МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 03.00.00.000 ПЗ

Група ІПм-23-2

Клемешов Денис

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Клемешов Денис Олександрович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми створення веб-застосунків для зберігання та організації інформації

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Клемешов Д.О.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Лютак Ігор Зіновійович, професор

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Клемешову Денису Олександровичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та алгоритми створення веб-застосунків для зберігання та організації інформації”

керівник проекту (роботи) Лютак Ігор Зіновійович, професор

затверджені наказом закладу вищої освіти від „22” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Архітектура, моделі та алгоритми функціонування систем для зберігання та організації інформації

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

1. Теоретичні відомості про системи для зберігання та організації інформації
2. Аналіз сучасних методів і технологій інтеграції даних у централізовані системи управління інформацією
3. Алгоритм організації та зберігання інформації та огляд існуючих програмних засобів
4. Розробка алгоритму та програмної реалізації системи для організації та зберігання інформації

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Canvas модель для проекту (рис. 3.2, ст. 58)
2. UML-діаграма станів для процесу Реєстрації (рис. 3.3, ст. 59)
3. Структура папок (рис. 3.6, ст. 64)
4. Схема бази даних (рис. 3.17, ст. 73)
5. Модель Page (рис. 3.38, ст. 84)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	20.09.2024	виконано
2	Аналіз сучасних технологій зберігання та організації інформації	01.10.2024	виконано
3	Способи забезпечення безпечної передачі даних у мережі інтернет	12.10.2024	виконано
4	Дослідження алгоритмів зберігання та організації інформації та огляд існуючих рішень	25.10.20234	виконано
5	Формулювання вимог та алгоритмів функціонування системи	05.11.2024	виконано
6	Програмна реалізація рішення	22.11.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 106 с., 59 рис., 40 джерело.

Тема: Моделі, методи та алгоритми створення веб-застосунків для зберігання та організації інформації.

Об’єкт дослідження: сучасні системи управління інформацією, алгоритми та моделі, що лежать в їх основі, з акцентом на ефективність, масштабованість та гнучкість.

Мета роботи: дослідження та аналіз, розробка моделей та алгоритмів створенн веб-застосунків для зберігання та організації інформації.

Предмет дослідження: моделі методи та алгоритми систем зберігання та організації інформації, шляхи покращення взаємодії користувача з системою.

Результати дослідження:

Проведено глибокий аналіз існуючих систем керування інформацією, на основі отриманих даних розроблено власну архітектуру та алгоритм її роботи.

Висновок:

В результаті досліджень було створено власний застосунок для зберігання та організації інформації, який відповідає сучасним стандартам та вирішує проблеми управління інформацією.

ВЕБ-ЗАСТОСУНОК, СИСТЕМА ДЛЯ ЗБЕРІГАННЯ ТА ОРГАНІЗАЦІЇ ІНФОРМАЦІЇ, АЛГОРИТМ, АРХІТЕКТУРА, ВЕБ-СЕРВЕР, МОДЕЛІ.

ANNOTATION

Master's work: 106 p., 59 fig, 40 sources.

Topic: Models, methods and algorithms for creating web applications for storing and organizing information.

Object of research: modern information management systems, algorithms and models underlying them, with a focus on efficiency, scalability and flexibility.

Purpose: research and analysis, development of models and algorithms for creating web applications for storing and organizing information.

Subject of research: models, methods and algorithms of information storage and organization systems, ways to improve user interaction with the system.

Research results:

An in-depth analysis of existing information management systems was carried out, and based on the data obtained, our own architecture and algorithm for its operation were developed.

Conclusion:

As a result of the research, we created our own application for storing and organizing information that meets modern standards and solves information management problems

WEB APPLICATION, SYSTEM FOR STORING AND ORGANIZING INFORMATION, ALGORITHM, ARCHITECTURE, WEB SERVER, MODELS.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI (Artificial Intelligence) – штучний інтелект; технологія, що імітує людські когнітивні функції для аналізу даних, прийняття рішень та автоматизації.

UML (Unified Modeling Language) – уніфікована мова моделювання; стандарт для опису, проектування та документування програмних систем.

API (Application Programming Interface) – інтерфейс прикладного програмування; набір інструментів та протоколів для взаємодії між різними програмними компонентами.

SQL (Structured Query Language) – мова структурованих запитів, що використовується для роботи з базами даних.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту; стандартний протокол для передачі даних у мережі Інтернет.

JSON (JavaScript Object Notation) – текстовий формат для обміну даними, який часто використовується в веб-додатках.

REST (Representational State Transfer) – архітектурний стиль для проектування мережевих API на основі протоколу HTTP.

CRUD (Create, Read, Update, Delete) – набір основних операцій, які виконуються з даними в системах управління базами даних.

CI/CD (Continuous Integration/Continuous Deployment) – практика автоматизації розробки, тестування та розгортання програмного забезпечення.

TypeORM – об'єктно-реляційний мапер для TypeScript і JavaScript, що використовується для роботи з базами даних.

AWS S3 (Amazon Simple Storage Service) – хмарний сервіс для зберігання об'єктів даних, розроблений Amazon Web Services.

Elasticsearch – пошуковий та аналітичний механізм для обробки великих обсягів даних у реальному часі.

ВСТУП

Актуальність роботи

У сучасному суспільстві, що базується на даних, ефективна організація та зберігання інформації набуває вирішального значення. Вимоги масштабованості, безпеки та дизайну, орієнтованого на користувача, не задовольняються багатьма існуючими варіантами. Метою цього дослідження є розробка потужної централізованої системи, яка вирішує ці проблеми і гарантує цілісність даних, легкий доступ до них та організаційну ефективність.

Порівняння роботи з відомими розв'язаннями проблеми

На відміну від існуючих методів, це дослідження пропонує унікальне поєднання найсучасніших алгоритмів, інтеграції штучного інтелекту та інтуїтивно зрозумілих користувацьких інтерфейсів. Запропоноване рішення використовує модульний дизайн, що забезпечує гнучкість і майбутню масштабованість на відміну від багатьох існуючих систем, які покладаються на негнучкі структури даних і застарілі методи. Це відрізняє його від традиційних платформ, яким може бракувати складної аналітики та гнучкості.

Мета і задачі дослідження

Метою цього дослідження є аналіз, розробка моделей та механізмів веб-застосунків зберігання та організації інформації. Основними завданнями є аналіз сучасних технологій зберігання даних, створення алгоритмів для максимізації пошуку даних та тестування системи в ряді реальних ситуацій.

- дослідження мети включало наступні задачі:
- вивчення сучасних методів і стратегій організації та зберігання інформації
- створення теоретичної бази для розробки інформаційних систем
- розробка архітектури системи
- забезпечення конфіденційності, безпеки та цілісності даних

- розуміння програмного продукту
- тестування та оцінка ефективності системи

Об’єктом дослідження є сучасні системи управління інформацією, алгоритми та моделі, що лежать в їх основі, з акцентом на ефективність, масштабованість та гнучкість.

Предметом дослідження є моделі методи та алгоритми систем зберігання та організації інформації, шляхи покращення взаємодії користувача з системою

Методи дослідження Системний дизайн, тестування програмного забезпечення, аналіз літератури та моделювання алгоритмів - ось деякі з теоретичних і практичних методів дослідження, використаних у цьому дослідженні. Крім того, для оцінки ефективності запропонованих засобів використовуються бенчмаркінг і порівняльний аналіз.

Наукова новизна одержаних результатів

Наукова оригінальність цього дослідження полягає в побудові модульної, масштабованої архітектури та застосуванні штучного інтелекту для покращення організації та пошуку інформації. Запропоновані алгоритми та архітектура системи забезпечують значні покращення порівняно з традиційними методами, привносячи свіжі точки зору в сферу управління даними.

Практичне значення одержаних результатів.

Можливості використання системи в корпораціях, академічних установах та державних установах ілюструють практичну цінність отриманих результатів. Забезпечуючи легкий доступ до структурованих даних, запропоноване рішення гарантує краще прийняття рішень, більш ефективну командну роботу та зменшення операційної неефективності.

Особистий внесок

Від концепції до реалізації я особисто брав участь у проектуванні, розробці та тестуванні системи. Щоб гарантувати ефективність, надійність і зручність системи, я провів глибоке дослідження, розробив основні алгоритми і провів велику кількість тестувань. В результаті цієї роботи я значно вдосконалив технології управління інформацією.

Структура магістерської роботи.

Магістерська робота викладена на 106 сторінках друкованого тексту, який складається з вступу, чотирьох розділів, висновків, списку використаних джерел (40 найменувань). Робота містить 59 рисунків.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОСНОВНІ ВИМОГИ ДО СИСТЕМИ

1.1 Огляд концепцій систем для зберігання та організації інформації

1.1.1 Цінність організаційних та інформаційних систем зберігання даних

У сучасній інформаційній економіці дані стали життєво важливим ресурсом для бізнесу. Ефективна організація та адміністрування даних стає все більш складним завданням, оскільки компанії розширюються і продукують величезні обсяги даних. Брак прозорості, неефективне прийняття рішень та неефективність можуть бути наслідком того, що інформація розкидана або зберігається в декількох не пов'язаних між собою системах. З іншого боку, ефективна система зберігання інформації гарантує, що відповідні дані будуть легко доступні в разі потреби, заохочує командну роботу і знижує ймовірність помилок, спричинених застарілими або неузгодженими даними.

Доступ до інформації в режимі реального часу стає все більш важливим на сучасному робочому місці. Системи, які полегшують обмін, редагування та оновлення документів, є важливими для команд, які працюють віддалено або географічно віддалені. Ефективне управління інформацією пов'язане з певними труднощами, особливо коли обсяги даних зростають, а організації стають дедалі складнішими.

Цінність ефективної системи організації та зберігання інформації демонструється її здатністю:

- Зробити інформацію доступною: За потреби співробітники можуть швидко отримати необхідну інформацію та документи.
- Сприяти співпраці: Групи можуть спільно працювати над документами, надавати коментарі та миттєво змінювати дані.
- Зменшення дублювання: Співробітники працюють з точною, актуальною інформацією, мінімізуючи надлишкові та застарілі дані.

- Спрощення робочих процесів: Використовуючи систему, яка допомагає керувати даними та організовувати їх відповідно до робочих процесів, команди можуть працювати більш ефективно.
- Підвищення безпеки даних: Організації можуть захистити конфіденційні дані, контролюючи, хто має доступ до певної інформації за допомогою ефективного управління дозволами.

1.1.2 Традиційні методи зберігання інформації

Традиційні рішення для зберігання інформації були переважно фізичними: картотеки та паперові записи були стандартною відповіддю для бізнесу. Ці методи добре підходили для невеликих компаній, але вони мали багато недоліків, зокрема обмежений фізичний простір, проблеми з пошуком конкретної інформації та ручна праця, необхідна для організації та класифікації даних. Попит на дедалі складніші системи для обробки величезних обсягів даних зростав разом з економічним розвитком.

Організації почали переходити від фізичних систем зберігання документів до цифрових рішень для зберігання файлів. Організації змогли зберігати дані більш ефективно завдяки таким технологіям, як жорсткі диски, мережеві диски зі спільним доступом і ранні хмарні системи (наприклад, Dropbox і Google Drive). Однак ці системи мали певні недоліки, а саме:

- Фрагментація: Командам було складно знаходити файли та працювати над ними разом, оскільки інформація все ще була розпорошена між кількома платформами.
- Контроль версій: Командам часто доводилося вручну відстежувати зміни, що ускладнювало управління численними версіями документів або файлів.
- Проблеми зі співпрацею: Ці платформи часто потребували сторонніх додатків для спілкування та редагування документів, і їм бракувало інтегрованих засобів для співпраці в режимі реального часу.

В результаті цих обмежень були створені більш досконалі системи управління знаннями (СУЗ), які покликані впоратися з труднощами управління документами, контролем версій і спільною роботою на централізованій платформі.

1.1.3 Розвиток систем управління знаннями

Організації почали впроваджувати системи управління знаннями (СУЗ) як способи зберігання, управління та впорядкування інформації в результаті розвитку інтернету та зростаючого попиту на цифрову трансформацію. Ці системи пропонують ресурси для аналізу даних, управління документами та командної роботи, а також централізують знання всередині компанії. Notion, SharePoint і Confluence - ось кілька прикладів відомих СУЗ.

Основне призначення цих систем полягає в наступному:

- Централізоване зберігання документів: Всі дані та документи зберігаються в одній системі для зручної організації та доступу до них.
- Контроль версій: Відстежуючи зміни в документах, ці системи дозволяють командам працювати над останньою версією, гарантуючи при цьому збереження попередніх ітерацій.
- Інструменти для спільної роботи: Команди можуть легко співпрацювати завдяки таким функціям, як редагування в режимі реального часу, коментування та сповіщення.
- Безпека та контроль доступу: Дозволяючи авторизованим користувачам редагувати, переглядати та ділитися інформацією, дозволи можна налаштувати так, щоб обмежити доступ до конфіденційних даних.
- Функції пошуку: Команди можуть швидко знаходити документи та інформацію у величезній базі даних завдяки вдосконаленим функціям пошуку.

Незважаючи на те, що СУЗ стала важливим інструментом для багатьох організацій, їм все ще доводиться стикатися з такими проблемами, як кастомізація, інтеграція інструментів та підтримка документації в актуальному стані. Незважаючи на свої переваги, багато сучасних систем СУЗ все ще не здатні автоматизувати

повторювані процеси, такі як оновлення застарілих документів, і задовольнити потреби сучасних команд у спілкуванні в режимі реального часу.

1.1.4 Важливі характеристики сучасних систем зберігання інформації

Системи для організації та зберігання інформації повинні відповідати ряду важливих організаційних вимог. Організовувати та зберігати дані стає все складніше, оскільки фірми розширюються і виробляють все більше інформації. Щоб успішно впоратися з цими проблемами, сучасні системи повинні мати такі характеристики:

- Централізоване зберігання: Уся інформація, записи та дані повинні зберігатися в одній системі, до якої користувачі мають вільний доступ. Централізація підвищує ефективність, оскільки позбавляє користувачів необхідності шукати різні фрагменти інформації на кількох платформах.
- Контроль версій: Контроль версій гарантує, що всі працюють над останньою версією документа, і дозволяє командам відстежувати зміни в ньому. Це дозволяє уникнути непорозумінь і знижує ймовірність помилок, спричинених застарілою інформацією.
- Співпраця та комунікація: Команди можуть співпрацювати ефективніше, коли вони мають доступ до функцій спільної роботи, таких як потокові дискусії, редагування в режимі реального часу та коментування. Обмін даними та документами спрощується і стає більш прозорим.
- Автоматизація: Команди можуть значно зменшити своє робоче навантаження, автоматизувавши такі завдання, як оновлення документів, нагадування про застарілі документи та збереження старих даних. Крім того, це гарантує, що документи залишатимуться актуальними і не потребуватимуть постійного ручного контролю.
- Пошук і вилучення: У великомасштабних системах складні пошукові можливості мають вирішальне значення. Навіть при величезних обсягах контенту користувачі повинні мати можливість легко знаходити документи, дані та метадані.

- Безпека та контроль доступу: наявність надійних систем безпеки та управління дозволами має вирішальне значення для компаній, які працюють з конфіденційними даними.
- Масштабованість: Зі збільшенням обсягу даних організації та кількості користувачів сучасна інформаційна система повинна мати можливість рости разом з ними. Це гарантує, що система буде продовжувати добре функціонувати в міру того, як компанія змінюватиметься.

1.1.5 Місце машинного навчання та штучного інтелекту в організації інформації

Використання штучного інтелекту (ШІ) та машинного навчання (МН) є основною сферою інновацій у сучасних системах зберігання та організації інформації. Автоматизуючи численні завдання управління інформацією, ці технології можуть підвищити інтелектуальність і адаптивність систем. Наприклад, навіть якщо запит користувача нечіткий або недостатній, пошукові системи на основі ШІ можуть зрозуміти його контекст і повернути більш релевантні результати.

Класифікація та тегування документів - ще одне можливе застосування штучного інтелекту. Користувачі могли б легше знаходити відповідні елементи, якби алгоритми штучного інтелекту автоматично класифікували документи відповідно до їхнього змісту. Аналогічно, ШІ може аналізувати вміст документа і рекомендувати зміни або вказувати на застарілу інформацію, яку потрібно оновити. Впроваджуючи ці технології, можна ще більше підвищити точність та ефективність своїх процедур управління інформацією.

Організація може отримати ряд важливих переваг, впровадивши сучасну систему зберігання та організації інформації:

- Підвищення продуктивності: Співробітники можуть більше зосередитися на своїх основних обов'язках і витрачати менше часу на пошук та оновлення інформації завдяки спрощенню процесу документообігу, скороченню непотрібної роботи та заохоченню до співпраці.

- Краще прийняття рішень: Особи, які приймають рішення, можуть приймати кращі рішення на основі найкращих доступних даних, коли вони мають простий доступ до актуальної та точної інформації, що підвищує загальну ефективність корпоративної стратегії.
- Покращена співпраця: Працюючи разом у режимі реального часу та легко обмінюючись інформацією, команди можуть співпрацювати більш ефективно, незалежно від того, де вони знаходяться. Це гарантує, що всі згодні з рішенням, і зменшує розбіжності, які часто виникають в організаціях.
- Покращена безпека даних: Потужна система контролю доступу гарантує захист приватних даних, дозволяючи авторизованим користувачам працювати разом без обмежень. В результаті організаціям простіше підтримувати конфіденційність і дотримуватися законів про захист даних.
- Економічна ефективність: Компанії можуть заощадити час і гроші на оновленні та підтримці своєї документації, використовуючи послуги автоматизації. Крім того, компанії можуть заощадити гроші, відмовившись від використання різних платформ і програмних інструментів, централізувавши всю інформацію в єдиній системі.

1.1.6 Зберігання інформації та проблеми організації

Організації стикаються з низкою перешкод, коли справа доходить до організації та збереження інформації, навіть незважаючи на очевидні переваги сучасних систем. Серед цих труднощів можна виділити наступні:

- Перевантаження даних: Небезпека інформаційного перевантаження зростає, коли компанії накопичують величезні обсяги даних. Просіювання величезних обсягів даних у пошуках потрібної інформації може бути неприємним і трудомістким процесом, який забирає багато часу. Щоб допомогти клієнтам швидко знаходити потрібні їм дані,

системи пропонують надійні можливості пошуку та варіанти класифікації.

- Фрагментація інформації: Організації часто зберігають інформацію в різних системах, що призводить до фрагментації даних, до яких важко отримати доступ і якими важко керувати. Об'єднуючи всі дані в єдину платформу, централізований репозиторій вирішує цю проблему і полегшує доступ до документів та їх оновлення для всієї команди.
- Адаптація користувачів: Співробітники, які звикли до застарілих інструментів і процедур, можуть заперечувати проти впровадження нової системи управління інформацією. Пропонуючи інтуїтивно зрозумілий, зручний інтерфейс, який не вимагає навчання і простий у впровадженні, система зменшує ці труднощі.
- Інтеграція зі старими системами: Багато компаній продовжують використовувати застарілі системи, що може ускладнити їх інтеграцію з сучасними системами управління інформацією. Завдяки адаптивним функціям, компанії можуть легко інтегрувати нову платформу зі своїми поточними системами.
- Забезпечення коректності даних: Дуже важливо зберігати коректність та узгодженість у всіх документах та системах, оскільки інформація змінюється та розвивається. Функція автоматичного оновлення та нагадування допомагає гарантувати, що документація залишається актуальною і точною.

1.2 Дослідження сучасних методів зберігання та організації інформації

1.2.1 Вплив методів зберігання інформації на ефективність

Методи, що використовуються для зберігання, організації та доступу до інформації, стали вирішальними для ефективності роботи організацій через експоненціальне зростання цифрових даних у сучасному корпоративному середовищі. У цьому розділі подано ретельний аналіз найпопулярніших сучасних

методів організації та зберігання інформації, а також розглянуто тактику і технології, які лежать в основі цих систем. Ці стратегії допомагають компаніям і організаціям долати зростаючі перешкоди, пов'язані з фрагментацією даних, їхнім обсягом і необхідністю швидкого, безпечного та ефективного доступу до життєво важливої інформації.

Удосконалені методи організації та пошуку, безперешкодна співпраця та інтеграція даних у реальному часі між кількома платформами - все це риси сучасних систем зберігання, які виходять за рамки простого зберігання даних. У цьому розділі ми розглянемо хмарні сховища, системи управління документами (DMS), системи управління знаннями (KMS), системи управління контентом (CMS) та гібридні сховища. Найкращі методи для сучасного управління інформацією будуть розкриті в цьому дослідженні шляхом оцінки переваг, недоліків і сфер застосування кожного методу.

1.2.2 Недоліки традиційних методів зберігання інформації

Перш ніж заглиблюватися в сучасні системи, необхідно зрозуміти розвиток методів зберігання та організації даних. Хоча вони мають багато недоліків, традиційні методи, такі як локальне зберігання файлів і паперові системи, послужили основою для створення сучасних альтернатив.

