоративного управління проектами, яка розроблялась у рамках даного проекту, орієнтована на використання в компаніях, де існує потреба в чіткій організації виконання завдань, контролі їх виконання та можливості керівників оперативно призначати завдання працівникам. Такі системи особливо актуальні для підприємств із багаторівневою структурою, де важливо забезпечити прозорість процесу постановки задач, моніторингу статусів та ефективної взаємодії між співробітниками та керівниками. Рішення, що реалізоване у даній роботі, дозволяє не лише підвищити керованість проектами, але й оптимізувати ресурси компанії та мінімізувати людський фактор при контролі завдань.

Першим кроком у розгортанні інфраструктури є встановлення операційної системи Ubuntu Server. В ході інсталяції задається основний користувач, який буде використовуватися для адміністрування системи. Одразу після встановлення виконуємо оновлення системних пакетів за допомогою команд sudo apt update && sudo apt upgrade -y.

Для підвищення безпеки доцільно створити окремого адміністративного користувача (наприклад, командою sudo adduser projectadmin) та надати йому права адміністратора через групу sudo (sudo usermod -aG sudo projectadmin). Також доцільно уникати використання облікового запису root для щоденних операцій.

Наступним етапом налаштовуємо захищений віддалений доступу до сервера через протокол SSH. Якщо сервер працює локально, SSH-сервер зазвичай уже встановлено, але при потребі його можна інсталювати через команду sudo apt install openssh-server. Для підвищення безпеки слід налаштувати авторизацію лише за допомогою SSH-ключів, заборонити прямий доступ користувачу root (PermitRootLogin no у файлі /etc/ssh/sshd_config) та, при

необхідності, змінити стандартний порт для з'єднання. Після змін служба SSH перезапускається командою sudo systemctl restart ssh. Такі налаштування суттєво знижують ризик несанкціонованого доступу до системи.

Мережеві налаштування передбачають призначення статичної IP-адреси, що забезпечує стабільний доступ до серверу. В Ubuntu для цього використовується система Netplan. Необхідно відредагувати відповідний конфігураційний файл, наприклад /etc/netplan/01-netcfg.yaml, де прописується інтерфейс, статична IP-адреса, шлюз та DNS-сервери. Зміни застосовуються командою sudo netplan apply. Коректність підключення перевіряється командами ip a та ping.

Після підготовки операційної системи і мережі здійснюється встановлення та налаштування веб-сервера. У цьому проекті для реверс-проксі обрано Nginx – один із найпотужніших і надійних веб-серверів, що ідеально підходить для роботи в парі з Node.js-застосунками. Nginx встановлюється командою sudo apt install nginx (рис. 3.1).

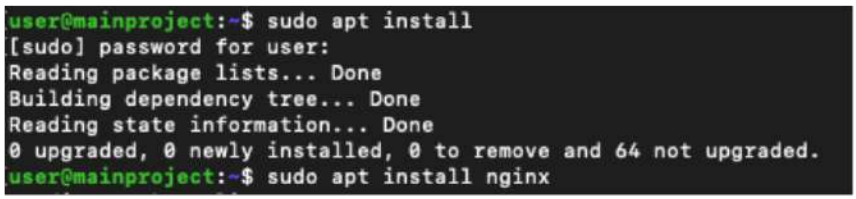

Рисунок 3.1 – Встановлення Nginx

Далі створюється окремий файл конфігурації, наприклад /etc/nginx/sites-available/project, у якому налаштовується проксування усіх HTTP-запитів до локального порту Node.js-сервісу (зазвичай 3000). Базовий файл конфігурації показаний на рисунку 3.2.

Рисунок 3.2 – Базовий файл конфігурації Nginx

Для забезпечення коректної маршрутизації запитів у проєкті використовується веб-сервер nginx, конфігураційний файл якого розташовано за шляхом /etc/nginx/sites-available/pms.com (додаток А). У цьому файлі налаштовано проксування запитів для фронтенд- та бекенд-частини системи, що дозволяє розділити обробку клієнтських та серверних запитів і забезпечити правильну взаємодію між ними. Сервер прослуховує вхідні HTTP-запити на порт 80 для домену pms.com та його піддомену www.pms.com. Всі запити, що починаються із /api/, перенаправляються на локальний бекенд-сервер Node.js, який працює на порту 3000. Для цього використовується блок location /api/, у якому за допомогою директиви proxy_pass http://localhost:3000/api; відбувається передача запитів на відповідний порт.

Всі інші запити (наприклад, відкриття головної сторінки, завантаження статичних ресурсів, роутинг SPA) обробляються фронтенд-частиною, що працює на Vite dev server на порту 5173. Це реалізовано за допомогою блоку location /, у якому прописано проксування на адресу http://localhost:5173/. Завдяки такому підходу весь трафік на домен спрямовується через nginx, який гнучко розподіляє запити між фронтендом і бекендом,

На першому етапі визначили, які саме сутності (таблиці) мають бути присутні в базі даних. Основними об’єктами системи є користувачі (users), задачі (tasks) та коментарі (comments). Це відповідає концепції організації корпоративних даних, де чітко відокремлюються ролі користувачів, їх взаємодія з бізнес-процесами (задачами), а також історія обговорень (коментарі). Для

кожної сутності визначаються основні поля, типи даних, ключі та зовнішні зв'язки, що забезпечує цілісність і структурованість даних (рис. 3.3).