Паперові документи слугували фізичним сховищем інформації протягом багатьох поколінь. Ці матеріальні документи зберігалися в архівах, картотеках і папках. Працівники звичайних підприємств повинні були вручну отримувати та обробляти папери, що створювало проблеми з безпекою, доступністю та ефективністю.

Обмеження паперових систем:

- Обмеження щодо зберігання: Паперові системи важко масштабувати, оскільки вони потребують багато фізичного простору для зберігання.
- Проблеми з можливістю пошуку: Пошук у фізичних файлах є трудомістким і займає багато часу. Існує значна ймовірність загубити або втратити життєво важливі документи.

- Перешкоди для співпраці: Важко співпрацювати з членами команди, які географічно розподілені та мають спільні документи.
- Ризики безпеки: Паперові документи схильні до втрати, крадіжки та впливу навколишнього середовища (наприклад, вогню та води).

Організації перейшли на локальне зберігання файлів з впровадженням цифрових обчислень, зазвичай зберігаючи файли на персональних комп'ютерах, внутрішніх серверах або жорстких дисках. Порівняно з паперовими системами, локальні файлові системи забезпечують швидший доступ до даних, але вони також принесли з собою нові труднощі.

Обмеження локального файлового сховища:

- Обмеження ємності: Зі збільшенням обсягу даних локальне сховище часто потребує більше обладнання через свою обмежену місткість.
- Вразливості безпеки: Локальні сховища файлів можуть бути вразливими до незаконного доступу та витоку даних за відсутності централізованого контролю безпеки.

1.2.3 Хмарні сховища

Впровадження хмарних обчислень повністю змінило те, як підприємства обробляють і зберігають дані. Дані, що зберігаються на віддалених серверах, якими керують провайдери хмарних послуг і до яких є доступ в режимі онлайн, називаються хмарними сховищами.

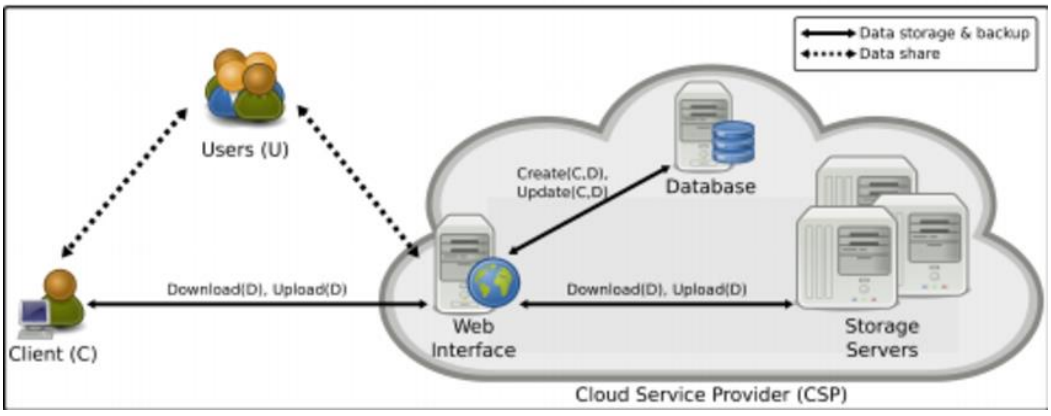

Рис. 1.1. Принцип роботи хмарного сховища

Основні переваги хмарних сховищ:

- Переваги масштабованості хмарних сховищ: Оскільки хмарне сховище пропонує майже безмежний простір для зберігання даних, компанії можуть просто масштабувати його відповідно до своїх потреб без необхідності додатково платити за фізичне сховище.
- Глобальна доступність: Якщо клієнти мають доступ до Інтернету, хмарне сховище гарантує доступ до своїх даних з будь-якої точки світу. Така глобальна доступність покращує безперервність бізнесу та полегшує віддалену роботу.
- Економічна ефективність: Оскільки хмарні сховища дозволяють організаціям орендувати обсяг пам'яті відповідно до їхніх потреб, це усуває потребу у великих початкових інвестиціях в апаратне забезпечення.
- Резервне копіювання та безпека даних: Поважні компанії, що надають послуги хмарних сховищ, використовують надійні заходи безпеки, включаючи багатофакторну автентифікацію, шифрування та автоматичне резервне копіювання даних. Це знижує ймовірність втрати даних і гарантує збереження важливих даних компанії.

Хмарні сховища мають певні недоліки, незважаючи на їхні численні переваги:

- Хмарне зберігання конфіденційних даних створює проблеми з конфіденційністю та безпекою, особливо коли дані зберігаються на серверах, розташованих за межами компанії.
- Для роботи хмарних сервісів необхідне підключення до Інтернету, і будь-яке відключення або проблема в мережі може перешкодити користувачам отримати доступ до даних.

1.2.4 Системи управління контентом (CMS)

Спеціалізовані платформи, які називаються системами управління контентом (CMS), призначені для роботи з цифровою інформацією, такою як документи, відео, фотографії та текст. Система управління контентом (CMS) - це комплексне рішення,

яке дозволяє компаніям ефективно створювати, впорядковувати, зберігати та розповсюджувати матеріали (рис. 1.2.). У таких галузях, як видавнича справа, електронна комерція та освіта, ці системи мають вирішальне значення.

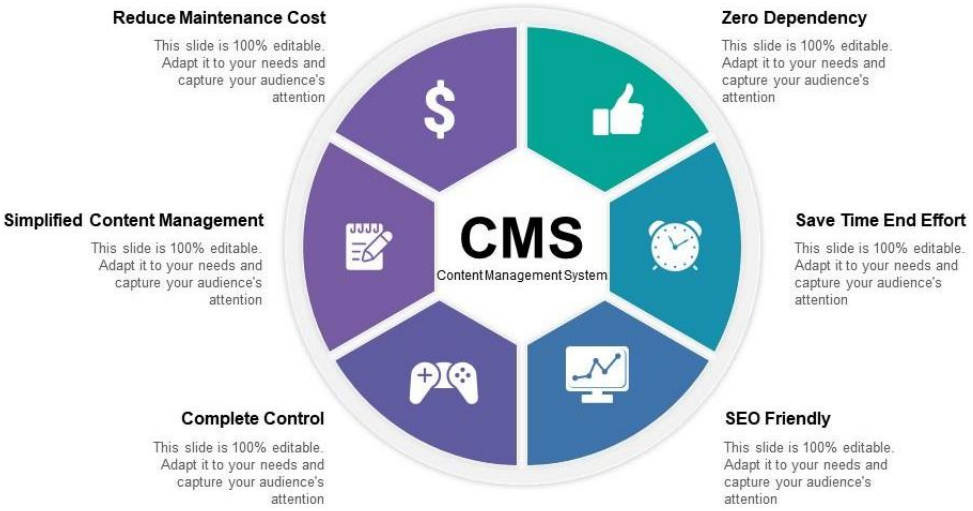

Рис. 1.2. Переваги CMS

Особливості централізованого репозиторію CMS: Платформи CMS покращують управління даними та їх пошук, пропонуючи єдине місце для всієї інформації, а саме:

- Управління робочим процесом та співпраця: Більшість платформ CMS постачаються з інструментами, які дозволяють групам співпрацювати над створенням і редагуванням контенту, що прискорює робочі процеси.
- Контроль версій: CMS-системи відстежують версії та зміни контенту, що дозволяє легко повертатися до попередніх ітерацій за потреби.
- Пошук та вилучення: Користувачі можуть швидко знайти певний матеріал завдяки розширеним функціям пошуку, таким як пошук за ключовими словами, тегами та метаданими.

Відомі системи управління контентом:

- WordPress: Завдяки величезному вибору плагінів і тем, WordPress є найпопулярнішою системою управління контентом, особливо для блогів і невеликих веб-сайтів.
- Joomla: Система управління веб-сайтами з розумною кривою навчання та широкими можливостями.
- Atlassian Confluence: Бізнес-орієнтована система управління контентом для внутрішньої документації та управління знаннями.

1.2.5 Системи управління знаннями, документами та гібридні системи

Мета систем управління знаннями (СУЗ) - зберігати, впорядковувати та поширювати знання всередині компанії. СУЗ дозволяє компаніям фіксувати як явні, так і неявні знання і робити їх доступними для колег по всій компанії (рис. 3.1).

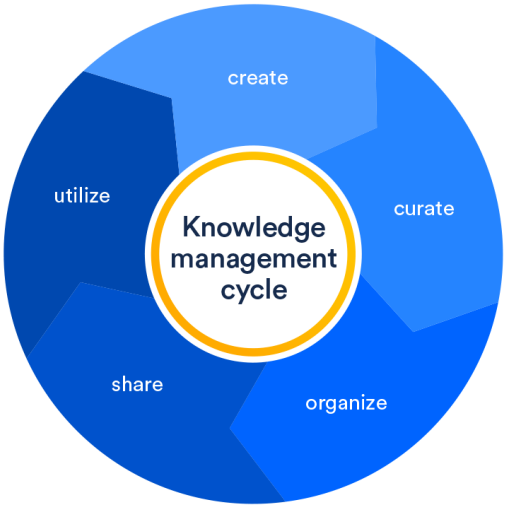

Рис. 1.3. Життєвий цикл знань в системі управління

Особливості сховища знань СУЗ:

- СУЗ пропонує централізоване місце для зберігання знань у різних форматах, таких як найкращі практики, документи та посібники.
- Інструменти для співпраці: Ці платформи полегшують обмін інформацією через внутрішні соціальні мережі, чати та дискусійні дошки.

- Пошук і вилучення: Щоб допомогти користувачам швидко знаходити найнеобхіднішу інформацію, системи часто включає в себе пошукові інструменти на основі штучного інтелекту.
- Збереження знань: Гарантуючи, що важливі організаційні знання зберігаються і стають доступними навіть тоді, коли співробітники звільняються з компанії,СУЗ сприяє збереженню знань.

1.3 Визначення вимог до системи для зберігання та організації інформації

Вибір правильної технології - це лише один з аспектів проектування системи зберігання та організації інформації. Він вимагає глибокого розуміння конкретних бізнес-вимог, типів даних, що зберігаються, а також способів обміну, доступу та захисту цих даних. Оскільки визначення вимог створює основу для архітектури, функціональності та загального дизайну системи, воно є важливим етапом у розробці системи управління інформацією.

Потреби в такій системі стають дедалі складнішими в контексті сучасних підприємств; вони включають не лише ємність сховища, але й інтеграцію різних стратегій управління інформацією, сприяння командній роботі та гарантію безпеки і конфіденційності даних. З акцентом на ключові елементи, які необхідно враховувати при створенні системи зберігання та організації інформації, цей розділ пропонує детальний посібник з визначення цих вимог.

1.3.1 Функціональні вимоги

Точні дії, функції та види діяльності, які повинна підтримувати система, викладені у функціональних вимогах. Ці специфікації слугують основою для архітектури системи та визначають її функціональність щодо пошуку даних, управління документами, взаємодії з користувачами та інших сфер. Ключові функціональні критерії, які необхідно взяти до уваги, перераховані нижче:

Зберігання та пошук даних є ключовим компонентом будь-якої системи управління інформацією. Великі обсяги різних типів даних (документи, відео, фотографії тощо) повинні ефективно зберігатися системою, щоб забезпечити швидкий і надійний пошук у відповідь на визначені користувачем запити.

Компанії, які обробляють велику кількість документів, повинні мати ефективне управління документами. Такі функції, як тегування метаданих, протоколи затвердження документів і контроль версій, повинні бути включені в систему.
Щоб полегшити класифікацію та пошук документів, система повинна дозволяти додавати до них певну інформацію (наприклад, автора, дату створення та відділ).

Однією з найважливіших вимог для сучасних підприємств є співпраця. Дозволяючи користувачам працювати над одним файлом, обмінюватися інформацією та залишати відгуки, система повинна сприяти командній роботі.

Команди повинні мати можливість легше співпрацювати незалежно від того, в якій точці світу вони знаходяться, завдяки підтримці системою редагування та внесення змін до спільних документів у режимі реального часу.

Можливість позначати тексти, виділяти важливі фрагменти та писати коментарі в межах системи.

У системі має бути передбачено детальний контроль доступу, який визначає, хто може читати, редагувати або видаляти документи залежно від ролей користувачів.

Будь-яка система повинна надавати пріоритет безпеці даних. В систему повинні бути вбудовані надійні механізми безпеки для захисту від втрати, крадіжки та несанкціонованого доступу до конфіденційних даних. Важливі функції безпеки включають в себе наступні:

- Шифрування: Щоб уникнути небажаного доступу або перехоплення, дані повинні бути зашифровані під час передачі та у стані спокою.
- Аутентифікація: Надійні методи автентифікації користувачів, включаючи двофакторну автентифікацію, гарантують, що доступ до конфіденційних даних може отримати лише уповноважений персонал.

- Контроль доступу: Залежно від посади в компанії, можливість користувачів переглядати або змінювати певні дані обмежується за допомогою контролю доступу на основі ролей (RBAC).
- Журнали аудиту: Для проведення аудиту відповідності та безпеки система повинна відстежувати всі дії користувачів, такі як доступ до файлів, їх модифікації та видалення.

Щоб запобігти втраті даних у випадку системних збоїв, стихійних лих або ненавмисного видалення, регулярне резервне копіювання даних та ефективні процедури відновлення мають вирішальне значення.

Автоматизоване резервне копіювання потрібне для регулярного збереження копій документів і баз даних система повинна мати можливість автоматизованого резервного копіювання.

Щоб зменшити час простою, система повинна мати чіткий план аварійного відновлення, який дозволяє швидко відновити дані.

Щоб полегшити безперешкодний обмін даними та інтероперабельність, система повинна пропонувати API для простої взаємодії з іншими бізнес-інструментами та системами.

Можливість інтеграції зі сторонніми програмами, включаючи бухгалтерське програмне забезпечення, платформи електронної пошти та інструменти для спільної роботи.

1.3.2 Нефункціональні вимоги

Експлуатаційні характеристики системи, такі як масштабованість, надійність, продуктивність і зручність використання, описуються нефункціональними вимогами. Ці специфікації гарантують, що система задовольняє очікування користувачів і працює з максимальною ефективністю в різних сценаріях.

Підприємства потребують більшого обсягу управління інформацією в міру свого розширення. Щоб система могла управляти зростаючими обсягами даних і активністю користувачів, вона повинна бути масштабованою.

Вертикальна масштабованість - це можливість збільшення потужності системи шляхом надання кожному серверу додаткових ресурсів (таких як процесор і пам'ять). Можливість розширити систему, додавши більше серверів або вузлів, щоб розподілити робоче навантаження і забезпечити високу доступність, відома як горизонтальна масштабованість.

Навіть для великих наборів даних необхідна високопродуктивна система, щоб гарантувати користувачам швидкий доступ до даних та їхнє швидке отримання.
У ситуаціях з високим навантаженням, швидкий час реакції для доступу до документів і пошуку даних.

Оптимізована база даних використовує стратегії кешування, індексування та оптимізації запитів, щоб гарантувати швидкий та ефективний пошук даних.
Нетехнічні користувачі повинні знайти систему простою у використанні та інтуїтивно зрозумілою. Чітка навігація, інтуїтивно зрозумілі робочі процеси та зручні інтерфейси мають важливе значення для забезпечення ефективної взаємодії користувачів з системою.

Інтерфейс користувача має бути зрозумілим, простим у використанні та логічно організованим за допомогою ярликів.

Для гарантії працездатності системи, коли це необхідно, з мінімальним часом простою, надійність та доступність є дуже важливими. Гарантія безперебійної роботи за умови регулярного технічного обслуговування та оновлень, система повинна мати час безвідмовної роботи не менше 99,9%. Система повинна бути здатна протистояти збоям (наприклад, відмовам обладнання), які не призводять до значних перерв у наданні послуг.

1.4 Висновоки до розділу

У цьому розділі висвітлено основи створення системи зберігання та організації інформації. Це дослідження заклало основу для розуміння стану систем управління інформацією на сьогоднішній день і конкретних потреб, необхідних для створення працездатного рішення. Визначено важливі елементи, на які слід орієнтуватися при

розробці та функціонуванні системи, розглянувши різноманітні ідеї, сучасні технології та вимоги користувачів.

Крім того подано короткий огляд ідей, що стосуються структури та зберігання інформації. У цьому розділі було розглянуто історичний розвиток систем управління інформацією, починаючи з ручних, паперових методів і закінчуючи впровадженням цифрових систем. Розглянуто вплив розвитоку цих систем на потреби в централізованому зберіганні даних. Ці ідеї були важливими для визначення цілей і завдань для розробки системи.

Проаналізовано сучасні підходи до організації та зберігання інформації, дослідивши такі технології, як розподілені бази даних, хмарні сховища та інструменти для співпраці в режимі реального часу. Цей розділ продемонстрував, як ці розробки, що забезпечують масштабованість, гнучкість та покращену доступність, повністю трансформували управління інформацією. Моє дослідження показало, як система використовує ці передові технології, щоб підтримувати свій статус високомасштабованої та кооперативної платформи. Це мало вирішальне значення для усвідомлення важливості хмарних технологій, синхронізації в режимі реального часу та простоти управління даними і контентом.

Також перераховано точні специфікації для розробки системи, включаючи функціональні та нефункціональні вимоги, які необхідно було виконати. Аналіз зосереджено на тому, щоб переконатися, що система запропонує рішення, яке задовольнить потреби користувачів, дотримуючись при цьому суворих вимог до продуктивності, безпеки та доступності.

РОЗДІЛ 2
ОСНОВНІ МОДЕЛІ ТА АЛГОРИТМИ ДЛЯ РОЗРОБКИ СИСТЕМ ДЛЯ ЗБЕРІГАННЯ ТА ОРГАНІЗАЦІЇ ІНФОРМАЦІЇ

2.1 Огляд фундаментальних теоретичних моделей управління та організації інформаці

2.1.1 Огляд основних теоретичних моделей управління інформацією

Численні теоретичні моделі слугують основою для створення систем зберігання та організації інформації, впливаючи як на їхню архітектуру, так і на функціональність. Ці моделі дозволяють розробляти ефективні, масштабовані та орієнтовані на користувача системи, пропонуючи організовану структуру для збору, зберігання, пошуку та використання інформації. У цьому розділі ми розглянемо найважливіші теоретичні моделі управління інформацією.

Реляційна модель, яку запропонував Едгар Ф. Кодд у 1970 році, повністю змінила спосіб зберігання та пошуку даних. Рядки (кортежі) і стовпці (атрибути) складають відношення (таблиці), в які ця модель впорядковує дані. Кожна таблиця має окремий ключ, який гарантує цілісність даних і полегшує зв'язок між різними наборами даних.

Фундаментальні ідеї:

- Незалежність даних забезпечує гнучкість, відокремлюючи фізичну реалізацію даних від їхньої логічної структури.
- Впорядковуючи дані в стандартизовані формати, нормалізація зменшує надлишковість і будь-які розбіжності.
- Мова запитів: Дані можна визначати, змінювати та отримувати за допомогою мови структурованих запитів (SQL).
- Актуальність для управління інформацією: Структуроване управління та зберігання даних є поширеним застосуванням реляційної моделі. Оскільки вона гарантує цілісність та узгодженість даних, її можуть використовувати додатки корпоративного рівня. Для збереження

організованих даних і зв'язків між різними сутностями, включаючи користувачів, документи і команди, система використовує PostgreSQL, реляційну базу даних, відповідно до реляційної парадигми.

Високорівневою концептуальною моделлю для візуального визначення даних та їхніх зв'язків є E-R модель. Цей підхід, вперше представлений Пітером Ченом у 1976 році, використовує діаграми для зображення сутностей, властивостей та їх взаємодій.

Основні компоненти:

- Сутності: Заміщають окремі елементи або ідеї (наприклад, Користувач, Документ).
- Атрибути: Описують характеристики сутності (наприклад, назва документа, ім'я користувача).
- Зв'язки: Показує, як дві сутності пов'язані між собою (наприклад, Користувач завантажує Документ).
- Відповідність управлінню інформацією: Модель E-R відіграє вирішальну роль у проектуванні систем баз даних, пояснюючи належну структуру даних.

Модель документів зберігає дані в напівструктурованому вигляді, зазвичай у вигляді JSON або XML-документів, на відміну від організованої природи реляційної моделі. Кожен документ містить всю необхідну інформацію, що робить його самодостатнім і простим для доступу.

Основні характеристики:

- Гнучкість схеми дозволяє зберігати напівструктуровані та неструктуровані дані.
- Для складних даних ієрархічне представлення дозволяє створювати багаторівневі структури.
- Ефективність: Ідеально підходить для додатків, де отримання даних відбувається частіше, ніж їх оновлення.
- Актуальність для управління інформацією: Системи, що потребують гнучкості та масштабованості, часто використовують модель документів.

Незважаючи на те, що системп в основному покладається на реляційні бази даних, він інтегрує модель документів у певні функції, включаючи напівструктуровані метадані або зберігання налаштувань конфігурації.

2.2 Алгоритми організації та пошуку даних

Будь-яка система управління інформацією повинна мати ефективну організацію та пошук даних. На зручність, масштабованість і швидкість роботи системи безпосередньо впливають алгоритми, що використовуються. Алгоритми, які підтримують організацію та пошук даних, ретельно розглядаються в цьому розділі, з акцентом на їх застосування, основні концепції та значення для системи.

2.2.1 Алгоритми сортування для організації даних

Одним із важливих процесів в організації даних є сортування. Добре відсортовані дані не лише підвищують ефективність пошуку, але й полегшують їхнє використання користувачами.

Метод «розділяй і володарюй», який називається швидким сортуванням, розділяє набір даних на менші підмножини, сортує їх, а потім об'єднує у відсортовану структуру. Для великих наборів даних середня часова складність складає O(nlogn), що свідчить про її надзвичайну ефективність.

Сортування злиттям це стратегія , яка відома своєю надійністю та стабільністю, - це сортування злиттям. Вона ідеально підходить для сортування пов'язаних даних або наборів даних, які повинні підтримувати порядок, а її часова складність становить O(nlogn).

При роботі з ієрархічними структурами даних або збереженні порядку в історії змін документів користувачі використовують сортування злиттям.

Radix Sort: Цей метод сортує дані шляхом послідовного сортування символів або чисел. Найкраще підходить для сортування алфавітно-цифрового вмісту, наприклад, міток часу або ідентифікаторів документів.

Використовується в системі: Ключі індексу Elasticsearch сортуються за допомогою Radix Sort, щоб гарантувати оптимальні результати запиту.

Основою інформаційно-пошукових систем є алгоритми пошуку, які гарантують, що користувачі зможуть швидко і точно знайти потрібну їм інформацію. Бінарний пошук: Бінарний пошук має часову складність O(logn) і працює з відсортованими наборами даних. Поки цільовий елемент не буде знайдено, він постійно ділить простір пошуку навпіл.

Застосування в інфостеках: Бінарний пошук використовується для виявлення певних записів в індексованих наборах даних, таких як документ за його унікальною ідентифікацією.

Методи обходу графів Depth-First Search (DFS) та Breadth-First Search (BFS) мають вирішальне значення для дослідження зв'язків між даними. DFS занурюється в гілку, перш ніж розвернутися, тоді як BFS досліджує кожного сусіда, перш ніж заглибитися.

Застосування в системі: Можливості відображення зв'язків, такі як пошук пов'язаних документів і відображення залежностей між даними, використовують BFS і DFS.

Важливий алгоритм для пошукових систем називається «перевернуте індексування». Він дозволяє здійснювати швидкий повнотекстовий пошук шляхом зіставлення термінів з текстами, в яких вони з'являються.

Застосування в системі: Elasticsearch, який використовує інвертоване індексування для швидкого та ефективного пошуку за ключовими словами.

Алгоритми хешування

Щоб уможливити постійний пошук даних у хеш-таблицях, хешування перетворює дані у значення фіксованого розміру (хеш-коди).

Функції хешування: Для створення чітких ідентифікаторів для записів даних використовуються функції хешування, такі як MD5, SHA-256 [5].

Застосування в системі: Хешування використовується для захисту та індексування даних, включаючи створення різних ідентифікаторів для оновлень документів або сеансів користувачів.

Колізії хешування вирішуються за допомогою таких стратегій, як ланцюжок і відкрита адресація, які гарантують точний і швидкий пошук даних.

Використання в системі: Цілісність хеш-індексів у базі даних залежить від вирішення колізій.

Алгоритми кластеризації допомагають впорядкувати дані та виявити інсайти, групуючи пов'язані елементи даних.

Кластеризація K-середніх: K-Means використовує схожість ознак для поділу даних на k кластерів. Розпізнавання образів і неконтрольоване навчання широко використовують цей метод.

Застосування в системі: K-Means використовується для кластеризації осіб або документів на основі спільних характеристик, що дозволяє надавати рекомендації за категоріями або цільові рекомендації.

Ієрархічна кластеризація: Ця техніка створює деревоподібну структуру кластерів даних шляхом поділу більших кластерів (дивізіональна) або об'єднання менших кластерів (агломеративна).

Застосування в системі: Щоб допомогти користувачам природно досліджувати дані, ієрархічна кластеризація використовується для створення вкладених категорій для класифікації документів.

Такі системи, як соціальні мережі, рекомендаційні системи та графи знань, які контролюють взаємозв'язки між даними, залежать від алгоритмів графів.

Найкоротший шлях між вузлами у зваженій мережі визначається за допомогою алгоритму Дейкстри [5].

Використання в системі: використовує алгоритм Дейкстри для оптимізації навігації по зв'язках між документами, гарантуючи, що користувачі зможуть своєчасно знаходити найбільш релевантний контент.

PageRank: Створений компанією Google, PageRank присвоює рейтинг вузлам мережі відповідно до їх зв'язків та значущості.

Застосування в системі: Щоб покращити користувацький досвід, використовує стратегії, натхненні PageRank, для ранжування документів відповідно до релевантності під час пошуку [8].

Пошук інформації за допомогою алгоритмів машинного навчання

Машинне навчання все частіше використовується в сучасних інформаційних системах для покращення пошуку та впорядкування даних.

Рекомендаційні системи: Щоб запропонувати релевантну інформацію, такі алгоритми, як фільтрація на основі контенту та спільна фільтрація, вивчають вподобання та поведінку користувачів.

Застосування системі: Системи рекомендацій підвищують продуктивність і залученість користувачів, пропонуючи релевантні документи.

Обробка природної мови (NLP): Методи NLP, які витягують значення з текстових даних, включають в себе вставки слів і TF-IDF (частота терміна - зворотна частота документа) [30].

Використання в системі: Розуміючи запити користувачів і співвідносячи їх з відповідними документами, обробка природної мови (NLP) покращує можливості пошуку.

Алгоритми організації та пошуку даних мають важливе значення для роботи систем зберігання та організації інформації. Ці алгоритми гарантують, що дані є доступними, значущими та придатними для будь-яких дій - від сортування та пошуку до групування та машинного навчання. Система забезпечує потужну та ефективну платформу, використовуючи поєднання різних алгоритмів. Використовуючи передові методи, такі як системи рекомендацій, обхід графів та інвертоване індексування, покращує організаційну ефективність та процес прийняття рішень, а також полегшує пошук даних.

2.3 Методи забезпечення цілісності та узгодженості даних

При розробці будь-якої системи, що зберігає та впорядковує інформацію, важливими компонентами є узгодженість та цілісність даних. Ці настанови гарантують, що дані в системі є точними, надійними та придатними для використання протягом усього часу її існування. Неточності, непорозуміння та неефективне прийняття рішень можуть виникнути через нездатність підтримувати цілісність

даних. Основні стратегії і тактики збереження узгодженості і цілісності даних, розглядаються в цьому розділі.

2.3.1 Визначення та значення цілісності даних

Точність, повнота та достовірність даних називається їхньою цілісністю. Якщо дані не змінені спеціально уповноваженими суб'єктами, вона гарантує, що інформація залишається незмінною під час зберігання, передачі та пошуку. Цілісність даних має вирішальне значення в таких системах, які діють як централізовані сховища організаційної інформації.

Важливими категоріями цілісності даних є:

- Цілісність сутностей, яка часто підтримується за допомогою первинних ключів, гарантує, що кожен запис у базі даних є окремим.
- Підтримка легітимних зв'язків між сутностями даних, наприклад, переконавшись, що зовнішні ключі посилаються на легітимні головні ключі, відома як посилальна цілісність.
- Цілісність домену: Забезпечує відповідність даних заздалегідь визначеним правилам (наприклад, числовим діапазонам або певним форматам) шляхом застосування обмежень на типи та значення даних.

Сучасні системи використовують ряд тактик і методів для збереження цілісності даних. Щоб гарантувати цілісність даних, реляційні бази даних, такі як PostgreSQL, яка використовується в багатьох системах, включають вбудовані обмеження: первинні ключі, зовнішні ключі, обмеження [6].

Також застосовуються наступні методи забезпечення цілісності дантх:

- Правила перевірки на прикладному рівні: Застосовуючи правила валідації, прикладний рівень забезпечує додатковий рівень захисту. TypeScript використовується, щоб гарантувати безпеку типів, а перевірки валідації на внутрішньому рівні забезпечують дотримання певних логічних правил, форматів і структур даних.
- Контрольні суми та хешування: цілісність даних перевіряється під час зберігання або передачі за допомогою методів хешування, таких як SHA-

256. Порівнюючи хеш-значення, система може виявити несанкціоновані зміни.

- Журнали аудиту: Ці журнали документують усі зміни даних, включаючи користувача, час і виконані дії. Таким чином гарантується підзвітність та відстежуваність.
- Зберігання версій: Система зберігає ретельну історію версій кожного документа, що дозволяє користувачам відстежувати зміни та відкочувати попередні ітерації.
- Шифрування даних: Під час зберігання або передачі даних шифрування гарантує їхню безпеку та недоторканність. Такі алгоритми, як AES-256, захищають дані від небажаних змін.

2.3.2 Визначення та значення узгодженості даних

Узгодженість даних гарантує, що інформація є послідовною і логічною у всіх екземплярах системи. Узгодженість особливо важко підтримувати в розподілених системах, оскільки багато вузлів виконують дії одночасно.

Підтримка узгодженості гарантує:

- Оновлення даних відображаються в кожному екземплярі.
- Неправильні або суперечливі стани даних не створюються одночасним доступом.
- Кожна програма та користувач, які взаємодіють з системою, бачать ті самі дані.

Виділяють наступні методи забезпечення узгодженості даних:

- Реляційні бази даних, такі як PostgreSQL, дотримуються властивостей ACID (атомарність, узгодженість, ізоляція, довговічність), щоб зберегти узгодженість під час транзакцій.
- Атомарність: Гарантує, що транзакція буде або належним чином завершена, або відкотиться повністю.
- Послідовність: Гарантує, що транзакції переміщують базу даних між легітимними станами.

- Ізоляція: Зупиняє вплив паралельних процесів один на одного.
- Довговічність: Гарантує, що навіть у випадку збою, зафіксовані транзакції будуть збережені.
- Двофазний протокол фіксації: Цей протокол гарантує, що кожен вузол розподіленої системи узгоджує транзакцію перед її фіксацією.

Для вирішення конфліктів в системах використовують наступні стратегії для вирішення конфліктів, спричинених одночасним оновленням даних:

- Останній запис перемагає (LWW): Цей метод зберігає останнє оновлення.
- Операції злиття: Відповідно до встановлених правил, конфліктуючі оновлення об'єднуються.
- Синхронізація та кешування: хоча кешування підвищує ефективність, воно може призвести до застарівання даних. Послідовність кешованих даних гарантується методами синхронізації, такими як анулювання кешу.

2.4 Моделі безпеки інформаційних систем

Безпека є важливим компонентом інформаційних систем, які призначені для зберігання та впорядкування конфіденційних або важливих даних. Надійна модель безпеки забезпечує безпечну участь користувачів, гарантуючи при цьому, що дані захищені від небажаного доступу, зміни або втрати. У цьому розділі розглядаються основні моделі безпеки, їхні принципи та використання в сучасних інформаційних системах.

2.4.1 Огляд моделей безпеки

Модель безпеки - це фреймворк або структура, яка описує, як політики безпеки реалізуються в інформаційних системах, щоб гарантувати доступність, конфіденційність і цілісність. Ці моделі мають вирішальне значення для збереження довіри користувачів, захисту комунікації та захисту конфіденційних даних.

Розробка та впровадження системних засобів безпеки, таких як контроль доступу, шифрування та автентифікація, ґрунтується на моделях безпеки. Моделі безпеки

інформаційних систем ґрунтуються на низці фундаментальних ідей. Завдяки конфіденційності доступ до конфіденційних даних мають лише уповноважені особи. Цілісність захищає від незаконних змін або пошкодження даних. Доступність гарантує, що дані будуть доступні авторизованим користувачам, коли це необхідно. Автентифікація підтверджує особу користувача, коли він отримує доступ до системи.

Дозволи надаються користувачам відповідно до їхніх ролей або обов'язків через авторизацію. Підзвітність відстежує та реєструє кожну дію користувача, щоб гарантувати відстежуваність та запобігти зловмисній поведінці.

Інформаційні системи часто використовують наступні моделі безпеки:

- Контроль дискреційного доступу (DAC) - власник даних вирішує, хто може їх переглядати або змінювати в рамках гнучкого підходу до управління доступом, відомого як DAC. Для визначення прав доступу використовуються списки контролю доступу (ACL) та ідентифікатори користувачів. Наприклад, власники документів мають можливість надавати іншим особам або командам права на читання, запис або коментування.
- Вимоги до контролю доступу (MAC) - суворий контроль доступу, встановлений центральним органом влади, а не власником даних, забезпечується за допомогою MAC - Користувачі можуть отримати доступ до даних лише в межах свого рівня допуску, який визначається рівнем класифікації, наданим кожному користувачеві та об'єкту даних (наприклад, конфіденційний, цілком таємний).
- RBAC, або контроль доступу на основі ролей - RBAC призначає дозволи відповідно до ролей користувачів всередині компанії. Користувачі успадковують дозволи від своїх ролей, які визначаються на основі посадових обов'язків. Адміністратори можуть призначити такі ролі, як «Редактор», «Переглядач» або «Менеджер», із заздалегідь визначеними рівнями доступу до документів і проектів.
- Контроль доступу на основі атрибутів (ABAC) - дозволи надаються за допомогою ABAC відповідно до характеристик, таких як ролі

користувачів, типи пристроїв, місцезнаходження або час. Умови атрибутів використовуються для динамічної оцінки політик.

2.4.2 Методи захисту інформаційних систем

Для захисту даних користувачів і функціональності системи сучасні інформаційні системи використовують різноманітні методи захисту:

- Процес шифрування: конфіденційна інформація захищена як під час передачі (в дорозі), так і під час зберігання (в стані спокою) завдяки шифруванню даних.
- MFA, або багатофакторна автентифікація: завдяки використанню декількох факторів, включаючи пароль і одноразовий код, що надсилається на телефон, MFA вимагає від користувачів автентифікації.
- Журнали моніторингу та аудиту: усі дії користувачів у системі, такі як доступ до файлів, редагування та зміна дозволів, відстежуються за допомогою журналів аудиту.
- Безпечні API: щоб зупинити небажаний доступ і зловживання API, внутрішні API системи захищені HTTPS, автентифікацією на основі токенів (наприклад, JWT) і методами обмеження швидкості.
- Відновлення та резервне копіювання даних: часте резервне копіювання гарантує, що дані можуть бути відновлені в разі кібератаки, системних збоїв або ненавмисного видалення.

Організації стикаються з низкою труднощів при впровадженні безпеки інформаційних систем, навіть при наявності складних моделей безпеки:

- Безпека і зручність використання: Суворий контроль доступу або часті вимоги МЗС є прикладами заходів безпеки, які можуть зробити систему менш зручною для користувачів. Впроваджуючи зручні для користувача заходи безпеки, такі системи прагнуть досягти рівноваги.
- Еволюція кіберзагроз: Постійно з'являються нові вразливості та методології атак, що вимагає від систем адаптації та випередження можливих загроз.

- Внутрішні загрози: Працівники або надійні користувачі можуть навмисно або ненавмисно поставити під загрозу безпеку даних. Такі небезпеки можна зменшити за допомогою моніторингу та контролю доступу.
- Вимоги відповідності: Проектування та експлуатація систем ускладнюються, коли дотримуються таких нормативних актів, як GDPR або HIPAA.

Щоб захистити дані користувачів і гарантувати цілісність системи, потрібно використовувати багаторівневий підхід до безпеки, який поєднує в собі найсучасніші моделі та методи:

- Контроль доступу на основі ролей та атрибутів: система пропонує тонкий контроль доступу до документів за допомогою гібридної методології.
- Повне шифрування: Для забезпечення конфіденційності всі дані, що надходять до системи, шифруються під час передачі та зберігання.
- Версії та аудиторські сліди: Для забезпечення відповідності та підзвітності веде ретельну історію версій та журнали активності.
- Відповідність стандартам безпеки: Щоб відповідати законодавчим нормам, платформа відповідає галузевим стандартам, включаючи GDPR та ISO 27001.

Розробка та впровадження надійних систем зберігання та організації інформації вимагає використання моделей безпеки. Доступність, конфіденційність та цілісність даних гарантується системою завдяки використанню методів та моделей шифрування, таких як RBAC та ABAC. Всеохоплююча структура безпеки зберігає зручність використання для кінцевих користувачів, одночасно вирішуючи сучасні проблеми, такі як внутрішні загрози та вимоги дотримання нормативних вимог.

2.5 Інтеграція штучного інтелекту в управління інформацією

Обробка, зберігання та використання даних в організаціях докорінно змінюється завдяки впровадженню штучного інтелекту (ШІ) в системи управління інформацією. Завдяки технологіям штучного інтелекту (ШІ), включаючи машинне

навчання, обробку природної мови (NLP) та інтелектуальну автоматизацію, великі обсяги даних організовуються, отримуються та аналізуються більш ефективно. Пошук зі штучним інтелектом враховує такі змінні, як наміри користувача, контекст і попередні взаємодії, щоб за допомогою складних алгоритмів видавати точніші та релевантніші результати пошуку. На відміну від звичайних методів пошуку, пошук зі штучним інтелектом постійно вдосконалюється, враховуючи відгуки та поведінку користувачів.

ШІ може зменшити потребу у взаємодії з людиною, автоматично класифікуючи і тегуючи контент відповідно до його атрибутів. Це особливо корисно для систем з великими обсягами даних.

Щоб допомогти користувачам швидше отримати доступ до потрібної інформації, наприклад, використовувати предиктивну аналітику, щоб рекомендувати документи, які, ймовірно, знадобляться користувачам на основі їхньої попередньої поведінки.

ШІ полегшує організацію величезних обсягів неструктурованих даних, дозволяючи швидше їх знаходити та ефективніше управляти ними. ШІ може запропонувати глибше розуміння організації інформації та рекомендувати ефективніші методи її структурування та класифікації, використовуючи алгоритми, які навчаються на даних.

Наприклад, визначаючи тенденції в доступі до даних, алгоритми штучного інтелекту можуть постійно вдосконалювати структуру зберігання документів, допомагаючи оптимізувати організацію інформації.

ШІ дозволяє компаніям видобувати корисну інформацію з величезних обсягів даних, сприяючи кращому та ефективнішому прийняттю рішень. Він має можливість вивчати минулі дані, виявляти закономірності та пропонувати пропозиції у світлі цих висновків.

Вивчаючи поведінку та вподобання користувачів, ШІ може допомогти у створенні персоналізованого досвіду. ШІ може пропонувати документи або інформацію залежно від потреб та інтересів користувача, розуміючи, як він взаємодіє з системою.

Конфіденційні дані можна легше захистити завдяки здатності ШІ виявляти порушення в режимі реального часу та передбачати можливі ризики для безпеки. ШІ здатний виявляти шкідливі дії або несанкціонований доступ, вивчаючи системні журнали та поведінку користувачів.

2.6 Оцінка існуючих моделей та алгоритмів в сучасних системах

2.6.1 Класифікація сучасних алгоритмів та моделей

Зберігання, організація, доступ до даних та маніпуляції з ними визначаються низкою моделей і методів, що використовуються сучасними інформаційними системами. Структури даних, адміністрування баз даних, машинне навчання та штучний інтелект - це лише деякі з добре відомих теорій комп'ютерних наук, які часто слугують основою для цих моделей та алгоритмів. В контексті управління інформацією ці моделі зазвичай поділяються на наступні групи.

Виділяють наступні моделі для баз даних:

- Реляційна парадигма: Ця популярна парадигма для структурованого зберігання і пошуку даних базується на таблицях і зв'язках між даними. Це надійний і надійний варіант для багатьох систем корпоративного рівня, оскільки він оперує даними за допомогою мови структурованих запитів (SQL) [6].
- В основі об'єктно-орієнтованої моделі лежить ідея, що об'єкти можуть містити як поведінку, так і дані. Найкраще підходить для систем, які потребують складних структур даних та об'єктних зв'язків.
- Модель NoSQL: Як альтернатива реляційним базам даних, нереляційні бази даних, або NoSQL, набули популярності, оскільки вони забезпечують більшу масштабованість і гнучкість, особливо при управлінні неструктурованими і напівструктурованими даними. До таких систем належать бази даних графів (наприклад, Neo4j), документно-орієнтовані бази даних (наприклад, MongoDB) та сховища ключ-значення (наприклад, Redis) [7].

Базовий метод пошуку, який шукає збіги у всьому наборі даних, називається лінійним пошуком. Незважаючи на свою простоту, його часова складність O(n) робить його неефективним для великих наборів даних.

За часової складності O(log n) бінарний пошук є більш ефективною технікою пошуку для відсортованих наборів даних. Бінарний пошук рекурсивно зменшує діапазон пошуку, розділяючи набір даних навпіл.