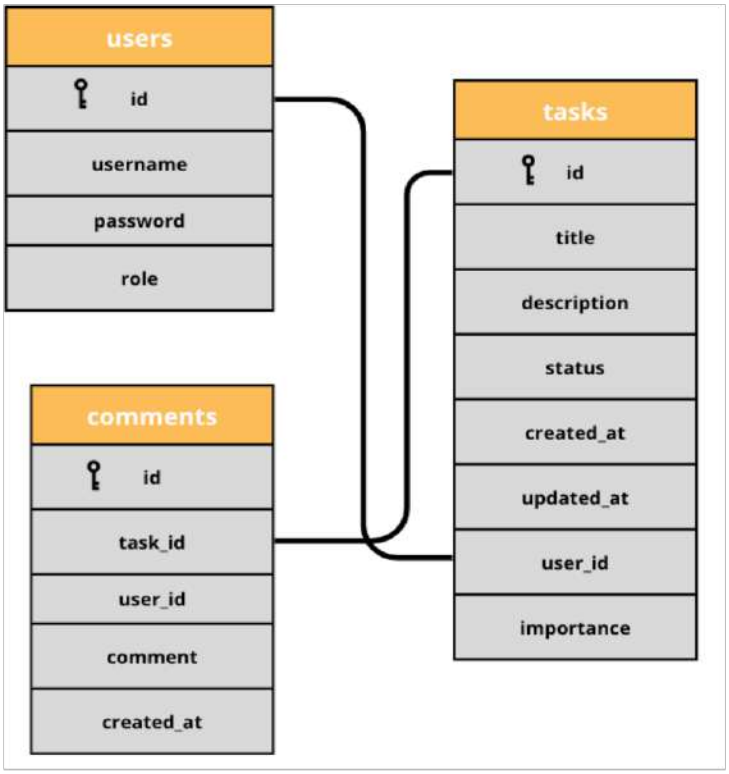

Рисунок 3.3 – Структура бази даних main_database

У структурі бази даних реалізовано класичні зв'язки типу «один до багатьох», де один користувач може бути відповідальним за декілька задач, що реалізується через зовнішній ключ user_id у таблиці tasks, а також може залишати численні коментарі, для чого у таблиці comments також зберігається user_id. Аналогічно, кожна задача може містити багато коментарів (зв'язок через task_id у таблиці comments). Таким чином, між таблицями users і tasks, users і comments, а також tasks і comments встановлені зв'язки «один до багатьох», що забезпечує гнучку і масштабовану структуру даних.

Далі, створюємо окрему базу даних (main_database) та окремого користувача (admin) (рис. 3.4), якому надаються лише необхідні привілеї (access rights) для роботи саме з цією базою.

```
main_database=# CREATE DATABASE main_database;
CREATE USER admin WITH ENCRYPTED PASSWORD 'пароль';
GRANT ALL PRIVILEGES ON DATABASE main_database TO admin;
```

Рисунок 3.4 – Створення БД та користувача admin

Такий підхід дозволяє мінімізувати ризики у разі компроментації облікових даних, а також підвищує гнучкість адміністрування – наприклад, у випадку розділення прав між розробниками, адміністраторами та іншими ролями.

Створюємо необхідні таблиці за структурою на рисунку 3.3, SQL-інструкціями (додаток Б). На етапі налаштування доступу конфігураційні файли pg_hba.conf та postgresql.conf змінюємо таким чином, щоб приймати підключення лише з локального хоста (127.0.0.1). Це додатково обмежує можливість несанкціонованого віддаленого доступу до даних. Для автентифікації обов'язково використовується шифрування паролів (md5).

Ще одним важливим компонентом організації життєвого циклу даних є резервне копіювання. У PostgreSQL для цього будемо використовувати утиліту pg_dump. Вона дозволяє створювати повні чи інкрементальні резервні копії бази даних у зручному для відновлення форматі. Запуск резервного копіювання автоматизується через системний планувальник (cron), що забезпечує регулярність і мінімізує ризик втрати інформації у разі технічних збоїв чи інших форс-мажорних обставин. Такий підхід забезпечує як відповідність політикам корпоративної безпеки, для цього було виконано наступні дії:

1) створено файл скрипту резервного-копіювання (додаток В);

2) за допомогою команди chmod +x надаємо права на виконання скрипту;

3) за допомогою задачника налаштовуємо циклічне виконання скрипту (рис. 3.5), скрипт виконуватиметься щоночі о 02:00.

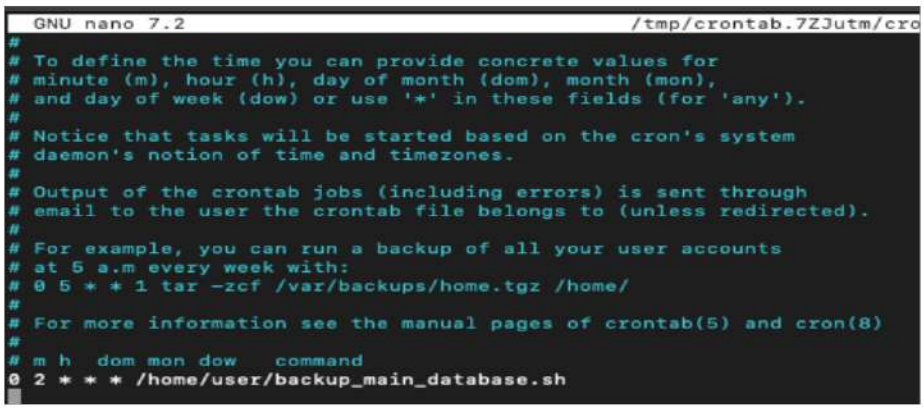

Рисунок 3.5 – Налаштування crontab

Для розгортання серверної частини застосунку необхідно встановити Node.js що виконується офіційним скриптом Nodesource. Після інсталяції Node.js та npm, переходять до каталогу backend-проекту, встановимо залежності (npm install), після чого запустимо сервіс (рис. 3.6).

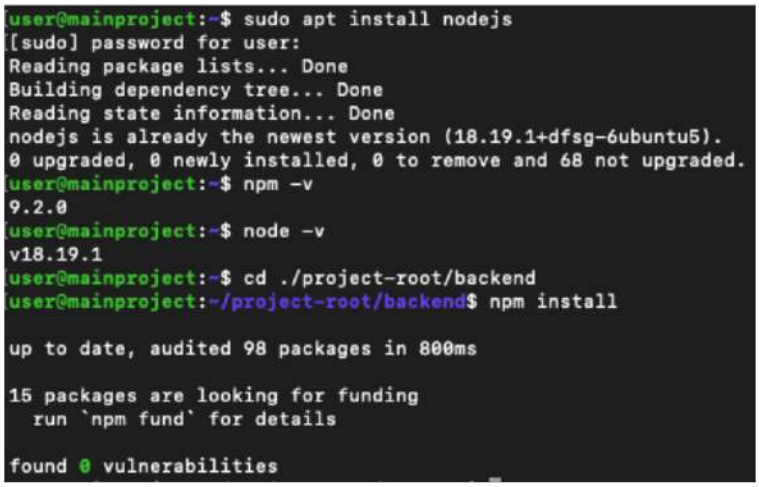

Рисунок 3.6 – Встановлення Node.js

Після встановлення необхідних залежностей одним із ключових кроків є налаштування конфігураційних файлів, які забезпечують коректну роботу серверної частини та взаємодію з іншими компонентами системи. Основними конфігураційними файлами у даному проєкті є:

– package.json – містить інформацію про проект, список зовнішніх бібліотек (залежностей), а також npm-скрипти для запуску, розробки та

тестування серверної частини. Наприклад, у нашому випадку для запуску бекенду використовується команда: npm run;

– .env – файл змінних, у якому зберігаються параметри підключення до бази даних, порт, на якому запускається сервер (рис. 3.7);

```
 GNU nano 7.2
DB_USER=admin
DB_HOST=192.168.0.111
DB_NAME=main_database
DB_PASSWORD=qwerty
DB_PORT=5432

JWT_SECRET=projectpms
```

Рисунок 3.7 – Файл змінних середовища серверу

– config/db.js – модуль для підключення до бази даних PostgreSQL (рис. 3.8). В цьому файлі логіка підключення реалізується через пакет pg, а всі необхідні дані зчитуються із .env за допомогою бібліотеки dotenv;

```
require('dotenv').config();
const { Pool } = require('pg');

const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: parseInt(process.env.DB_PORT,10),
});

console.log('Password type:',typeof process.env.DB_PASSWORD);

module.exports = pool;
```

Рисунок 3.8 – Підключення до PostgreSQL із використанням dotenv

– server.js – головний файл запуску сервера. В ньому відбувається підключення до маршрутизаторів (routes), middleware, а також стартує HTTP-сервер на вказаному в .env порту.

На завершення даного етапу було повністю розгорнуто серверну інфраструктуру корпоративної системи управління проєктами. Підготовлено та налаштовано всі необхідні компоненти: встановлено та сконфігуровано

операційну систему Ubuntu Server, веб-сервер Nginx, систему керування базами даних PostgreSQL і середовище виконання Node.js. Створено основну структуру бази даних, організовано захищене підключення до неї, налаштовано ключові конфігураційні файли проекту, а також реалізовано автоматизоване резервне копіювання даних. Серверна частина програмного комплексу повністю готова до подальшої інтеграції та проведення тестування системи.

3.2 Реалізація системи управління проєктами

Для розробки клієнтської частини було обрано: бібліотеку React для побудови інтерфейсів, інструмент Vite для швидкої збірки й розробки, а також React Router для організації навігації в SPA. Керування станом додатку реалізовано через Context API та кастомні React-хуки (рис. 3.9). Такий підхід забезпечує модульність, простоту подальшої підтримки. Весь фронтенд організовано як односторінковий застосунок (SPA), де маршрутизація між сторінками здійснюється на боці клієнта без повного перезавантаження сторінки.

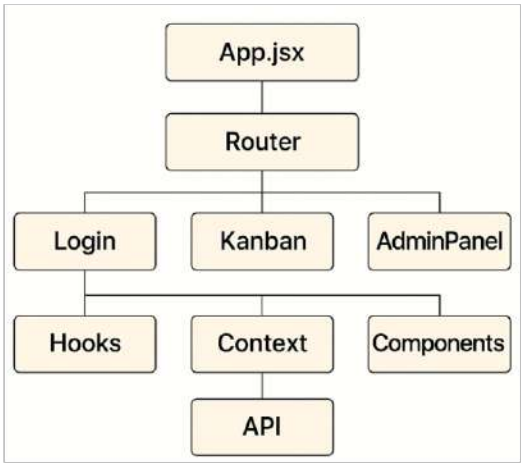

Рисунок 3.9 – Загальна схема архітектури клієнтської частини

Файлова структура проекту оптимізована під компонентний підхід, що підвищує повторюваність коду та полегшує масштабування функціоналу. Основні елементи виглядають так (рис. 3.10):

Рисунок 3.10 – Структура папки frontend у проекті

У файлі App.jsx підключаються маршрути та глобальні компоненти (додаток Г), а main.jsx є точкою входу, де ініціалізується React-додаток. У каталозі components/ містяться багаторазові компоненти: PrivateRoute (захист сторінок від неавторизованих користувачів), AdminRoute (доступ лише для адміністраторів). Папка hooks/ містить власні хуки, зокрема AuthContext для збереження стану авторизації та useTasks для роботи із задачами через API. Основні сторінки (Login, Kanban, AdminPanel) зберігаються у pages/. Окремі CSS-файли відповідають за стилізацію як окремих компонентів, так і всього додатку в цілому.

Головними сторінками інтерфейсу є: Login.jsx**,** Kanban.jsx. У сторінці авторизації із формою входу, дані передаються на бекенд, а отриманий токен автентифікації зберігається у контексті (AuthContext) або localStorage. Далі токен автоматично додається до захищених запитів для авторизації користувача (додаток Д). На основній сторінці для роботи користувача із задачами реалізовано інтерактивну канбан-дошку, яка відображає задачі у різних статусах, дозволяє їх створювати, редагувати, переносити, залишати коментарі, бачити відповідальних і дедлайни (рис. 3.11). Всі дії одразу синхронізуються з сервером.