Хешування: Використання хеш-функції для зіставлення даних з ключами фіксованого розміру, хешування - це підхід до баз даних, який прискорює пошукові операції. Це дуже корисно для підвищення ефективності пошуку та індексування великих наборів даних.

Для організації та пошуку даних важливе значення має сортування. Сучасні системи часто використовують такі алгоритми, як швидке сортування, сортування злиттям і сортування групою для швидкого та ефективного сортування великих наборів даних.

Деревоподібні та графові алгоритми - структури даних, такі як графи, бінарні дерева та B-дерева, мають вирішальне значення для реляційної та ієрархічної організації даних. Деревоподібні та графові структури даних можна переглядати та змінювати за допомогою таких алгоритмів, як алгоритм Дейкстри, пошук в ширину (BFS) та пошук в глибину (DFS).

Методи кластеризації використовуються для об'єднання пов'язаних фрагментів даних у машинному навчанні. Часто використовувані методи кластеризації для класифікації та організації неструктурованих даних включають K-Means, DBSCAN та ієрархічну кластеризацію.

Створюючи окремий код для кожного набору даних або документа, контрольні суми та хеш-функції захищають цілісність даних і можуть бути використані для підтвердження їхньої легітимності та виявлення помилок, допущених під час зберігання або передачі.

Виявлення та виправлення помилок: Системи передачі та зберігання даних використовують такі алгоритми, як корекція помилок Ріда-Соломона та код Хеммінга

для виявлення та виправлення потенційних помилок, які можуть виникнути під час передачі даних або при зчитуванні зі сховища.

Навчання під наглядом: Використовуючи марковані навчальні дані, контрольоване навчання застосовує такі алгоритми, як нейронні мережі, дерева рішень і машини опорних векторів (SVM) для створення класифікацій або прогнозів. Навчання без нагляду: Для пошуку прихованих патернів або структур у немаркованих даних використовуються алгоритми кластеризації та виявлення аномалій, такі як K-середні та ізоляційні ліси.

Заохочуючи системи за досягнення бажаних результатів і караючи їх за прийняття неідеальних рішень, навчання з підкріпленням допомагає оптимізувати процеси прийняття рішень. Завдання, пов'язані з організацією та динамічним пошуком даних, можуть отримати вигоду від навчання з підкріпленням.

2.6.2 Оцінка сучасних алгоритмів і моделей

Ефективність, масштабованість, простота реалізації та адаптивність до нових випадків використання - ось деякі з аспектів, які можна використовувати для оцінки ефективності моделей та алгоритмів, описаних вище. Тепер давайте оцінимо ці моделі у світлі сучасних інформаційних систем.

Ефективність - це швидкість, з якою алгоритм може обробляти інформацію і видавати бажаний результат. Оскільки вони забезпечують швидший пошук даних, ніж лінійні методи пошуку, такі алгоритми, як бінарний пошук і хешування, вважаються ефективними в системах управління інформацією. Аналогічно, при роботі з великими наборами даних алгоритми сортування, такі як швидке сортування і сортування злиттям, працюють краще, ніж більш прості методи, такі як бульбашкове сортування.

Ефективність, однак, може відрізнятися залежно від конкретних потреб системи. Реляційні бази даних, наприклад, добре працюють зі структурованими даними, тоді як бази даних NoSQL, такі як MongoDB, ефективніші з неструктурованими даними, особливо при горизонтальному масштабуванні в розподілених системах [7].

Масштабованість - це здатність системи обслуговувати зростаючу кількість користувачів або даних без помітного падіння продуктивності. Завдяки своїй вродженій здатності масштабуватися у віддалених системах, бази даних NoSQL та алгоритми машинного навчання демонструють винятково хороші результати в цій сфері.

Наприклад, при обробці значних обсягів неструктурованих або напівструктурованих даних система може отримати вигоду від масштабованості баз даних NoSQL. Система повинна мати можливість збільшуватись у розмірі без втрати функціональності в міру того, як бізнес збирає додаткові дані. Оскільки алгоритми машинного навчання можуть безперервно навчатися на нових даних, не вимагаючи повного перенавчання, вони також ефективно масштабуються зі збільшенням обсягу даних [7].

При виборі відповідного алгоритму або моделі для управління інформацією важливим фактором є простота реалізації. Реляційні бази даних, особливо у звичайних додатках з чітко визначеними схемами, прості у налаштуванні та підтримці. Однак вони можуть виявитися складними в управлінні значними обсягами неструктурованих даних, що робить їх менш придатними для сучасних додатків, які вимагають масштабованості та гнучкості.

Однак знання з науки про дані та обчислювальні ресурси для навчання моделей необхідні для інтеграції моделей штучного інтелекту або алгоритмів машинного навчання в системи управління інформацією. Незважаючи на те, що ці алгоритми можуть багато чого запропонувати, їхня інтеграція в сучасні системи може бути складнішою і тривалішою.

Здатність алгоритму або моделі пристосовуватися до мінливих потреб або непередбачених труднощів називається адаптивністю. Оскільки алгоритми ШІ та машинного навчання можуть навчатися на нових даних і вдосконалюватися з часом, вони є дуже гнучкими. Завдяки своїй універсальності вони особливо корисні для таких систем, які повинні змінюватися, щоб пристосуватися до нових моделей даних і мінливих організаційних вимог.

Подібно до цього, бази даних NoSQL і хмарні рішення забезпечують гнучкість в управлінні різними типами даних і зміною бізнес-вимог. Вони можуть розвиватися разом із зростаючими потребами організації та легко взаємодіяти з іншими системами.

При оцінці існуючих моделей і алгоритмів необхідно враховувати ряд факторів, щоб переконатися, що вони відповідають цілям і специфікаціям системи управління інформацією.

Найкращі алгоритми та моделі значною мірою визначаються типом даних, незалежно від того, чи є вони неструктурованими, напівструктурованими або структурованими. Реляційні бази даних і SQL-запити ідеально підходять для обробки структурованих даних, тоді як для ефективного структурування і пошуку неструктурованих даних можуть знадобитися NoSQL-бази даних і алгоритми на основі штучного інтелекту.

2.7 Висновки до розділу

У цьому розділі детально розглянуті основні концепції та алгоритми, які є основою для сучасних систем зберігання, організації та пошуку інформації. Проаналізовано теоретичні основи, стратегії організації даних, а також заходи безпеки, що підтримують ефективне управління інформацією. Основною метою було виявити сучасні методи, їхню придатність для конкретної системи, а також те, як ці методи підвищують ефективність, адаптивність і масштабованість інформаційних систем.

У процесі аналізу встановлено кілька ключових закономірностей:

- Ефективність та масштабованість є критичними факторами для системи зберігання та пошуку даних. Алгоритми, такі як хешування та бінарний пошук, значно підвищують швидкість пошуку, забезпечуючи ефективне управління великими обсягами даних. Вони є необхідними елементами архітектури системи для її здатності працювати з великими, динамічними обсягами даних і забезпечувати високу продуктивність.

- Безпека та цілісність даних — важливі складові для будь-якої системи управління інформацією. Виявлено, що для захисту конфіденційних даних від несанкціонованого доступу необхідно застосовувати сучасні методи шифрування та контроль доступу. Ці механізми гарантують високу безпеку даних та збереження їх цілісності навіть у разі зовнішніх загроз.
- Інтеграція штучного інтелекту та машинного навчання відкриває нові можливості для удосконалення системи. Штучний інтелект дозволяє прогнозувати тенденції та приймати рішення на основі великих обсягів даних, що значно покращує функціональність системи, дозволяючи їй адаптуватися до нових вимог бізнесу і користувачів.
- Впровадження сучасних технологій: Системи, що включають такі інновації, як машинне навчання, дозволяють не лише вирішувати поточні задачі з управління даними, але й бути готовими до викликів, пов'язаних із збільшенням складності обробки та аналізу даних. Сучасні алгоритми та стратегії, включаючи алгоритми безпеки, дозволяють забезпечити масштабованість і конкурентоспроможність системи в майбутньому.

Виходячи з досліджених концепцій та алгоритмів, можна зробити висновок, що сучасні системи управління інформацією мають бути побудовані на поєднанні традиційних підходів (як реляційні бази даних) з новітніми технологіями, зокрема, штучним інтелектом і алгоритмами безпеки. Це забезпечить не лише вирішення поточних задач, але й дозволить системі адаптуватися до нових, мінливих вимог і забезпечити її стабільність, безпеку та високу ефективність на довготривалий період. Тому, наступними етапами в розробці цієї системи будуть інтеграція цих алгоритмів у практичну реалізацію, тестування їх функціональності та постійне вдосконалення на основі реальних відгуків користувачів і випадків застосування.

РОЗДІЛ 3
РОЗРОБКА АЛГОРИТМУ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ ЗБЕРІГАННЯ ТА ОРГАНІЗАЦІЇ ІНФОРМАЦІЇ

3.1 Вплив управління інформацією на організаційну ефективність та прийняття рішень

Щодня кожен з нас генерує велику кількість інформації: в інтернеті, на фізичних носіях. Це створило потребу в упорядкованому сховищі, де можна швидко і легко отримати доступ до необхідних даних. Особливо це важливо для корпоративного середовища де актуальність та легкість доступу до інформації є критично важливими.

Упрвління інформацію включає в себе наступні складові:

- Зберігання інформації
- Пошук інформації
- Забезпечення цілісності даних
- Збір інформації
- Актуальність інформації та її постійне оновлення
- Розповсюдження знань
- Безпека даних

Ефективність цих пунктів напряму впливає на швидкість прийняття рішень. Сучасна система зберігання та організації інформації повинна в першу чергу вирішувати ці проблеми. Компанії приймають рішення на основі якості та повноти інформації, не ефективна система може викликати труднощі в забезпеченні цих потреб. Також система повинна гарантувати абсолютну безпеку інформації та її цілісність. Це можна забезпечити за допомогою розробки авторизованого доступу до системи та використання розподілених хмарних сховищ для зберішання інформації.

Така система повинна вирішувати наступні проблеми:

- Фрагментованість інформації - різні компанії по різному зберігають свої дані, в електроних листах, месенджерах, локально на комп’ютері, що

створює ризики втрати даних. Система повинна забезпечити централізоване зберігання всіх знань в рамках компанії, що усуває проблемо розкиданості даних по різних джерелах

- Проблема підтримки актуальності днаих - з часом будь-яка інформація можу втрпати свою актуальність, що може призвести до різного роду помилок, система поаинна містити механізми відстежування та нагадувань про потребу в оновленні інформації
- Комунікація та підтримка зв’язку між членами команди - система повинна надавати можливості для спілкування між колегами, наприклад залишати коментарі, аудіо повідомлення, реакції. Це допоможе швидше та легше виправляти помилки та збільшувати цінність інформації

Користувацький інтерфейс повинен бути легким для освоєння для швидкого онбордингу нових працівників в команду. Всі елементи повинні дотримуватись стандартів UI/UX дизайну.

Враховуючи всі факти системи для організації та зберігання інформації мають надзвичайно великий вплив на управління інформацією та прийняття рішень.

3.2 Розробка алгоритму зберігання та організації інформації

Нижче представлено детальний опис алгоритмів і моделей, що використовуються для зберігання, пошуку та управління інформацією в системі. Основою системи є React (фронтенд), Node.js (бекенд), PostgreSQL (база даних), Elasticsearch (пошук), а також механізми ролей, прав доступу та шифрування.

3.2.1 Алгоритм збереження інформації

Алгоритм збереження інформації є багатостадійним процесом, що забезпечує ефективність, надійність та безпеку даних під час їх передачі та зберігання. Завдяки взаємодії різних компонентів системи (React, Node.js, PostgreSQL, Elasticsearch, механізм ролей та шифрування), процес організовано так, щоб кожен крок забезпечував необхідні функціональні та нефункціональні вимоги.

Етапи алгоритму:

1. Клієнтська ініціалізація (React)

- Користувач вводить дані у формі на веб-додатку.
- Валідація даних на клієнтському рівні (перевірка обов'язкових полів, форматування та цілісності).
- Створення об'єкта даних у JSON-форматі.
- Відправка запиту на сервер через HTTP POST або REST API.

2. Отримання запиту на сервері (Node.js)

- Сервер приймає HTTP-запит і передає його у middleware для обробки.
- Перевірка автентифікації та авторизації користувача за допомогою JWT-токенів перевіряється валідність сесії. Визначаються роль та права доступу користувача.
- Перевірка цілісності даних: Сервер валідує отриманий об'єкт даних за допомогою Joi.

3. Шифрування конфіденційних даних

- Для конфіденційних полів даних (паролі, токени, приватна інформація) використовується AES-256 алгоритм шифрування.
- Серверний модуль шифрування виконує перетворення даних перед збереженням.

4. Збереження у базу даних PostgreSQL

- Дані, що пройшли валідацію та шифрування, зберігаються у реляційній базі даних PostgreSQL.
- Використовується транзакційний режим для забезпечення цілісності даних.

5. Індексація даних у Elasticsearch

- Для прискорення пошуку збережених даних вони індексуються в Elasticsearch.
- Індексація відбувається паралельно із збереженням у PostgreSQL для мінімізації затримки.

- Дані зберігаються у форматі JSON з урахуванням полів, що будуть використовуватися для пошуку.

6. Повідомлення користувача про успіх операції

- Сервер повертає відповідь про успішне збереження даних на клієнт.
- React оновлює інтерфейс користувача, показуючи повідомлення про успішне виконання операції.

Додаткові особливості алгоритму:

1. Логи та моніторинг:

- Кожна операція логірується для подальшої перевірки або діагностики.
- Використовуються Winston або Morgan для логування дій.

2. Обробка помилок:

- Якщо під час збереження виникає помилка (наприклад, збій бази даних), система повертає користувачу відповідь про помилку.

3. Резервне копіювання:

- Регулярне резервне копіювання даних у PostgreSQL забезпечує додаткову безпеку.

4. Балансування навантаження:

- Використання Load Balancer на рівні Node.js для розподілу навантаження при збереженні великих обсягів даних.

Розроблено розгалужену діаграму (рис. 3.1), яка демонструє покроковий процес зберігання та обробки інформації в системі. Діаграма відображає взаємодію між різними компонентами додатку, починаючи з введення даних користувачем через інтерфейс React UI і закінчуючи оновленням інтерфейсу в реальному часі. Кожен етап включає важливі кроки, такі як валідація на клієнтській та серверній стороні, шифрування чутливих даних, збереження в базі даних PostgreSQL, індексація в Elasticsearch та запис логів у систему моніторингу. Діаграма також передбачає обробку помилок на кожному етапі і забезпечує ефективне управління доступом за допомогою перевірки автентифікації та ролей. Завдяки такій розгалуженій структурі, система здатна ефективно обробляти запити, забезпечуючи високий рівень безпеки, продуктивності та зручності для кінцевого користувача.

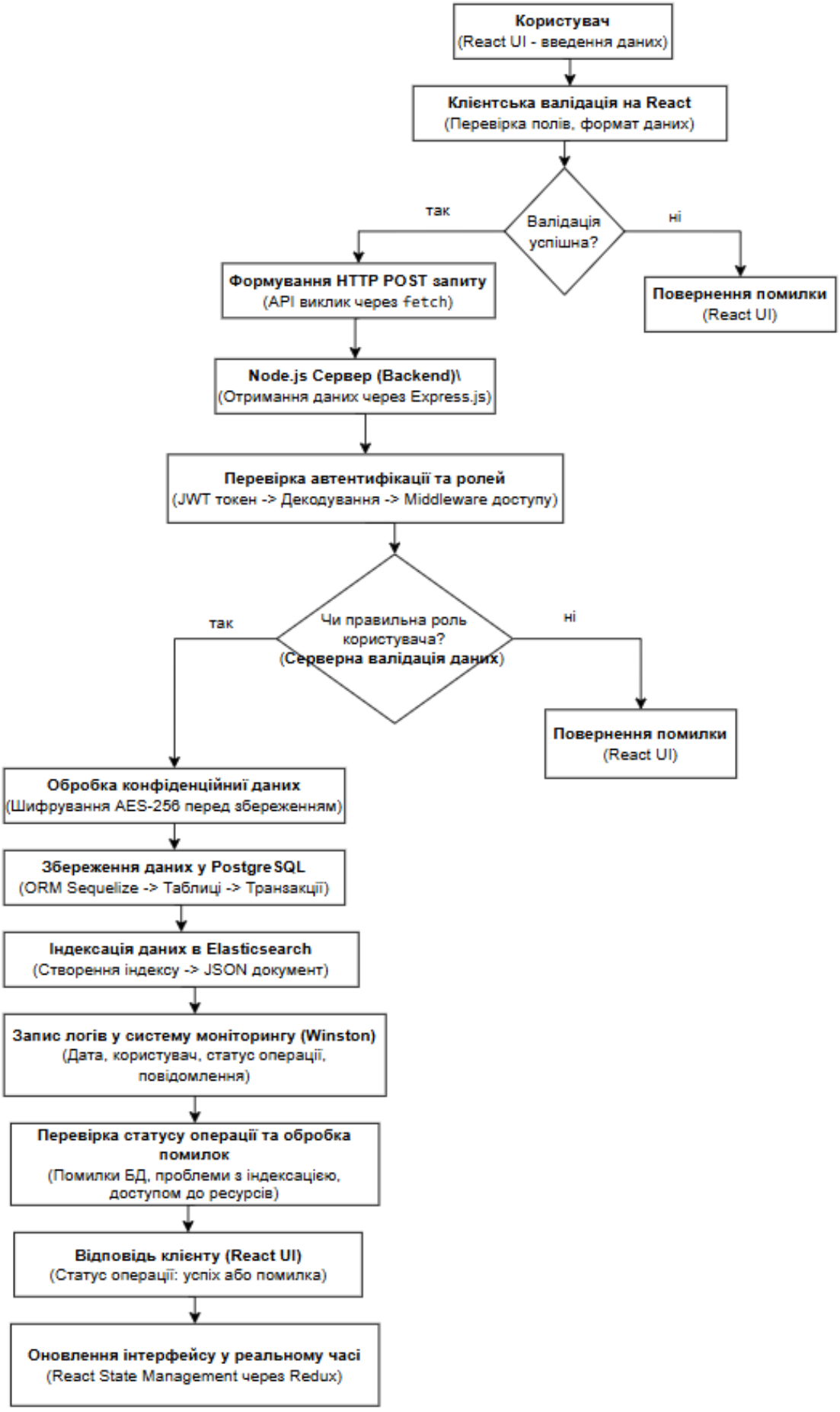

Рис. 3.1. Алгоритм збереження інформації

3.2.2 Алгоритм організації та пошуку інформації

Для зберігання даних використовується реляційна база даних PostgreSQL. Дані зберігаються у вигляді таблиць, де кожна таблиця відповідає певному об'єкту системи (користувачі, документи, метадані тощо).

Основні аспекти організації даних:

1. Створення таблиць та зв'язків:
   - Використання ORM Sequelize або SQL-запитів для створення структур бази даних.
   - Нормалізація даних: розбиття великих таблиць на менші для усунення надлишкових даних.
2. Шифрування чутливих даних:
   - Перед збереженням у БД чутливі дані (наприклад, паролі або конфіденційні поля) шифруються за допомогою AES-256.
   - Це забезпечує захист від несанкціонованого доступу, навіть якщо база даних буде скомпрометована.
3. Індексація даних:
   - Для підвищення швидкодії пошуку застосовуються індекси на ключові поля.

Для забезпечення швидкого пошуку та фільтрації даних використовується Elasticsearch — інструмент, що працює на основі інвертованого індексу. Дані з PostgreSQL синхронізуються з Elasticsearch у форматі JSON.

1. Процес індексації даних:
   - Формування JSON-документа: Дані з PostgreSQL перетворюються у JSON-формат для Elasticsearch.
2. Передача даних до Elasticsearch:
   - Використовується бібліотека elasticsearch.js для взаємодії з Elasticsearch.
   - Індексація документа відбувається за допомогою REST API.
3. Інкрементальна синхронізація:
   - Під час створення, оновлення чи видалення даних у PostgreSQL виконується оновлення індексу в Elasticsearch.

Пошук у системі здійснюється на основі індексованих даних в Elasticsearch. Користувач вводить запит через інтерфейс (React), який передається на сервер (Node.js), а потім до Elasticsearch.

Основні типи пошуку:

1. Прямий пошук за ключовими словами: Використовується для пошуку документів за заголовком, змістом або іншими полями.

2. Фільтрація за метаданими: Можливість пошуку з додатковими умовами (наприклад, за датою створення чи автором).

3. Пошук з ранжуванням: Для більш точного пошуку використовується PageRank-подібна стратегія, що враховує релевантність документа до запиту користувача:

- Пріоритет надається заголовкам та часто використовуваним термінам.

4. Автодоповнення та пошук за підрядками: Під час введення користувачем запиту здійснюється пошук за частковими збігами:

Після отримання результатів пошуку від Elasticsearch:

1. Node.js сервер форматує отримані дані у зрозумілий вигляд.

2. Результати передаються на фронтенд через HTTP-відповідь.

3. React-інтерфейс відображає результати у зручному вигляді (списки, картки з документами тощо).

3.3 Проєктування та розробка архітектури централізованої системи управління інформацією

3.3.1. Проектування та аналіз вимог до системи

При виборі архітектруи важливо розуміти, які користувачі будуть використовувати систему та з яких пристроїв. Проаналізувавши потреби та проблеми було вирішено обрати клієнт-серверну архіотектуру.

Перш ніж розробляти архітектуру системи, дуже важливо було повністю зрозуміти унікальні вимоги та труднощі, з якими стикаються компанії. Для збору вимог використовувалися інтерв'ю, анкети та сесії зворотного зв'язку з потенційними

користувачами, включаючи членів команди з різних сфер, таких як операційна діяльність, маркетинг, фінанси та людські ресурси.

Враховуючи всі фактори виникає потреба вирішення важливих системних вимог:

- Централізоване сховище інформації: Користувачам потрібна була платформа, яка б дозволила їм зберігати всі організаційні ресурси, документи та дані в одному місці.
- Доступ у режимі реального часу та оновлення: Щоб гарантувати, що рішення ґрунтуються на найсвіжіших наявних даних, інформація мала бути доступною в режимі реального часу.
- Інструменти для співпраці: Важливо було включити такі функції, як коментарі, анотації та можливості спільного використання, які б дозволили командам ефективно працювати разом.
- Безпека та контроль доступу: Щоб гарантувати конфіденційність і відповідність нормативним вимогам, конфіденційні дані потребували надійних заходів безпеки з детальним контролем доступу користувачів.
- Інтеграція з поточними додатками: Система повинна була мати можливість легко взаємодіяти з іншими широко використовуваними програмами, такими як програмне забезпечення для управління проектами, хмарні сховища та електронна пошта.

Ці специфікації послужили основою для архітектури, яка зосередилася на створенні орієнтованої на користувача, безпечної та масштабованої платформи, яка могла б задовольнити потреби як малого, так і великого бізнесу.

Архітектура системи була створена з урахуванням продуктивності, масштабованості та адаптивності.

Після створення архітектури системи було розпочато розробку елементів, які б вирізняли систему як комплексне рішення для управління та зберігання інформації. Основними особливостями платформи є наступні:

- Усі записи, файли та документи організації об'єднані в єдину систему. Співробітники можуть легко отримати доступ до найновіших версій

документів і позбутися необхідності використовувати окремі системи зберігання файлів. Система гарантує, що файли будуть надійно збережені та доступні за потреби, взаємодіючи зі сторонніми сервісами, такими як AWS S3.

- Завдяки платформі члени команди можуть легше співпрацювати. Користувачі можуть ділитися файлами з колегами з різних відділів, надавати відгуки та коментувати документи. Технологія забезпечує простий підхід до роз'яснення ідей або надання зворотного зв'язку, підтримуючи як голосові нотатки, так і текстові коментарі. Ця функція підвищує ефективність комунікації в гібридних робочих ситуаціях і особливо корисна для віддалених команд.
- Переконатися, що користувачі використовують найновіші версії документів і що вони актуальні, - одне з найважливіших завдань управління інформацією. Автоматизована система контролю версій, відстежує зміни в документах і сповіщає користувачів про появу нових версій. Крім того, платформа пропонує нагадування про оновлення документів, гарантуючи, що важлива інформація завжди актуальна
- Наявність потужних можливостей пошуку має вирішальне значення, оскільки в системі зберігається велика кількість даних. Elasticsearch, розподілена пошукова система, дозволяє користувачам швидко і точно здійснювати пошук у величезних обсягах даних. Щоб забезпечити пошук потрібної інформації з найменшими витратами, користувачі можуть обмежити результати пошуку на основі різних факторів, включаючи ключові слова, тип файлу, статус документа тощо.

3.3.2 Створення Canvas моделі та UML діаграм

Для того, щоб визначити всю бізнес-активність організації, включаючи пропозицію, інфраструктуру та клієнтів, була створена бізнес-модель Canvas для проекту зі створення веб-додатку для зберігання та організації ресурсів та інформації в середовищі Visual Paradigm Online (рис. 3.2).

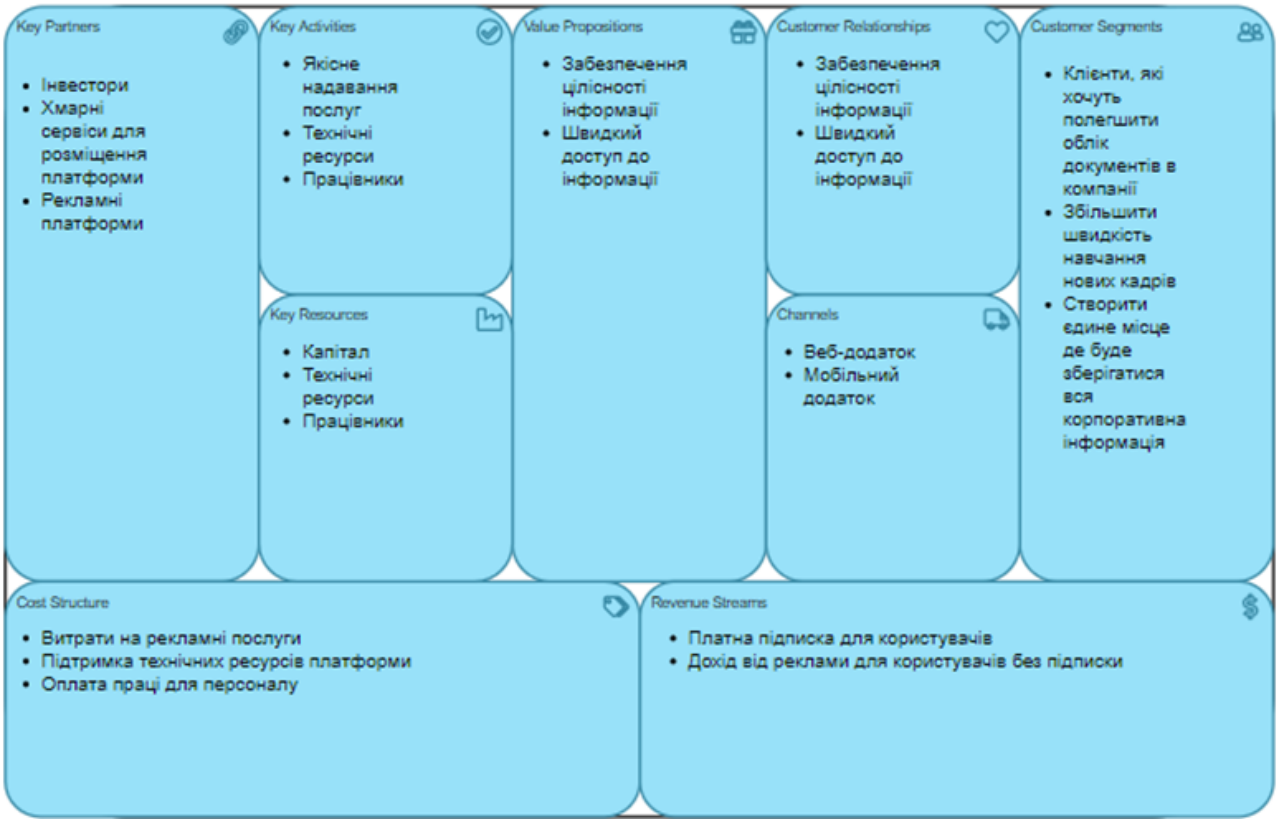

Рис. 3.2. Canvas модель для проекту

Серед користувачів додатку можна чітко визначити три основниі категорії:

- Клієнти, які хочуть полегшити облік документів в компанії;
- Збільшити швидкість навчання нових кадрів;
- Створити єдине місце де буде зберігатися вся корпоративна інформація;

Ключові цінності:

- Забезпечення цілісності інформації;
- Швидкий доступ до інформації;

Канали:

- Веб-додаток;
- Мобільний додаток;

Взаємини з клієнтами:

- Забезпечення цілісності інформації;
- Швидкий доступ до інформації;

Потоки доходів:

- Платна підписка для користувачів;
- Дохід від реклами для користувачів без підписки;

Ключові ресурси:

- Капітал;
- Технічні ресурси;
- Працівники;

Ключові дії:

- Якісне надавання послуг;
- Технічні ресурси;
- Працівники;

Ключові партнери:

- Інвестори;
- Хмарні сервіси для розміщення платформи;
- Рекламні платформи;

Структура витрат:

- Витрати на рекламні послуги;
- Оплата праці для персоналу;
- Підтримка технічних ресурсів платформи;

Далі створюємо UML діаграму станів для процесу Реєстрації (рис. 3.3).

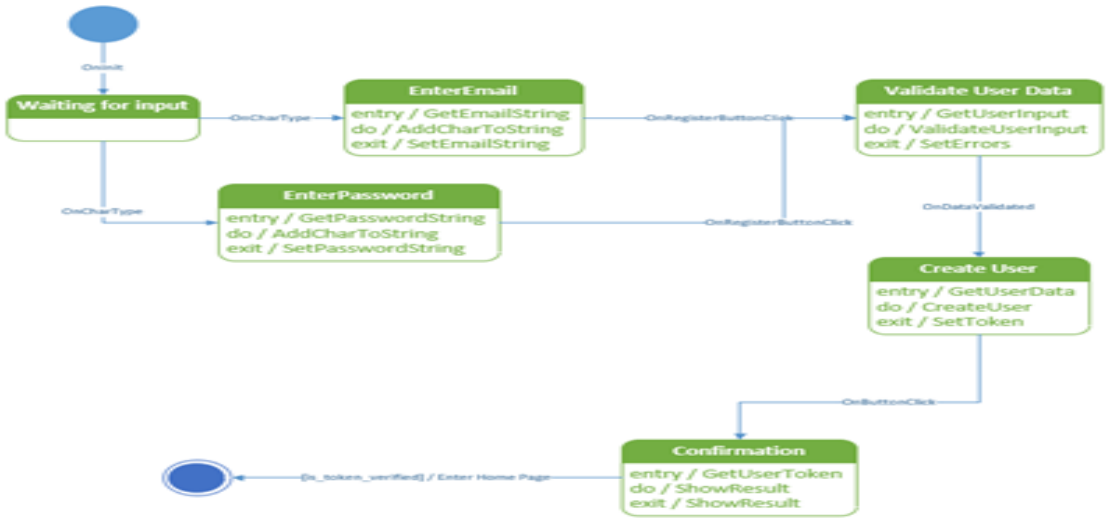

Рис. 3.3. UML-діаграма станів для процесу Реєстрації

Описаний покроковий процес валідації даних користувачів та реєстрації їх у системі.

Також створюю UML діаграму яка відображає покроковий процес авторизації користувача у системі (рис. 3.4).

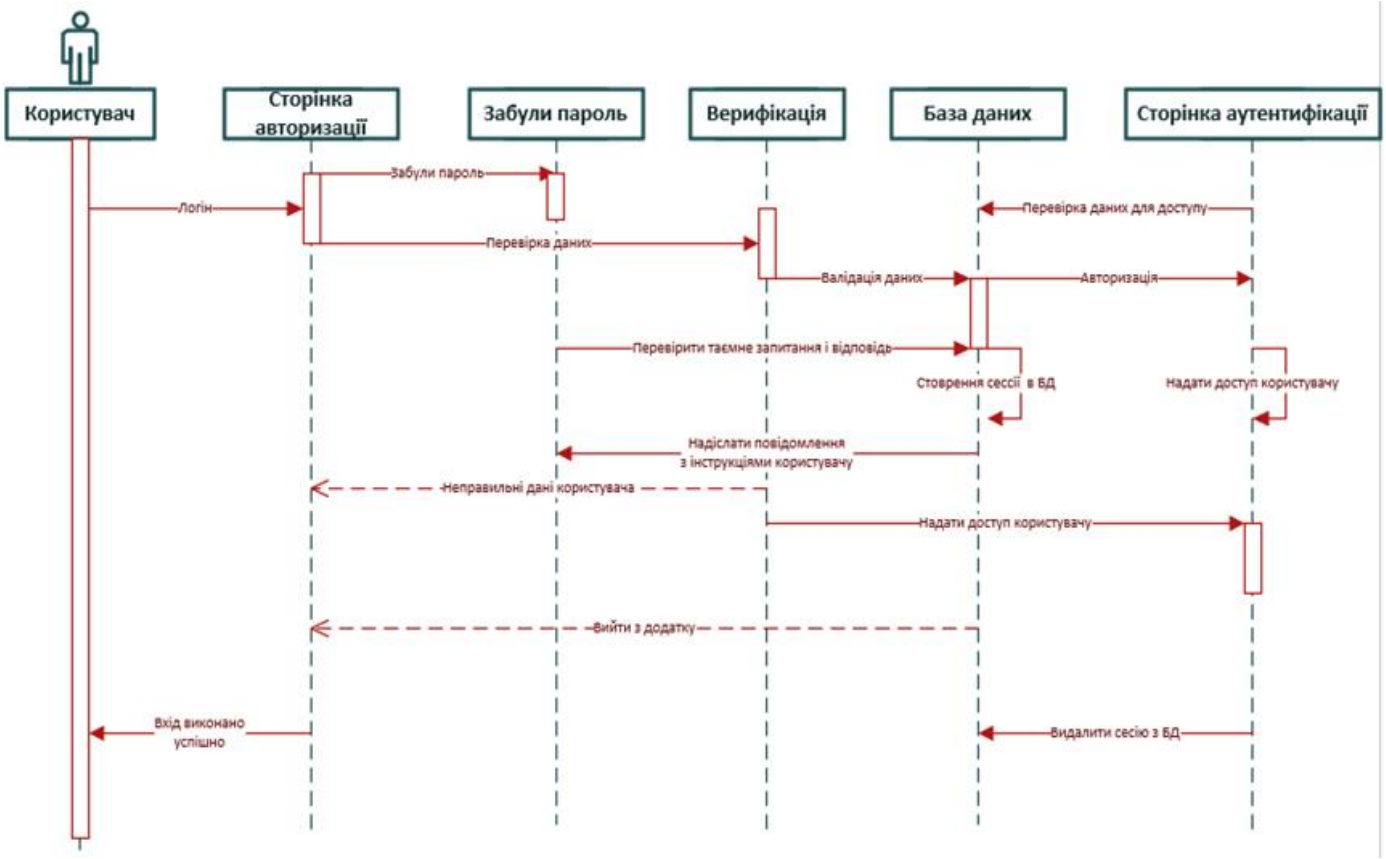

Рис. 3.4. UML-діаграма послідовності для Авторизації.

3.3.3. Вибір технологій для задоволення потреб системи

Веб-додаток буде складатися з клієнтської та серверної частини, використовуючи такі технології як:

- Node.js - платформа яка дозволяє створити сервер на основі двигуна V8 використовуючи мову програмування JavaScript [21].
- Express - бібліотека яка має весь необхідний функціонал для описання основної логіки роботи сервера

- TypeScript - мова програмування яка дозволяє описувати типи та моделі та додає рівень компіляції в JavaScript [15].
- PostgreSQL з TypeORM - база даних SQL, та серверна бібліотека яка створює легкий доступ до бази даниз використовуючи полатформу NodeJS
- WebSocket - технолгія дозволяє реалізувати одночасну взаємодію декількох користувачів (наприклад для редагування сторінок)
- Elasticsearch - гнучке та маштабоване рішення для реалізації пошуку [25].
- AWS S3 - надійне та безпечне сховище для зберігання всієї інформації

Використання цих технологій допоможе створити, ефективний, безпечний, та доступний застосунок [36].

Веб-сервіси часто використовують архітектурний стиль REST (Representational State Transfer - передача представницького стану), як і багато програм на основі Express.js. Це набір рекомендацій, які визначають, як має працювати взаємодія між клієнтом і сервером через протокол HTTP. RESTful API мають бути незалежними від стану, масштабованими та простими у використанні для багатьох клієнтів [22].

Реалізація RESTful архітектури в додатку Express.js передбачає проектування і створення кінцевих точок, які відповідають принципам REST. Ось основні компоненти RESTful-архітектури і те, як вони використовуються в Express.js-проектах:

Основою REST є концепція ресурсів, тобто речей, до яких можна отримати доступ і які можна змінити. Ресурси можуть бути представлені у вигляді маршрутів та кінцевих точок у додатку Express.js. Зазвичай, кожна кінцева точка пов'язана з певним ресурсом або набором ресурсів. Наприклад, /users можна використовувати для представлення групи користувачів, а /users/:id - для представлення конкретної особи.

RESTful API виконують операції над ресурсами за допомогою звичайних HTTP-дієслів (рис. 3.5). Серед них дієслова GET, POST, PUT, PATCH і DELETE. Ви можете створити маршрути в Express.js, які відповідають цим HTTP-дієсловам, і методи контролера подбають про відповідні дії. Наприклад, маршрут POST /users

можна використовувати для додавання нового користувача, а маршрут GET /users - для отримання списку користувачів.

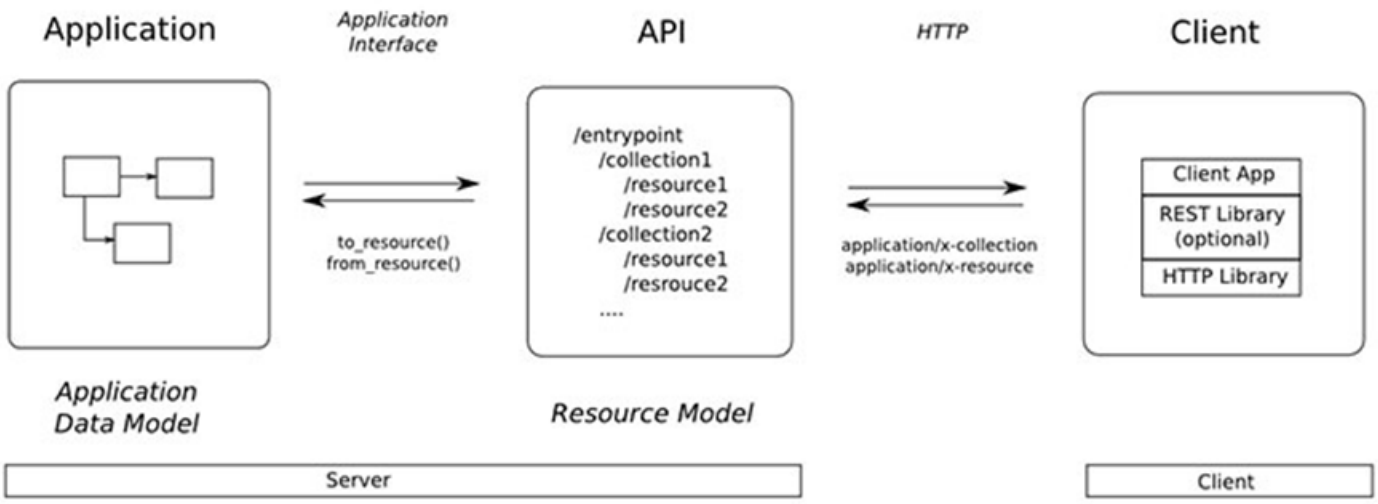

Рис. 3.5. Принци проботи REST

Структура даних, користувацький досвід, безпека та взаємодія з існуючими інструментами і процесами - це лише деякі з багатьох аспектів, які необхідно ретельно враховувати під час проектування та впровадження системи зберігання та організації інформації. Метою розробки системи є створення потужної, ефективної та інтуїтивно зрозумілої платформи, яка б централізувала дані, полегшувала прийняття рішень між командами та відділами, підвищувала співпрацю та покращувала доступ до знань.

3.3.4 Забезпечення якості та тестування

Процес розробки будь-якої системи повинен включати тестування. Платформа була протестована в кілька етапів, включаючи:

- Модульне тестування: Бібліотека тестування Jest була використана для перевірки точності кожного модуля і компонента системи.
- Інтеграційне тестування: Щоб гарантувати безперебійний потік даних, було ретельно перевірено взаємодію фронтенду, бекенду та бази даних між собою.

- Тестування прийнятності для користувача (UAT): Система була протестована групою користувачів з цільової аудиторії, щоб переконатися, що вона є зручною та відповідає вимогам.

3.4 Програмна реалізація системи для зберігання та організації інформації

Реальна реалізація програмного забезпечення, включаючи кодування, налагодження та тестування, відбувається на етапі розробки. Цей розділ пропонує детальну інформацію про методологію розробки

Перед початком розробки ретельно вивчаються вимоги, отримані на ранніх етапах проекту. Це допомагає визначити очікувану функціональність програми, а також основні функції та обсяг роботи. Процес розробки будується на основі вимог, які спрямовують команду розробників на створення надійного та зручного для використання продукту.