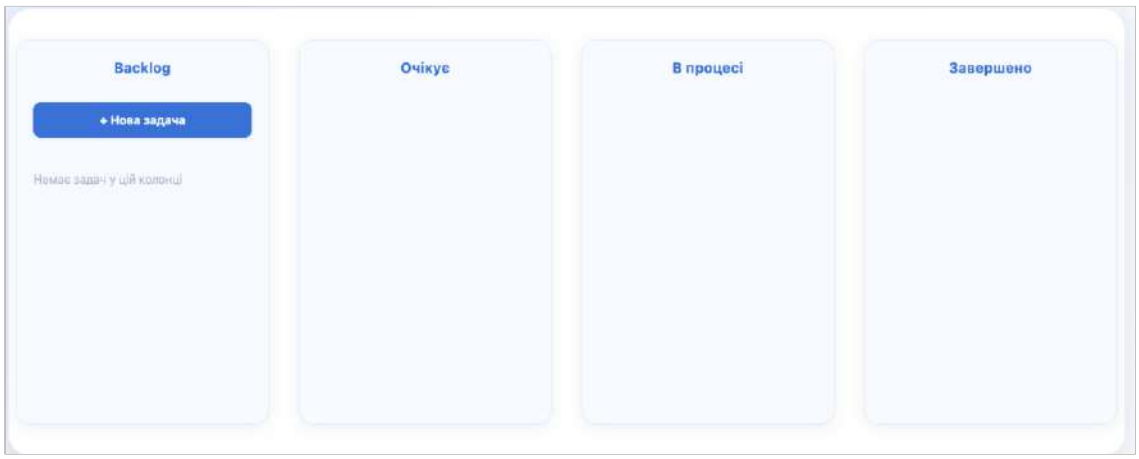

Рисунок 3.11 – Інтерфейс канбан-дошки

AdminPanel.jsx – сторінка для адміністратора з розширеними можливостями: створення та редагування користувачів, призначення ролей, перегляд і керування задачами усього проекту, а також контроль статусу виконання.

На цій сторінці адміністратор може передавати задачі між користувачами, стежити за активністю й результативністю команди. Для захисту доступу до окремих сторінок застосовуються спеціальні компоненти. PrivateRoute.jsx пропускає тільки залогінених користувачів, а AdminRoute.jsx лише користувачів із правами адміністратора (рис. 3.12). Усі перевірки прав здійснюються на підставі інформації із глобального контексту.

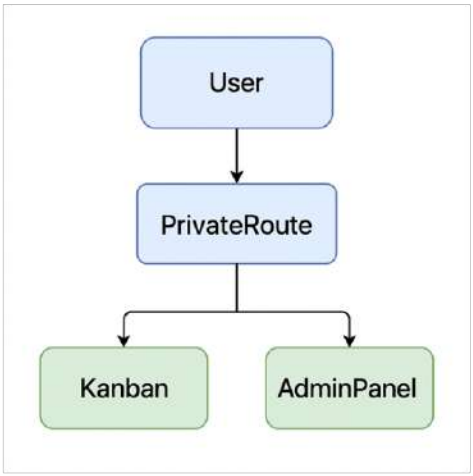

Рисунок 3.12 – Логіка контролю доступу на фронтенді

Всі дані в системі (користувачі, задачі, коментарі) отримуються та змінюються за допомогою HTTP-запитів до серверної частини (Node.js API). Для цього у хуках та утилітах проекту реалізовано стандартні запити (fetch або axios), які забезпечують роботу із такими ендпоінтами:

– GET /tasks отримання задач користувача;

– POST /tasks створення нової задачі;

– PUT /tasks/:id редагування задачі;

– DELETE /tasks/:id видалення задачі.

Та інші для роботи з користувачами та коментарями (рис. 3.13). Для захищених операцій у запитах використовується токен автентифікації (Bearer-token у headers).

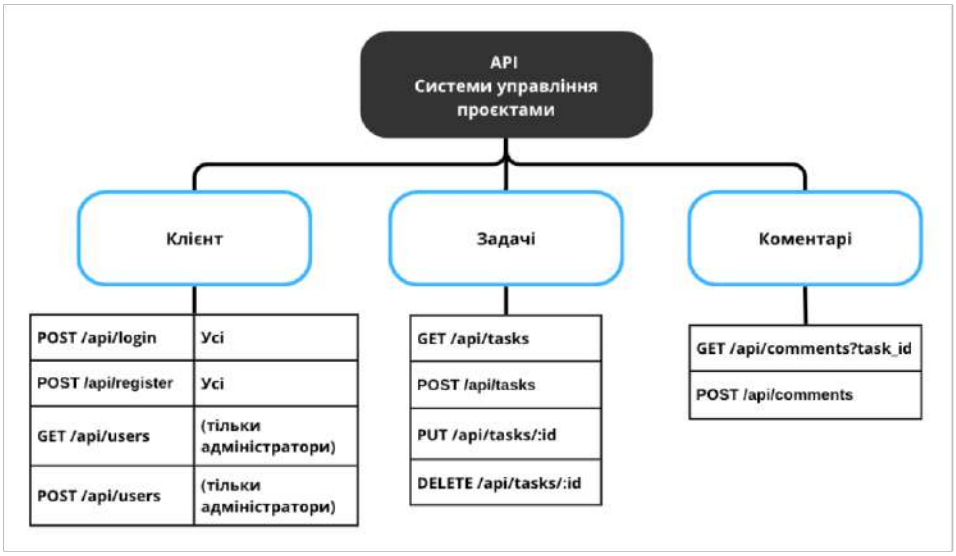

Рисунок 3.13 – Взаємодія між клієнтом та сервером через API

В результаті виконаної розробки клієнтської частини корпоративної системи управління проєктами вдалося реалізувати сучасний, функціональний та захищений веб-застосунок. Інтерфейс побудовано на основі компонентного підходу з використанням React, що забезпечило гнучкість структури, простоту масштабування та швидкість роботи застосунку. Всі ключові бізнес-процеси (авторизація, керування задачами, контроль доступу користувачів та адміністрування) – реалізовано через інтуїтивно зрозумілі сторінки й

компоненти, які тісно інтегровані з серверною частиною через стандартизовані REST API. Завдяки впровадженню механізмів захисту маршрутів та контролю прав доступу, система гарантує належний рівень безпеки та розмежування ролей користувачів.