3.4.1 Створення клієнтської частини системи

Створено новий React-додаток за допомогою команди npx create-react-app. Create React App пропонує лише інструкції для створення фронтенду; він не керує базою даних чи бекендом. Він оптимізує код всередині за допомогою webpack і Babel.

Поділ проекту на кілька модулів полегшує навігацію файлами та створює основу для майбутніх удосконалень проекту. Логічні компоненти додатку розташовані в різних папках.

Кожен компонент представляє собою незалежну частину функціональності, що полегшує його тестування, повторне використання та підтримку. Наприклад, компоненти користувацького інтерфейсу (UI) можна розмістити в папці components, а специфічні для бізнес-логіки функції — у папці components.

Для забезпечення ефективності роботи та спрощення процесу налагодження використовуються такі інструменти, як ESLint для перевірки коду на помилки, Prettier для автоматичного форматування та Jest для тестування компонентів.

За допомогою бібліотеки React Router додається підтримка багатосторінкового інтерфейсу. Наприклад, конфігурація маршрутизації може включати такі шляхи, як /, /about, /contact. Кожен шлях відповідає окремому компоненту або сторінці.

Для стилізації компонентів можуть використовуватися CSS-модулі, Styled Components або інші популярні бібліотеки, такі як Tailwind CSS чи Material-UI. Це дозволяє легко підтримувати єдиний стиль та створювати адаптивний дизайн. Для стилізації компонентів можуть використовуватися CSS-модулі.

Розроблено структуру папок для різних типів файлів (рис. 3.6).

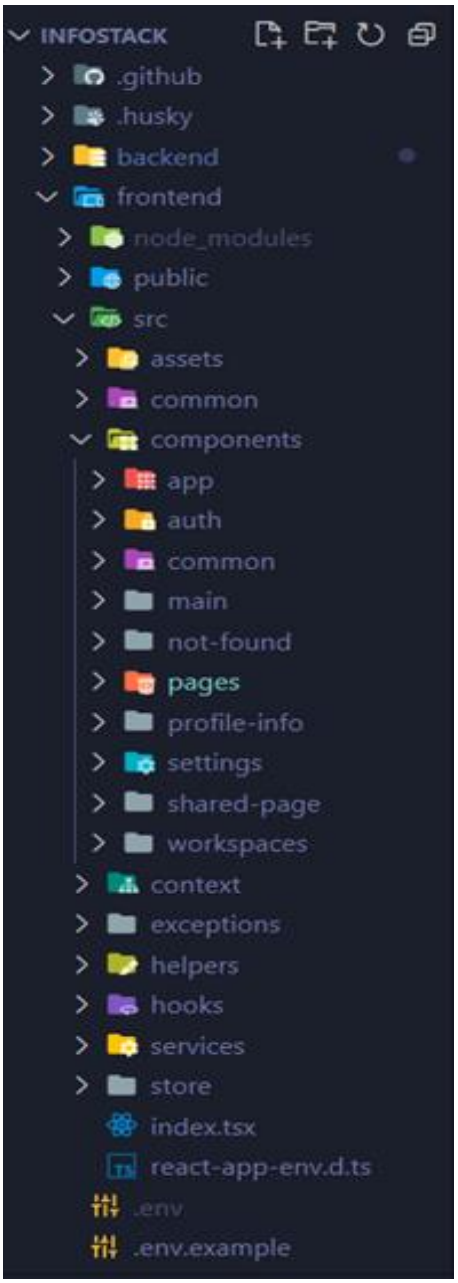

Рис. 3.6. Структура папок

Встановлюю потрібні бібліотеки за допомогою пакетного менеджера Node Package Manager. Перелік встановлених бібліотек (рис. 3.7):

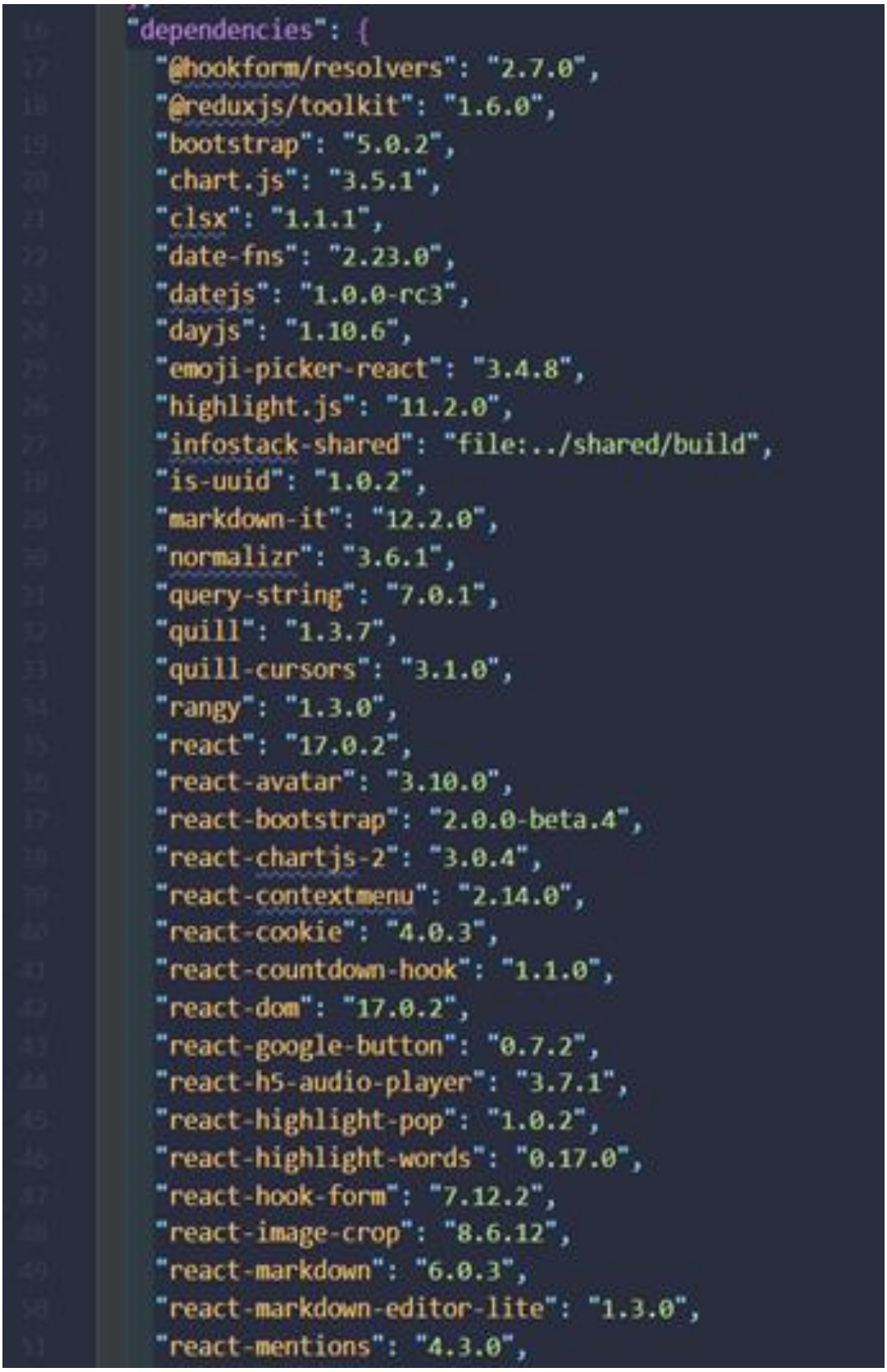

Рис. 3.7. Перелік встановлених бібліотек

Створюю компоненти які відповідають за процес авторизації користувачів (рис. 3.8).

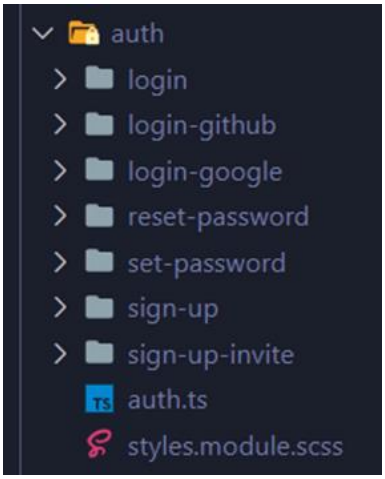

Рис. 3.8. Перелік компонентів для авторизації

Також розробляю спеціальні компоненти які будуть використовуватись у всьому додатку (рис. 3.9).

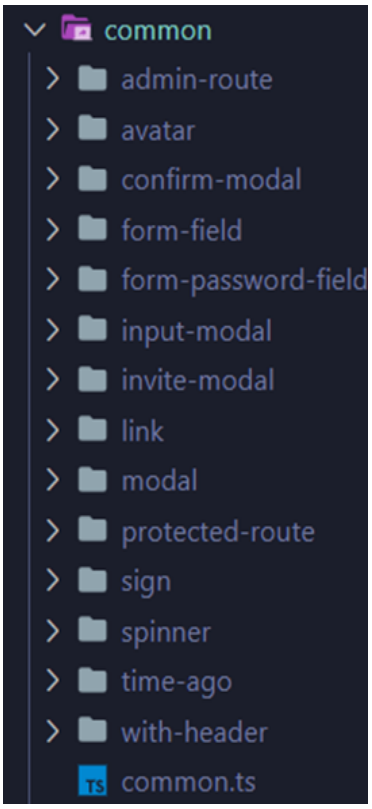

Рис. 3.9. Перелік спільних компонентів

Переглядаю усі створені компоненти (рис. 3.10):

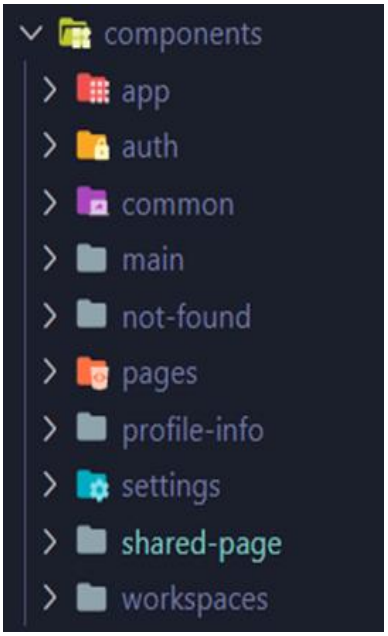

Рис. 3.10. Перелік усіх компонентів

У головному компоненті App.tsx (рис. 3.11) створюється структура маршрутизації додатка. За допомогою налаштування роутів визначається логіка навігації між різними сторінками або функціональними частинами веб-застосунку. Це дозволяє забезпечити інтуїтивну і швидку взаємодію користувача із системою, динамічне завантаження контенту без перезавантаження сторінки, а також централізоване управління навігаційними шляхами.

У цьому компоненті реалізується логіка переходів між різними секціями системи, що дозволяє користувачам легко переміщуватися між функціональними частинами додатку. Такий підхід сприяє організованій структурі додатку та забезпечує швидку і зручну роботу з ним.

Додатково, компонент App.tsx інтегрує глобальні налаштування, такі як теми, контексти або обробка станів, що забезпечує узгодженість роботи всього додатку.

Цей компонент служить точкою входу для всього додатку, забезпечуючи взаємодію між усіма його модулями. Завдяки організованій структурі роутів, система легко масштабується та підтримує динамічне додавання нових функціональних можливостей.

frontend > src > components > app > app.tsx > ...

```tsx
import { ToastContainer } from 'react-toastify';
import { AppRoute } from 'common/enums';
import { ProtectedRoute, Route, Switch } from 'components/common/common';
import {
  Login,
  LoginGoogle,
  LoginGitHub,
  SignUp,
  SignUpInvite,
  SetPassword,
  ResetPassword,
} from 'components/auth/auth';
import Workspaces from 'components/workspaces/workspaces';
import Main from 'components/main/main';
import NotFound from 'components/not-found/not-found';
import { SharedPage } from 'components/shared-page/shared-page';

const App: React.FC = () => {
  return (
    <>
      <Switch>
        <Route path={AppRoute.LOGIN} component={Login} exact />
        <Route path={AppRoute.LOGIN_GOOGLE} component={LoginGoogle} exact />
        <Route path={AppRoute.LOGIN_GITHUB} component={LoginGitHub} exact />
        <Route path={AppRoute.SIGN_UP} component={SignUp} exact />
        <Route path={AppRoute.INVITE} component={SignUpInvite} exact />
        <Route path={AppRoute.RESET_PASSWORD} component={ResetPassword} exact />
        <Route path={AppRoute.SET_PASSWORD} component={SetPassword} exact />
        <Route path={AppRoute.SHARE} component={SharedPage} />
        <ProtectedRoute
          path={AppRoute.WORKSPACES}
          component={Workspaces}
          exact
        />
        <ProtectedRoute path={AppRoute.ROOT} component={Main} />
        <Route path="*" component={NotFound} />
      </Switch>
      <ToastContainer />
    </>
  );
};

export default App;
```

Рис. 3.11. Код компонента App

Далі розробляю компоненти, які будуть виконувати функцію сторінок (рис. 3.12):

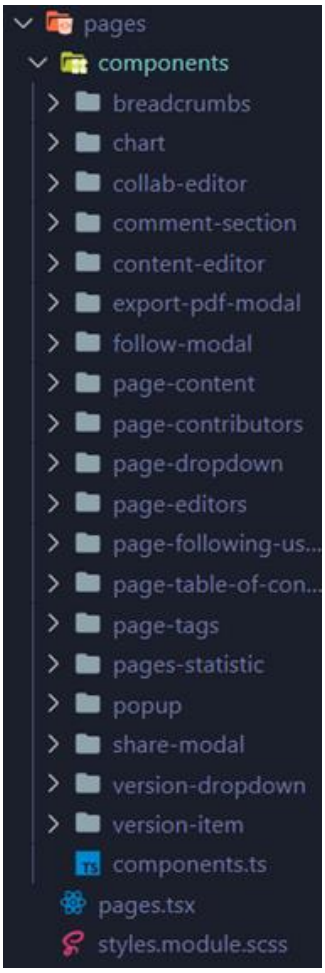

Рис. 3.12. Перелік сторінок

В даному розділі було розглянуто основні компоненти з яких складається веб сайт, конфігурацію та налаштування проекту

3.4.2 Реалізація серверної частини

Потужна інфраструктура програмного забезпечення підтримується різноманітними внутрішніми технологіями.

Авторизація та аутентифікація здійснюється за допомогою веб-токенів JSON. Мініатюрна, автономна технологія JWT може бути використана для безпечної передачі даних авторизації та автентифікації між сторонами. Використовуючи JWT,

сервер може зберігати стан сеансу між запитами, автентифікувати користувачів і надавати безпечний контроль доступу до своїх ресурсів.

Ці внутрішні технології працюють разом, щоб створити міцну основу, яка забезпечує масштабовану продуктивність, безпечне зберігання файлів, ефективний пошук, надійне управління даними та комунікацію в режимі реального часу.

Завдяки продуманому підбору та інтеграції цих технологій система здатна запропонувати багатофункціональне, високопродуктивне програмне рішення для ефективної організації та адміністрування інформації.

Код з головного файлу сервера, тут відбувається його конфігурація та запуск (рис. 3.13).

```
const app: Express = express();
const httpServer = createServer(app);

const io = new Server(httpServer, {
  cors: {
    origin: nodeEnv === 'production' ? url : 'http://localhost:3000',
    methods: ['GET', 'POST'],
  },
});

io.on(SocketEvents.CONNECTION, socketHandlers);

app.use(cors());
app.use(express.static(path.join(__dirname, './public')));
app.use(express.json({ limit: '50mb' }));
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());

app.use('/api/', authorizationMiddleware, socketMiddleware(io));

routes(app);

app.use(errorHandlerMiddleware);

app.use('*', (_req, res) => {
  res.sendFile(path.join(__dirname, '/public/index.html'));
});

httpServer.listen(port, async () => {
  try {
    await createConnection(ormconfig);
  } catch (error) {
    logger.info(`App started with error: ${error}`);
  }
  logger.info(`Server is running at ${port}.`);
});

export default app;
```

Рис. 3.13. Основний код сервера

Далі розробляю необхідні сервіси для взаємодії з базою даних (рис. 3.14).

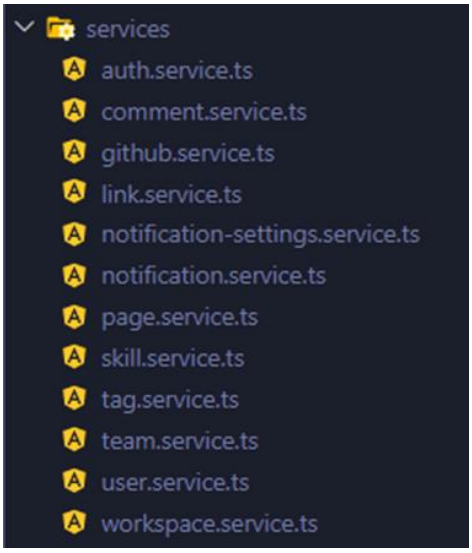

Рис. 3.14. Сервіси

Після цього створюю усі необхідні роути, які будуть обробляти запити користувачів, та утилізувати сервіси для знаходження необхідних даних (рис. 3.15).

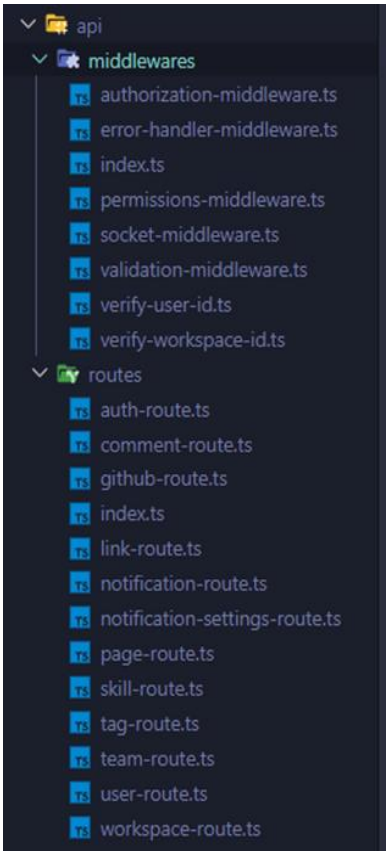

Рис. 3.15. Список API

Налаштовую базу даних. Для цього використовую змінні оточення де зберігається інформація для з’єднання з базою даних (рис. 3.16).

```
const getDbConfig = (): PostgresConnectionOptions => {
  const ssl =
    env.app.nodeEnv === 'production'
      ? {
        rejectUnauthorized: false,
      }
      : false;

  const baseConfig: PostgresConnectionOptions = {
    type: 'postgres',
    migrationsRun: env.db.migrationsRun,
    migrations: [env.db.migrationsDir],
    entities: [env.db.enititiesDir],
    synchronize: env.db.synchronize,
    logging: env.db.logging,
    ssl,
  };

  return {
    ...baseConfig,
    host: env.db.host,
    port: env.db.port,
    username: env.db.username,
    password: env.db.password,
    database: env.db.name,
  };
};

export default getDbConfig();
```

Рис. 3.16. Конфігурація бази даних

Також будую схему бази даних (рис 3.17), яка описує всі сутнорсті, їх поля та зв’язки:

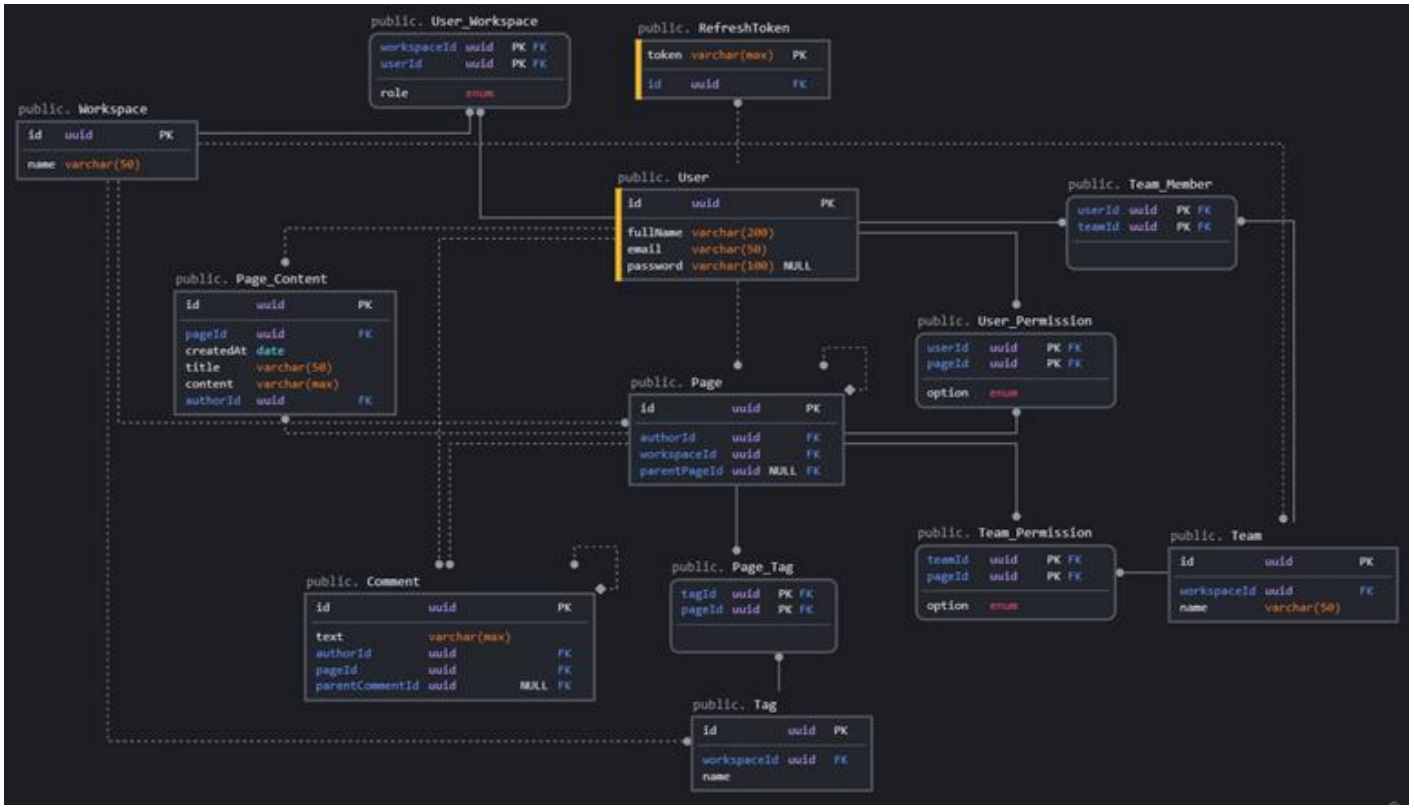

Рис. 3.17. Схема бази даних

3.4.3 Розгортання системи

Процедура розгортання має важливе значення для забезпечення доступу користувачів до програми та її використання. Для розгортання ми використовуємо платформу Render.com.

Перед розгортанням ми переконуємося, що необхідна інфраструктура є на місці. Для цього серверні ресурси, такі як процесор, пам'ять і сховище, повинні бути налаштовані відповідно до специфікацій програмного забезпечення. Вимоги до інфраструктури системи задовольняє Render.com, який пропонує надійну та розширювану платформу для розміщення веб-додатків.

Сервер має надійну та масштабовану інфраструктуру, швидкі процеси розгортання, а також потужні можливості моніторингу та безпеки. Щоб гарантувати

плавний перехід від розробки до виробництва, розгортання ретельно планується. Це дає користувачам надійну та зручну платформу для ефективного управління, пошуку та зберігання даних.

Проект був розгорнутий за допомогою платформи Render. Для розгортання Render-додатку вам потрібно лише перенести свій код до системи контролю версій. Після того, як ви прив'яжете свій обліковий запис GitHub або GitLab до свого облікового запису Render, Render буде автоматично створювати і розгортати ваші сервіси з кожним поштовхом.

На головній сторінці сервісу Render можемо бачити основну інформацію про послуги які він надає (рис. 3.18).

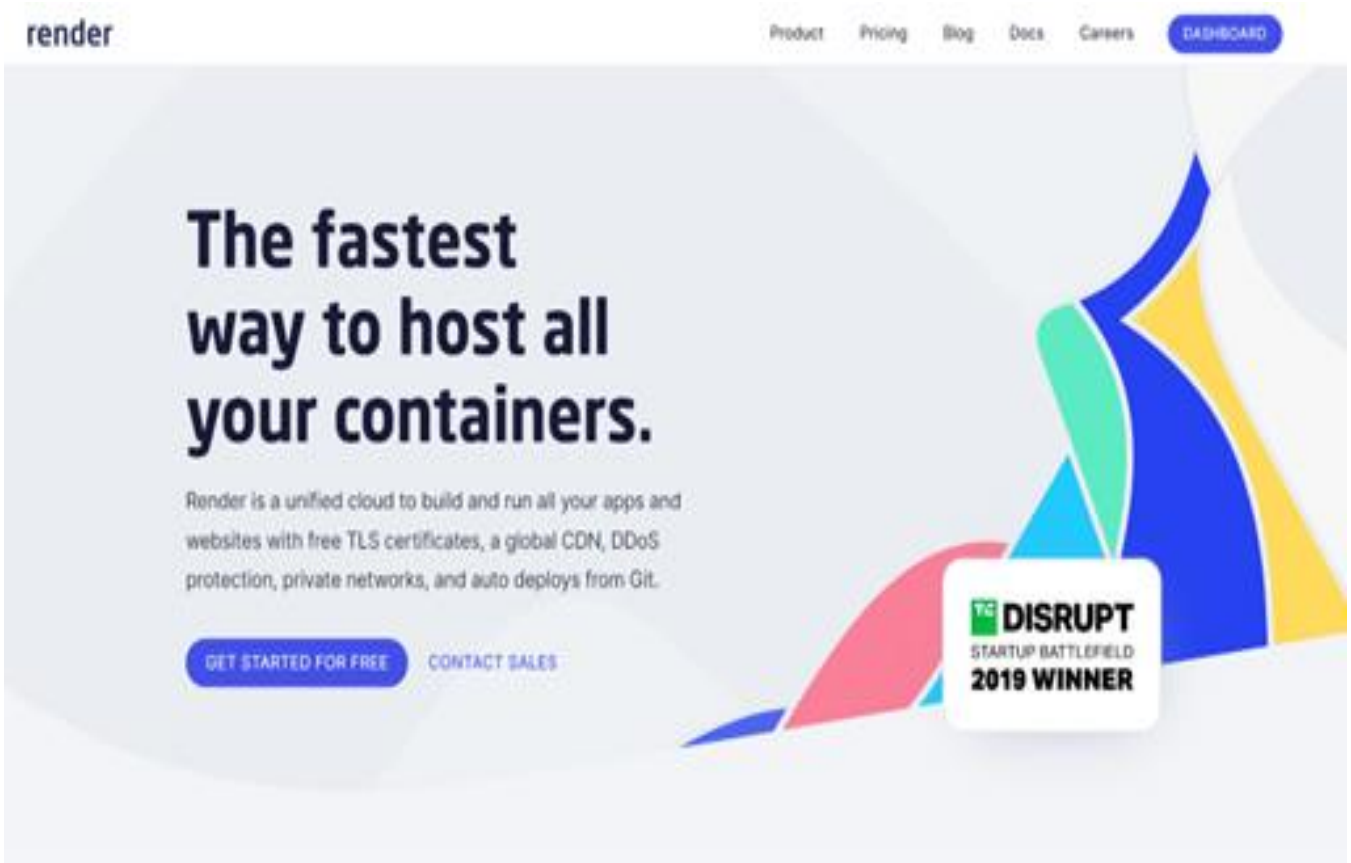

Рис. 3.18. Головна сторінка сервісу Render

Першим кроком є реєстрація на платформі та створення новго веб сервісу. Після авторизації натискаємо кнопку New і обираємо Web Service (рис. 3.19).

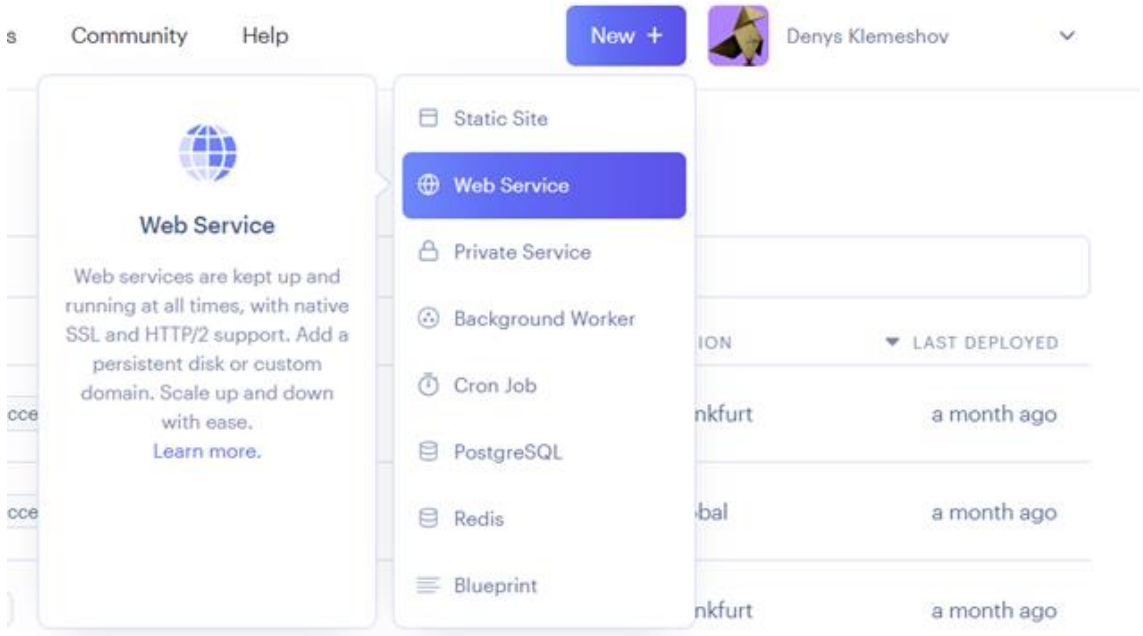

Рис. 3.19. Створення веб сервісу

Далі відбувається підключення репозиторію GitHub, потрібно надати доступ до свого репозиторію сервісу Render. Та натиснути кнопку Connect (рис. 3.19).

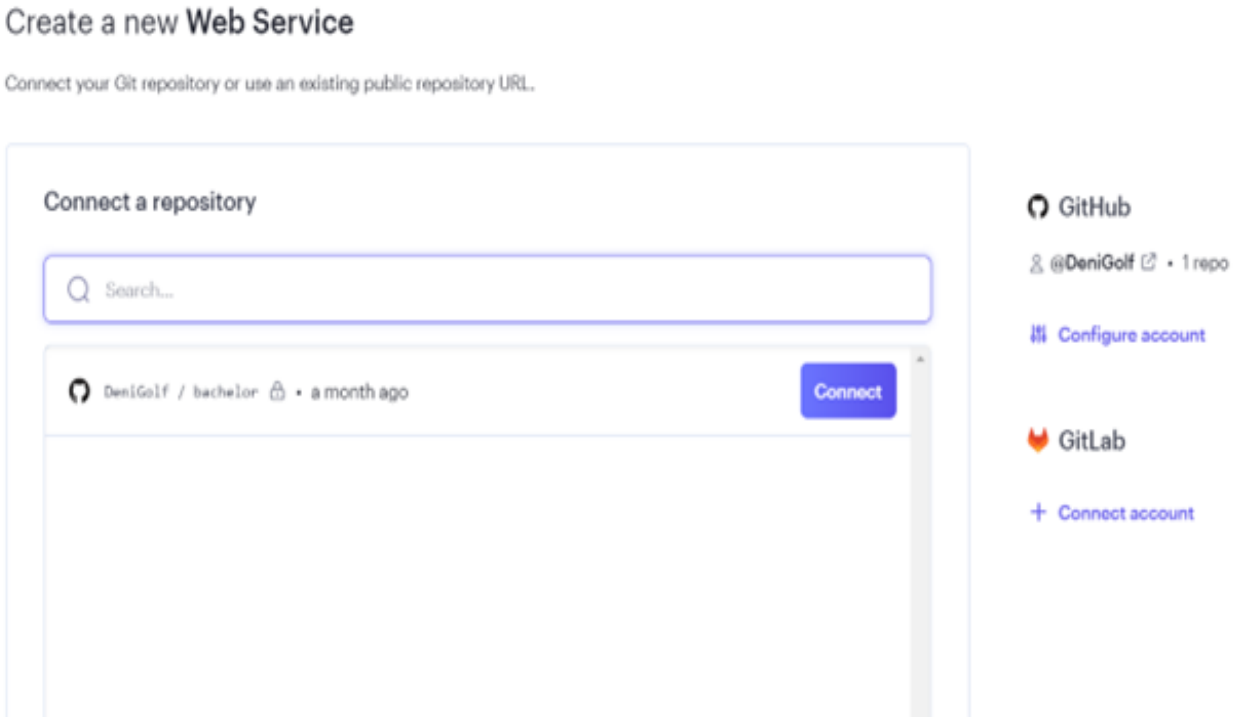

Рис. 3.20. Вибір репозиторія

Наступний крок це заповнення інформації про веб сервіс. Вказуємо назву сервісу, регіон де буде розміщуватись (рис. 3.21), гілку дані з якої буде використано, платформу на якій буде розгортатись додаток, команду для build та start.

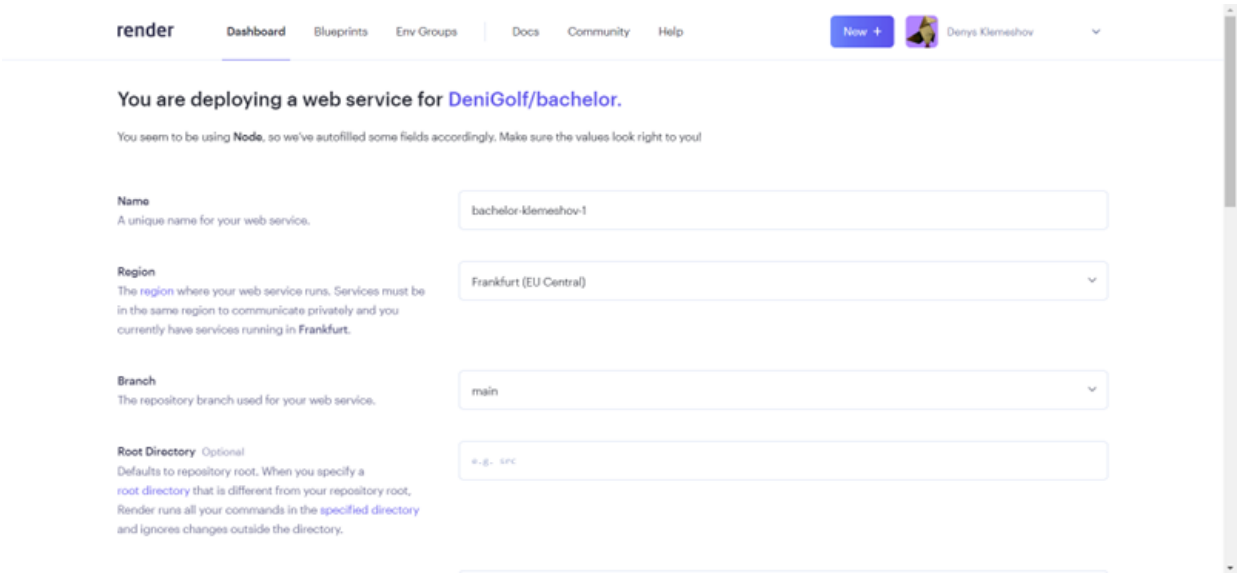

Рис. 3.21. Заповнення інформації про веб сервіс

Далі обираємо тарифний план та настискаємо кнопку Create Web Service (рис. 3.22).

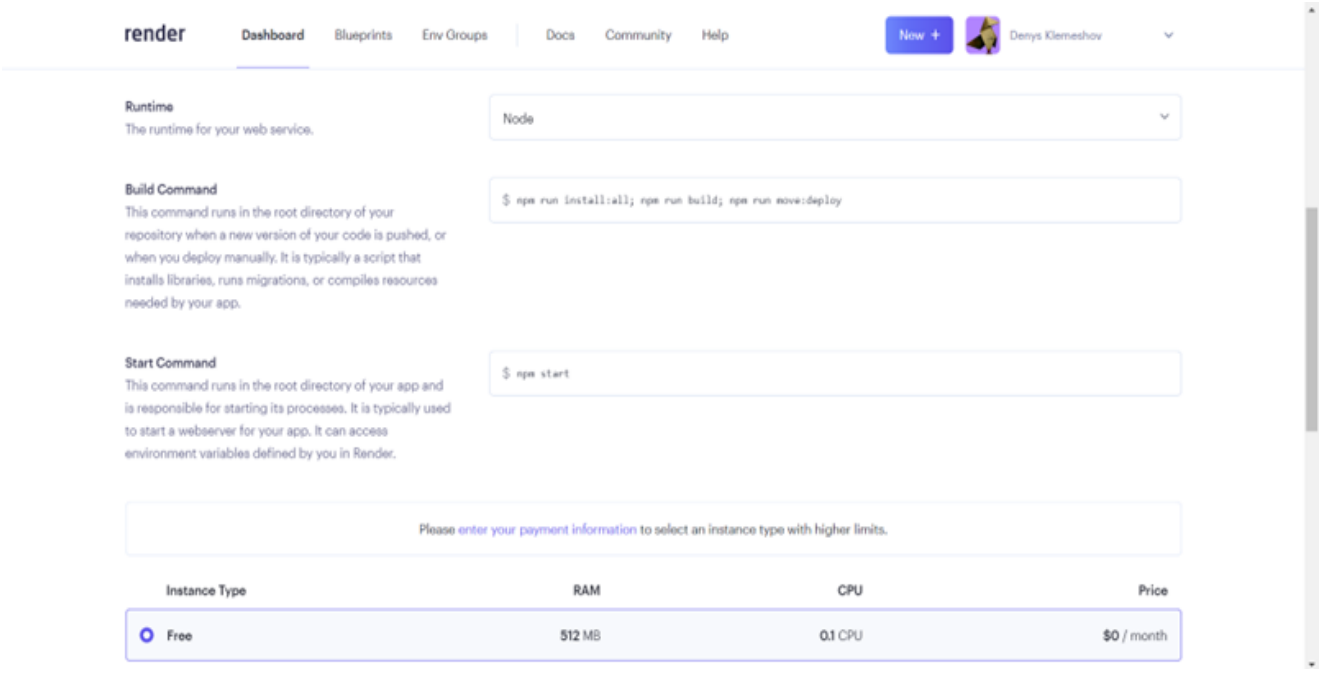

Рис. 3.22. Заповнення інформації про веб сервіс

Відправляю форму для створення сервісу (рис. 3.23).

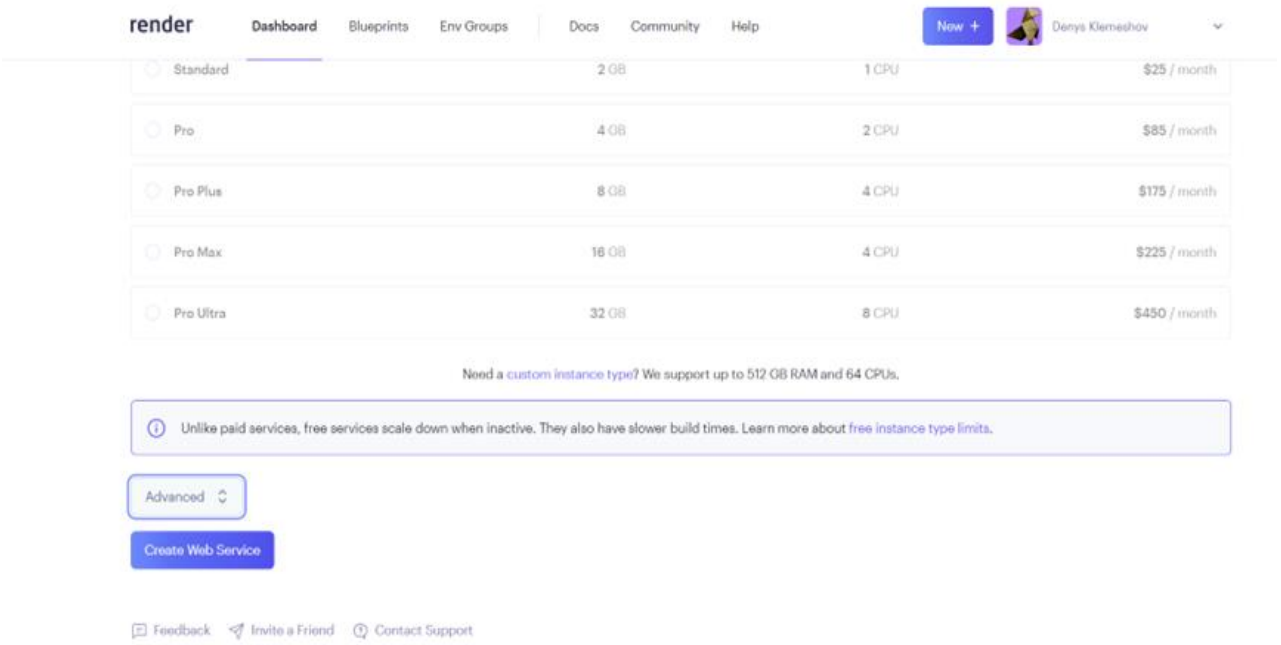

Рис. 3.23. Відправка форми створення веб сервісу

Переглядаю інформації про створений сервіс (рис. 3.24).