3.3 Тестування та перевірка стабільності системи

Якість і надійність корпоративної системи управління проєктами значною мірою визначаються результатами тестування її функціональності, захищеності та стійкості до збоїв чи втрати даних. Перед впровадженням системи у виробниче середовище було проведено комплексну перевірку – від функціонального тестування REST API до оцінки відмовостійкості та працездатності механізмів реплікації даних у базі.

На першому етапі тестування основна увага приділялась перевірці роботи всіх основних API-сервісів системи – автентифікації, управління задачами та користувачами, обробки коментарів. Для цього використовувалися такі інструменти, як Postman і curl, які дозволяють імітувати запити від імені різних типів користувачів та оцінити поведінку системи при коректних і некоректних даних. Було детально перевірено вхід через ендпоінт /auth/login із валідними та хибними паролями, перевірена видача токена та реакція на помилки автентифікації (рис. 3.14).

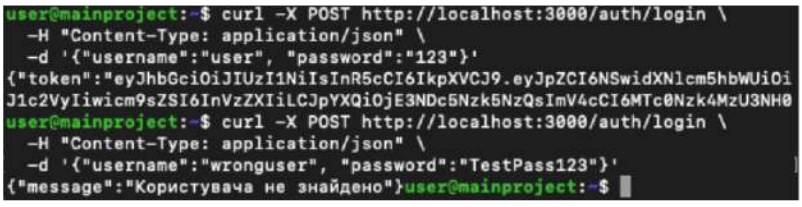

Рисунок 3.14 – Перевірка автентифікації користувачів

Аналогічно перевірялися операції з коментарями (отримання та додавання) та адміністративні функції для роботи з користувачами.

У процесі тестування кожен ендпоінт перевірявся на відповідність очікуваним результатам (додаток Е): повернення коректного коду статусу HTTP (200, 201, 400, 401, 403, 404), відповідність формату і вмісту даних у відповіді, адекватна реакція на помилки (наприклад, спроби редагувати неіснуючу задачу, видалити вже видалену, створити задачу без обов'язкових полів тощо). Окремо фіксувалися сценарії, що пов'язані з правами доступу: спроби отримати список задач без токена чи з токеном користувача, який не має відповідних прав, та доступ до адміністративних функцій звичайними користувачами (додаток Е). Всі API-кінцеві точки реагували згідно з логікою системи, не допускали витоку даних чи виконання дій без автентифікації.

Особлива увага приділялася питанням інформаційної безпеки, зокрема правильності реалізації хешування паролів та обробки токенів авторизації. У системі паролі користувачів зберігаються у базі даних лише у вигляді хешів, що підтверджується при перегляді записів у таблиці users: значення у полі password не співпадають із введеним паролем, а є хеш-рядком (рис. 3.15), отриманим за допомогою сучасного алгоритму bcrypt. При спробі здійснити логін із некоректним паролем система не повертає додаткової інформації про причину помилки, що унеможливлює підбір паролів за відповідями сервера.

```
main_database=# SELECT username, password FROM users;
 username |                          password
----------+------------------------------------------------------------
 admin    | $2b$10$akvv5ZKr1M2A2LgTGy6z0OdLCeeD8O7qpNee8yRtQVMmVG3U5be4W
 user     | $2b$10$uh0RIg9s1AlC2q7ewR9gNuL.atiAbOOn8Z8IvQ2PskkBKkpDV2p5O
 daryna   | $2b$10$4sWj3NwOx8w07cQP4.YfoerPA6QzPGJWwETWYsrgfA28HSFtCJAJi
 vlad     | $2b$10$pB7oMi0CDwmrO27ZI/81luDwueITqrwCxlQczJ4dNDeiJ3SF5MTUS
(4 rows)
```

Рисунок 3.15 – Відображення паролів у таблиці

Додатково тестувалися всі аспекти роботи з JWT-токенами. Після авторизації користувача токен зберігається на стороні клієнта (у localStorage), а при кожному захищеному запиті додається до заголовків Authorization. При спробі виконати запит із підробленим або простроченим токеном сервер повертає відповідь з кодом 401 (Unauthorized) (додаток Е), а доступ до захищених

ресурсів не надається. Була протестована поведінка системи у випадках видалення токена, підміни його на некоректний чи “старий” (прострочений), а також за відсутності токена. Усі випадки коректно обробляються серверною частиною (додаток Е).

Ще одним важливим напрямом тестування були перевірки політики CORS (Cross-Origin Resource Sharing) та захисту від типових веб-атак. Сервер налаштований на прийом запитів лише з дозволених джерел – під час розробки це localhost:5173. При спробі виконати запит із стороннього домену система повертає CORS-помилку, не передаючи дані у відповідь (рис. 3.16).

Рисунок 3.16 – Тестування виконання запиту із стороннього домену

Усі параметри CORS прописані у конфігураційних файлах серверної частини. Параметризовані SQL-запити, що використовуються у коді, а також валідація та санітизація введених даних мінімізують ризик SQL-ін’єкцій і XSS-атак.

Для перевірки захисту від SQL-ін’єкцій було виконано наступний тест на рисунку 3.17. У результаті система сприйняла введене значення як звичайний текст, і не було отримано жодних помилок чи витоку даних. Це свідчить про використання параметризованих запитів у серверній частині, що гарантує захист від SQL-ін’єкцій.

Рисунок 3.17 – Перевірка захисту від SQL-ін’єкцій

Однією з ключових вимог до корпоративних інформаційних систем є забезпечення їхньої надійності, стійкості до збоїв і готовності до роботи у режимі 24/7. Для досягнення цих цілей у проєкті було реалізовано механізм реплікації бази даних PostgreSQL, що дозволяє дублювати дані з основного сервера на резервний (standby) у реальному часі.

Налаштування реплікації відбувалося за принципом streaming replication – основний (primary/master) сервер постійно надсилає журнал змін (WAL – Write-Ahead Log) на додатковий (standby/replica), що забезпечує повну синхронізацію даних і можливість автоматичного перемикання (failover) у разі збою основного вузла.

Для цього на основному сервері PostgreSQL було створено окремого користувача, який має права лише на реплікацію. В конфігураційному файлі PostgreSQL (postgresql.conf) були змінені наступні параметри зображені на рисунку 3.18.

```
listen_addresses = '*'
wal_level = replica
max_wal_senders = 5
wal_keep_size = 128
hot_standby = on
```

Рисунок 3.18 – Конфігураційний файл postgresql.conf

Також у файлі контролю доступу (pg_hba.conf) дозволено підключення standby-сервера за IP.

Після внесення змін у конфігураційні файли та перезапуску основного сервера виконувалась ініціалізація standby-сервера. Його директорія даних була повністю очищена, а потім наповнена актуальною копією даних з primary-сервера за допомогою інструмента pg_basebackup. Під час виконання цієї операції використовувався створений раніше реплікаційний користувач, а сам процес відбувався у режимі streaming, тобто з подальшим автоматичним отриманням усіх нових змін із master. Далі, у налаштуваннях standby було

вказано інформацію про основний сервер та облікові дані для підключення, а сам сервер запущено у режимі очікування реплікації.

Для перевірки працездатності налаштованої реплікації були проведені практичні експерименти. На основному сервері до таблиці задач було додано новий запис за допомогою уже наявного інтерфейсу (рис. 3.19).

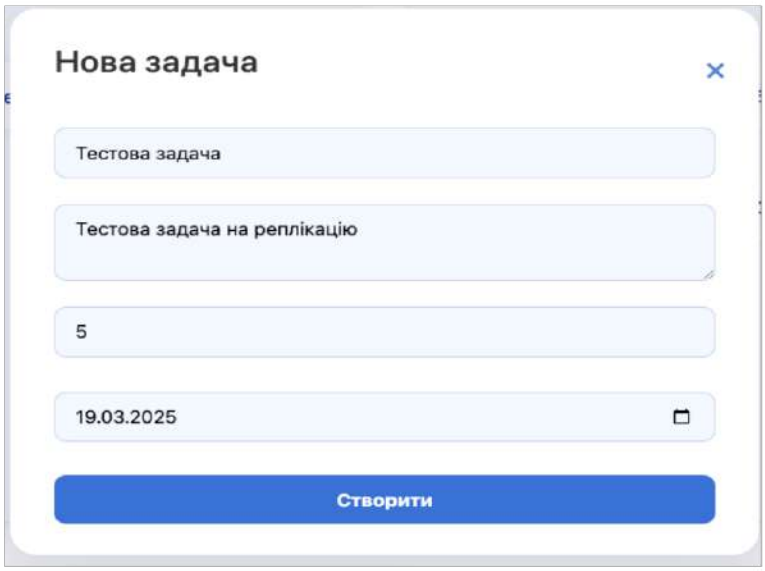

Рисунок 3.19 – Створення тестової задачі

Перед перевіркою репілкації, переглянемо наявність задачі на основному сервері, за допомогою SQL-запиту на рисунку 3.20.

Рисунок 3.20 – SQL-запит виведення усіх задач на головному сервері

У відповідь було отримано таблицю із наявними задачами, які є створені у системі. Після чого перевірено його наявність на standby-сервері – запис з'являвся автоматично через кілька секунд після вставки (рис. 3.21).

Рисунок 3.21 – Список задач на сервері репліки

У разі необхідності можливо підвищити standby до статусу primary, що дозволить не лише зчитувати дані, але й продовжити повноцінну роботу системи навіть при критичних збоях.

У результаті проведеного комплексного тестування корпоративної системи управління проєктами було підтверджено її працездатність, функціональну повноту та високу надійність. Перевірка API, автентифікації, управління задачами й користувачами показала відповідність усіх сервісів заявленій логіці, захищеність від несанкціонованого доступу й вразливостей, а також стійкість до помилок і некоректних даних. Тестування безпеки продемонструвало правильне хешування паролів, стійкість до атак типу SQL-ін'єкція та XSS, коректну обробку JWT-токенів і ефективну роботу політики CORS.

Впровадження механізмів реплікації на основі streaming replication у PostgreSQL забезпечило надійну синхронізацію даних між основним і резервним вузлами, що гарантує безперервність роботи системи навіть у разі критичних збоїв або втрати доступу до primary-сервера. Практичні перевірки додавання та передачі даних, а також моделювання ситуацій відмови підтвердили повну готовність системи до експлуатації у виробничому середовищі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було вирішено всі поставлені завдання.

Розроблено архітектуру корпоративної системи управління проєктами. Яка базується на сучасному стеку технологій та передбачає чітку взаємодію між клієнтською та серверною частинами, захищену структуру зберігання даних і можливість масштабування системи у майбутньому.

Реалізувано серверну частину із захищеним API для взаємодії з базою даних. Вона дозволяє виконувати всі необхідні операції з базою даних, керувати задачами, користувачами та коментарями, а також забезпечує належний рівень інформаційної безпеки. Для автентифікації користувачів і контролю доступу до ресурсів використано JWT-токени та рольову модель доступу.

Досліджено можливості впровадження реплікації та резервного копіювання у PostgreSQL. Налаштовано streaming replication, що забезпечує постійну синхронізацію даних між основним і резервним вузлами та гарантує збереження інформації у разі збоїв. Реалізовано регулярне автоматичне резервне копіювання за допомогою утиліти pg_dump та системного планувальника cron.

Спроєктовано структуру бази даних для зберігання інформації про задачі, користувачів та коментарі. Створено взаємозв'язані таблиці для зберігання інформації про задачі, користувачів та коментарі, визначено ключі, типи даних і зовнішні зв'язки, що забезпечують цілісність і структурованість даних.

Завдяки комплексному підходу, реалізована система є надійним, масштабованим та захищеним інструментом для автоматизації процесів управління проєктами, що відповідає сучасним вимогам бізнесу і може бути використана для впровадження на підприємствах різних галузей.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bcryptjs. *npm*. URL: https://www.npmjs.com/package/bcryptjs (date of access: 18.03.2025).

2. Begg C., Connolly T. Pearson EText for Database Systems: A Practical Approach to Design, Implementation, and Management, Global Edition. Pearson Education, Limited, 2022.

3. dotenv. *npm*. URL: https://www.npmjs.com/package/dotenv (date of access: 18.03.2025).

4. DSpace ELAKPI. *Репозитарій КПІ ім. Ігоря Сікорського*. URL: https://salo.li/1830ffe (дата звернення: 20.02.2025).

5. Express Node.js web application framework. *Express Node.js web application framework*. URL: https://expressjs.com/ (date of access: 10.03.2025).

6. Garcia-Molina H., Ullman J. D., Widom J. Database systems: the complete book. 3rd ed. Pearson Education, 2021. 1273 p.

7. Grafana OSS and Enterprise Grafana documentation. *Grafana Labs*. URL: https://grafana.com/docs/grafana/latest/ (date of access: 10.03.2025).

8. How to Introduce Redmine to Your Team: A Step-by-Step Guide. *RedmineUP Blog*. URL: https://salo.li/f47eAd5 (date of access: 14.03.2025).

9. JSON Web Token Introduction jwt.io. *JSON Web Tokens jwt.io*. URL: https://jwt.io/introduction/ (date of access: 14.03.2025).

10. Node.js Profiling Node.js Applications. *Node.js Run JavaScript Everywhere*. URL: https://salo.li/601022c (date of access: 14.03.2025).

11. Nodejs Security OWASP Cheat Sheet Series. *Introduction OWASP Cheat Sheet Series*. URL: https://salo.li/96B1f1f (date of access: 20.03.2025).

12. Node.js User Authentication Guide. *LoginRadius*. URL: https://salo.li/0C6f8F1 (date of access: 23.03.2025).

13. npm Docs. *npm Docs*. URL: https://docs.npmjs.com/ (date of access: 23.03.2025).

14. Overview Prometheus. *Prometheus Monitoring system & time series database*. URL: https://prometheus.io/docs/introduction/overview/ (date of access: 23.03.2025).

15. pgBackRest Reliable PostgreSQL Backup & Restore. *pgBackRest Reliable PostgreSQL Backup & Restore*. URL: https://pgbackrest.org/ (date of access: 23.03.2025).

16. PostgreSQL: Documentation. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/docs/ (date of access: 23.03.2025).

17. Postgres Tutorials | EDB. *EDB Postgres AI: Cloud Agility with Sovereign Control*. URL: https://www.enterprisedb.com/postgres-tutorials (date of access: 23.03.2025).

18. Project planning and scheduling software OpenProject. *OpenProject.org*. URL: https://salo.li/4F282C0 (date of access: 23.03.2025).

19. Red Hat Enterprise Linux operating system. *Red Hat We make open source technologies for the enterprise*. URL: https://salo.li/BE98d7a (date of access: 23.03.2025).

20. The CentOS Project. *The CentOS Project*. URL: https://www.centos.org/ (date of access: 23.03.2025).

21. Ubuntu Server documentation. *Ubuntu Server*. URL: https://documentation.ubuntu.com/server/ (date of access: 20.03.2025).

22. Welcome node-postgres. *Welcome node-postgres*. URL: https://node-postgres.com/ (date of access: 23.05.2025).

23. What is Docker. *Docker Documentation*. URL: https://docs.docker.com/get-started/docker-overview/ (date of access: 23.03.2025).

24. Блага Н. В. Управління проєктами: навчальний посібник. Львів : ЛьвДУВС, 2021. 156 с.

25. Войтенко О. С О. С. Управління проєктами. Київ : КНУБА, 2022. 276 с.

26. Основні принципи управління проєктами для початківців - Блог системи управління проектами Worksection. *Worksection*. URL: https://salo.li/E8b472f (дата звернення: 23.02.2025).

27. Рассел Дж., Нельсон Дж., Пфердехірт У. Технічне управління проектами в живому та геометричному порядку. Медісон : University of Wisconsin-Madison, 2021. 231 с.

28. Силабус навчальної дисципліни «управління проєктами». *Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука* | *MEGU*.URL: https://salo.li/e36a967 (дата звернення: 20.02.2025).

29. Цепенди І. Є., Кропельницької С. О. Керівництво з управління проектами розвитку інтерактивний навчальний посібник. Івано-Франківськ: Прикарпат. нац. ун-т ім. Василя Стефаника, 2021. 351 с.

30. Тишко М., Мельник К. Оптимізація продуктивності та відмовостійкості систем управління проектами через реплікацію серверів. *Програмне та апаратне забезпечення в інформаційних технологіях* : зб. тез доп. міжнар. наук.-практ. конф. молодих науковців та студентів, м. Луцьк. 6 травня, 2025. С.172-174.

ДОДАТКИ

Додаток А

Конфігураційний файл Nginx

```
server {
    listen 80;
    server_name pms.com www.pms.com;

     location /api/ {
        proxy_pass http://localhost:3000/api;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location / {
        proxy_pass http://localhost:5173/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

}
```

Додаток Б

SQL-інструкції для створення таблиць

```
-- створення таблиця із даними про користувача
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    role TEXT NOT NULL
);

-- створення таблиці для завдань
CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    description TEXT,
    status TEXT NOT NULL,
    deadline TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    importance INTEGER DEFAULT 0
);


-- створення таблиці для коментарів завдань
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    task_id INTEGER REFERENCES tasks(id) ON DELETE CASCADE,
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    comment TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Додаток В

Bash-скрипт для резервного копіювання БД

```
#!/bin/bash
# Каталог для зберігання бекапів
BACKUP_DIR="/home/user/db_backups"
DB_NAME="main_database"
DB_USER="admin"
DB_HOST="127.0.0.1"

# Створити папку для бекапів, якщо ще не існує
mkdir -p "$BACKUP_DIR"

# Створення дампу
PGPASSWORD='qwerty' pg_dump -U $DB_USER -h $DB_HOST $DB_NAME >
"$BACKUP_DIR/${DB_NAME}_$(date +%F_%H-%M-%S).sql"
```

Додаток Г

Код файлу App.jsx

```jsx
import AdminPanelPage from './pages/AdminPanelPage';
import AdminUsersPage from './pages/AdminUsersPage';
import Kanban from './pages/Kanban';
import Login from './pages/Login';
import PrivateRoute from './components/PrivateRoute';
import AdminRoute from './components/AdminRoute';
import { useAuth } from './hooks/useAuth';
import { Routes, Route, Navigate } from 'react-router-dom';

function App() {
  const { user, login, loading } = useAuth();

  if (loading) {
    return <p style={{ textAlign: 'center' }}>Завантаження...</p>;
  }

  return (
    <Routes>
      <Route
        path="/login"
        element={user ?
          (user.role === 'admin' ? <Navigate to="/admin" /> : <Navigate to="/kanban" />)
          : <Login onLogin={login} />}
      />
      <Route
        path="/kanban"
        element={
          <PrivateRoute>
            <Kanban />
          </PrivateRoute>
        }
      />
      <Route
        path="/admin"
        element={
          <AdminRoute>
            <AdminPanelPage />
          </AdminRoute>
        }
      />
      <Route
        path="/admin/users"
        element={
          <AdminRoute>
            <AdminUsersPage />
          </AdminRoute>
        }
      />
      <Route path="*" element={<Navigate to="/login" />} />
    </Routes>
  );
}

export default App;

```

Додаток Д

Код файлу Login.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const Login = ({ onLogin }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      const res = await fetch('/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
      });

      const text = await res.text();

      if (!res.ok) {
        console.error('❌ Помилка логіну:', res.status, text);
        setError('Помилка входу: ' + text);
        return;
      }

      const data = JSON.parse(text);
      localStorage.setItem('token', data.token);
      onLogin();

      // Парсимо роль з JWT-токена
      const user = JSON.parse(atob(data.token.split('.')[1]));

      if (user.role === 'admin') {
        navigate('/admin');
      } else {
        navigate('/kanban');
      }
    } catch (err) {
      console.error('❌ Помилка входу:', err);
      setError('Помилка мережі або сервера');
    }
  };

  return (
    <div style={{ maxWidth: '400px', margin: '2rem auto' }}>
      <h2>Вхід</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Ім'я користувача"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required
        /><br />
        <input
          type="password"
```

```
          placeholder="Пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        /><br />
        <button type="submit">Увійти</button>
      </form>
      {error && <p style={{ color: 'red' }}>{error}</p>}
    </div>
  );
};

export default Login;
```

Додаток Е

Тестування API

Перевірка авторизації

1) Успішна авторизація

```
user@mainproject:~$ curl -X POST http://localhost:3000/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"user","password":"123"}'
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NSwidXNlcm5hb
iwicm9sZSI6InVzZXIiLCJpYXQiOjE3NDgxNjcwODgsImV4cCI6MTc0ODE3MDY4OH0.V
xOgPYjDKB-UmhMXz4OrmJHGktInO3Q"}user@mainproject:~$
```

2) Введення неправильного паролю

```
user@mainproject:~$ curl -X POST http://localhost:3000/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"user","password":"невірний"}'
{"message":"Невірний пароль"}user@mainproject:~$
```

3) Неіснуючий користувач

```
user@mainproject:~$ curl -X POST http://localhost:3000/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"impossible_user","password":"невірний"}'
{"message":"Користувача не знайдено"}user@mainproject:~$
```

Отримання задач

1) Спроба отримати задачі без токену

```
user@mainproject:~$ curl -i http://localhost:3000/tasks
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
```

2) Спроба отримати задачі з неправильним токеном

```
user@mainproject:~$ curl -i http://localhost:3000/tasks -H "Authorization: Beare
r fake_token"
HTTP/1.1 403 Forbidden
X-Powered-By: Express
```

3) Отримання задач із реальним токеном

```
user@mainproject:~$ curl -i http://localhost:3000/tasks -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
pZCI6NSwidXNlcm5hbWUiOiJ1c2VyIiwicm9sZSI6InVzZXIiLCJpYXQiOjE3NDgxNjY0NDUsImV4cCI6MTc0ODE3MDA0NX0.naOR9Mw_NgrRUtZfihoaGTDVz
CKWp7BqCPIouSXuNr0"
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:5173
Vary: Origin
Access-Control-Allow-Credentials: true
Content-Type: application/json; charset=utf-8
Content-Length: 518
ETag: W/"206-53IzIHhClapAY2vjFTMBtDY0p7c"
Date: Sun, 25 May 2025 10:09:33 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[{"id":13,"title":"Зробити звіт","description":"Взяв в роботу ","status":"in-progress","deadline":"2025-05-21T00:00:00.000
Z","created_at":"2025-05-19T18:52:59.553Z","updated_at":"2025-05-25T09:11:19.978Z","user_id":5,"importance":10,"username":
"user"},{"id":48,"title":"задача репліка","description":"","status":"todo","deadline":"2025-05-31T00:00:00.000Z","created_
at":"2025-05-25T09:12:07.756Z","updated_at":"2025-05-25T10:06:01.607Z","user_id":5,"importance":7,"username":"user"}]user@
```