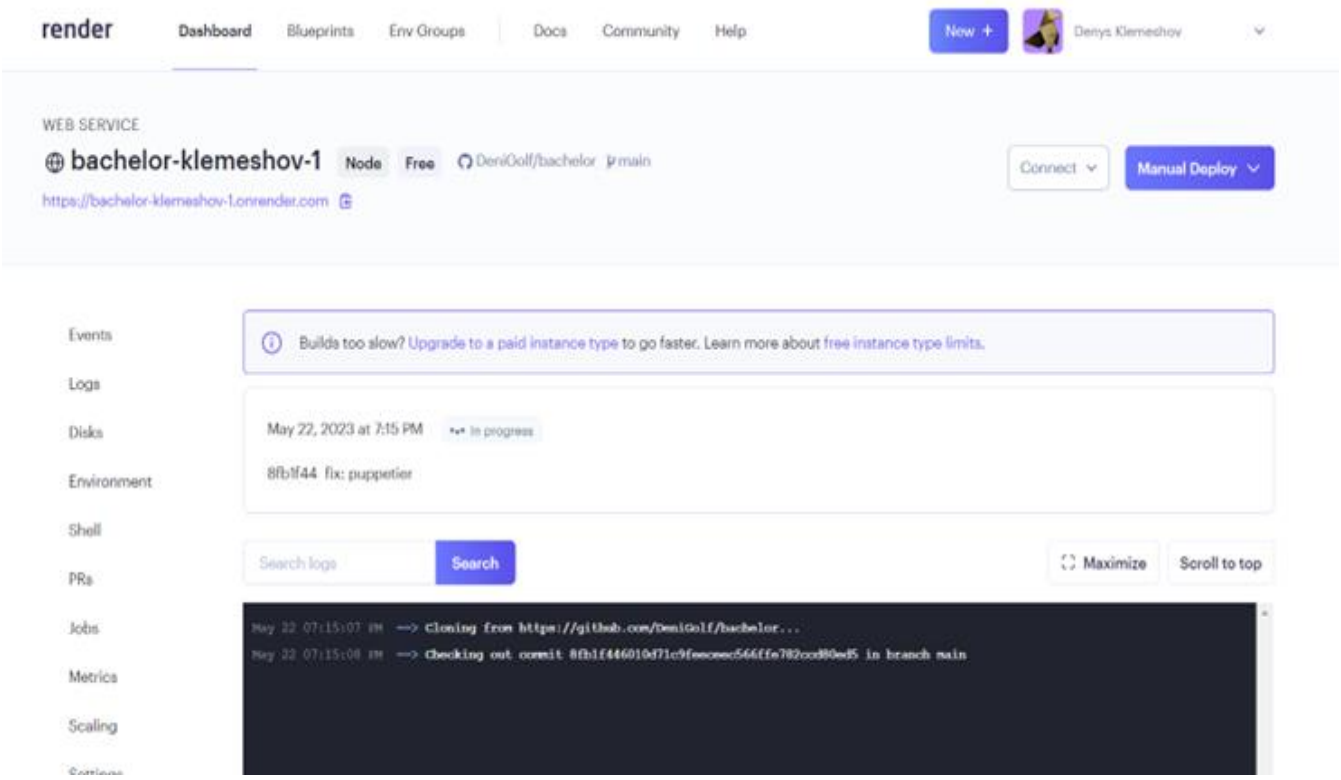

Рис. 3.24. Перегляд інформації про веб сервіс

Потрібно також створити базу даних. Для цього натискаємо New і обираємо PostgreSQL. Вказуємо назву бази даних, регіон, версію, обираємо тарифний план. Натискаємо Create Database (рис. 3.25).

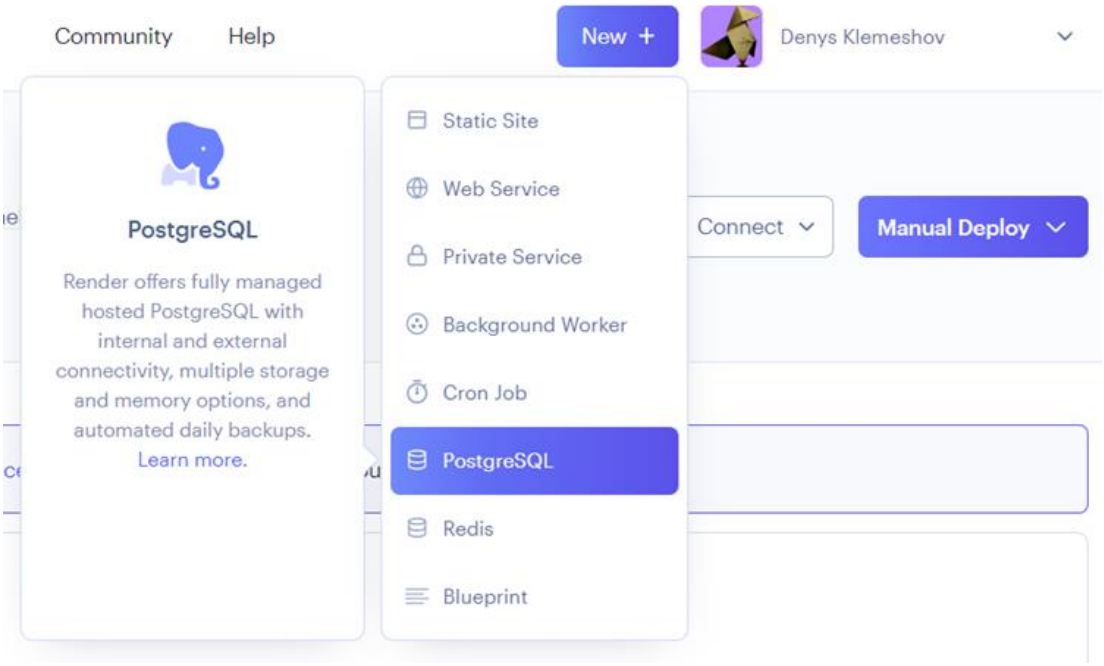

Рис. 3.25. Створення PostgreSQL бази даних

Заповнюю інформацію про базу даних (рис. 3.26).

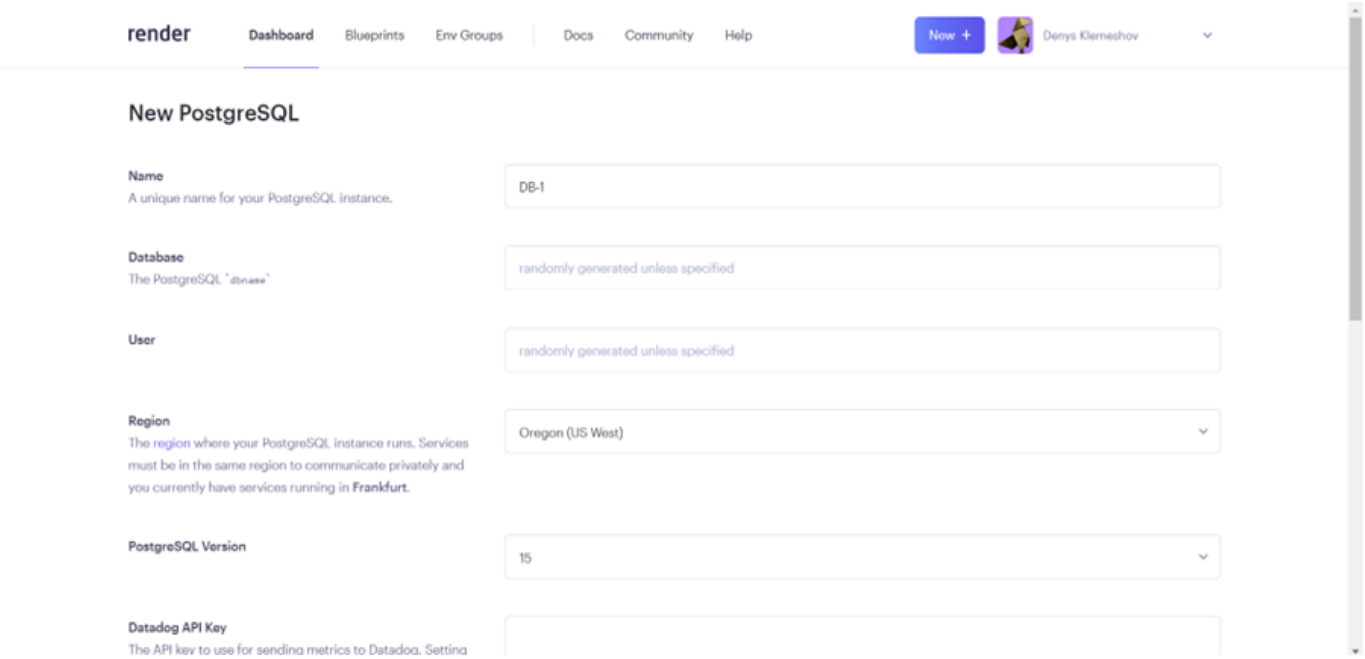

Рис. 3.26. Заповнення інформації про базу даних

Відправляю форму для створення бази даних (рис. 3.27).

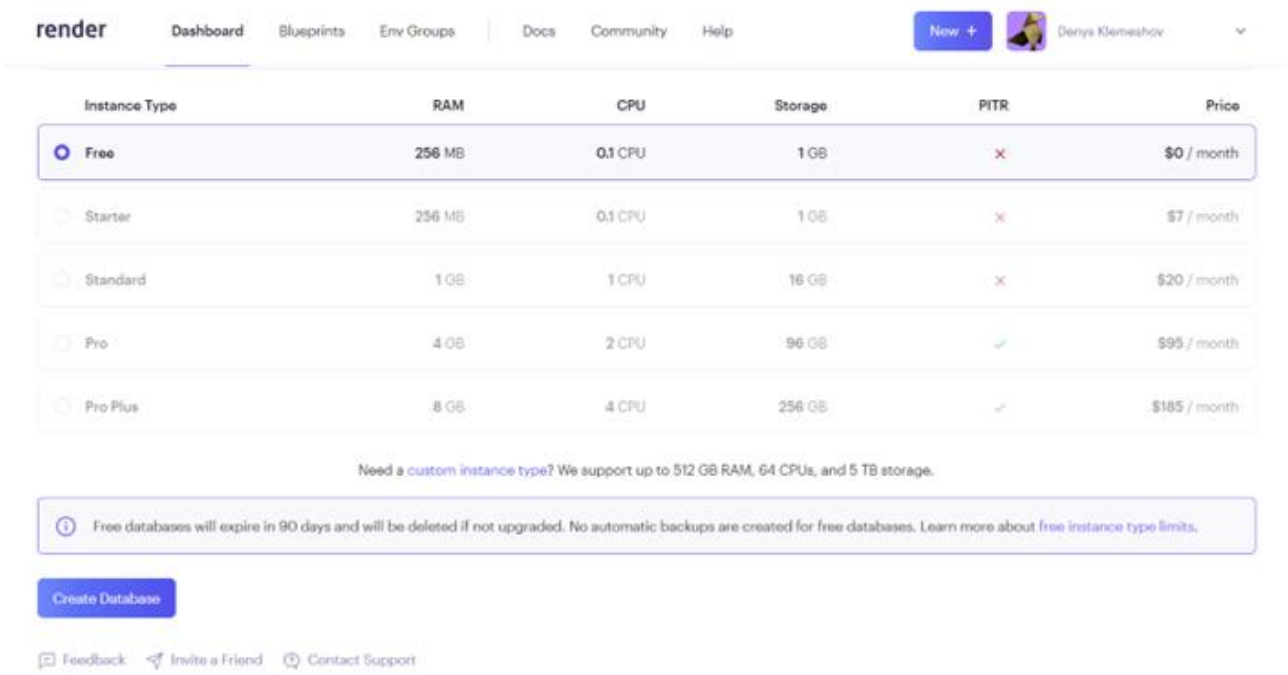

Рис. 3.27. Відправка форми для створення бази даних

Переглядаємо інформацію про створену базу даних (рис. 3.28).

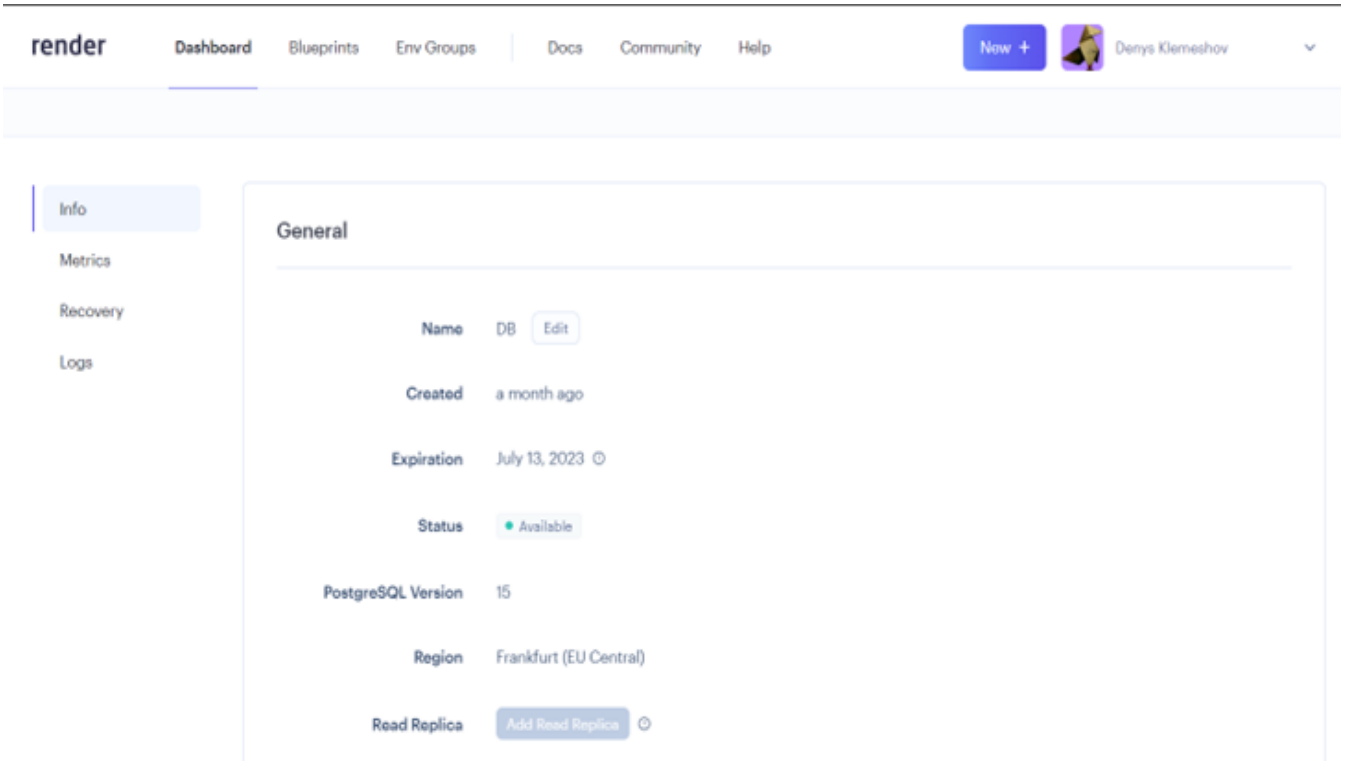

Рис. 3.28. Перегляд інформації про базу даних

Переходимо до секції Connection. Тут ми можем отримати дані для підключення бази даних до раніше створеного сервісу (рис. 3.29).

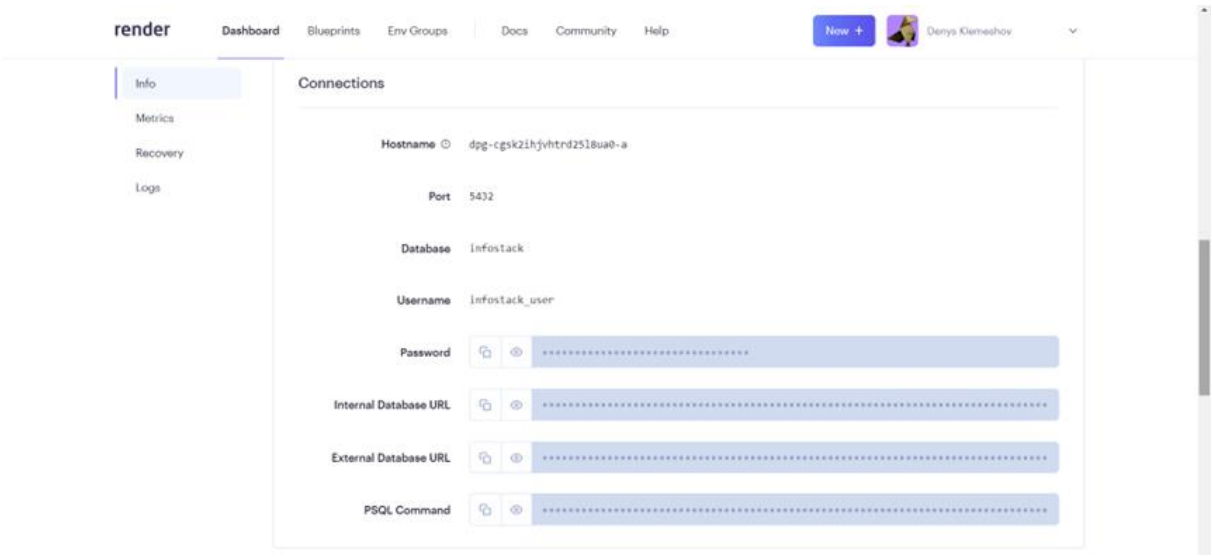

Рис. 3.29. Перегляд даних для підключення до бази даних

Нам також потрібно створити кластер для ElasticSearch. Для цього використовуємо сервіс Bonsai. Тут можна налаштувати хостинг для кластеру (рис. 3.30).

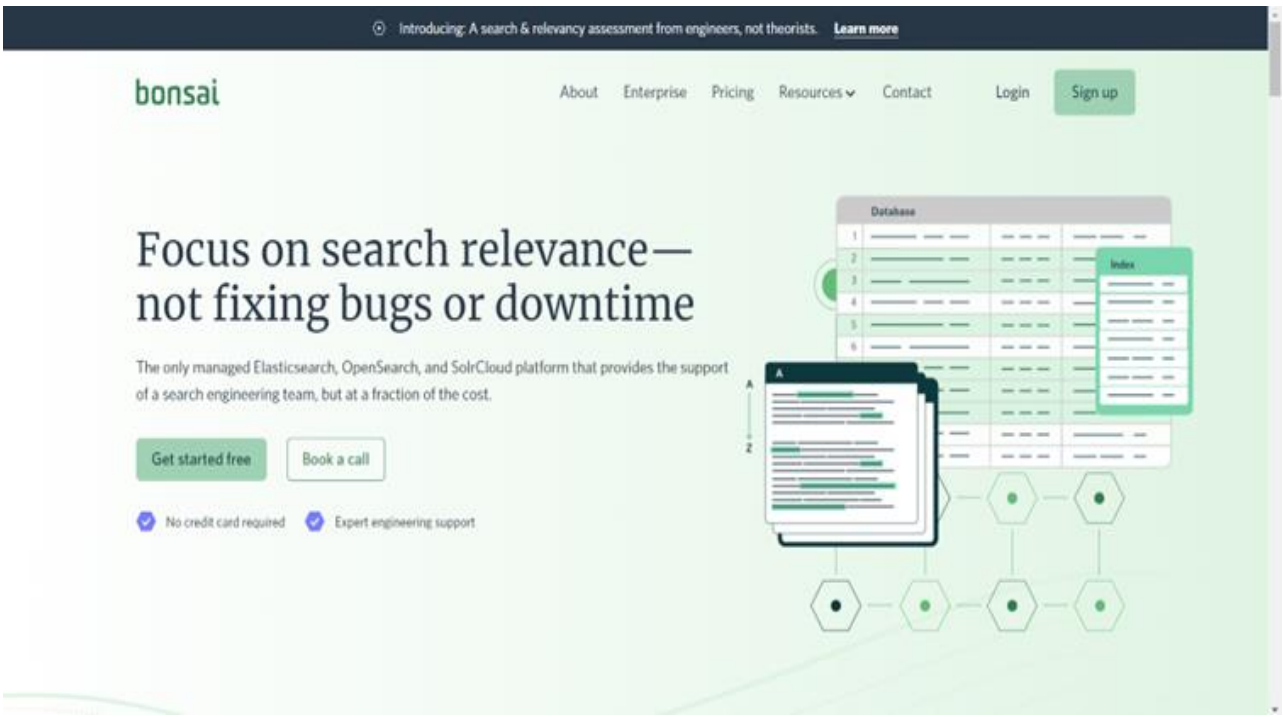

Рис. 3.30. Головна сторінка сервісу Bonsai

Створюємо кластер та переглядаємо інформацію про нього (рис. 3.31).

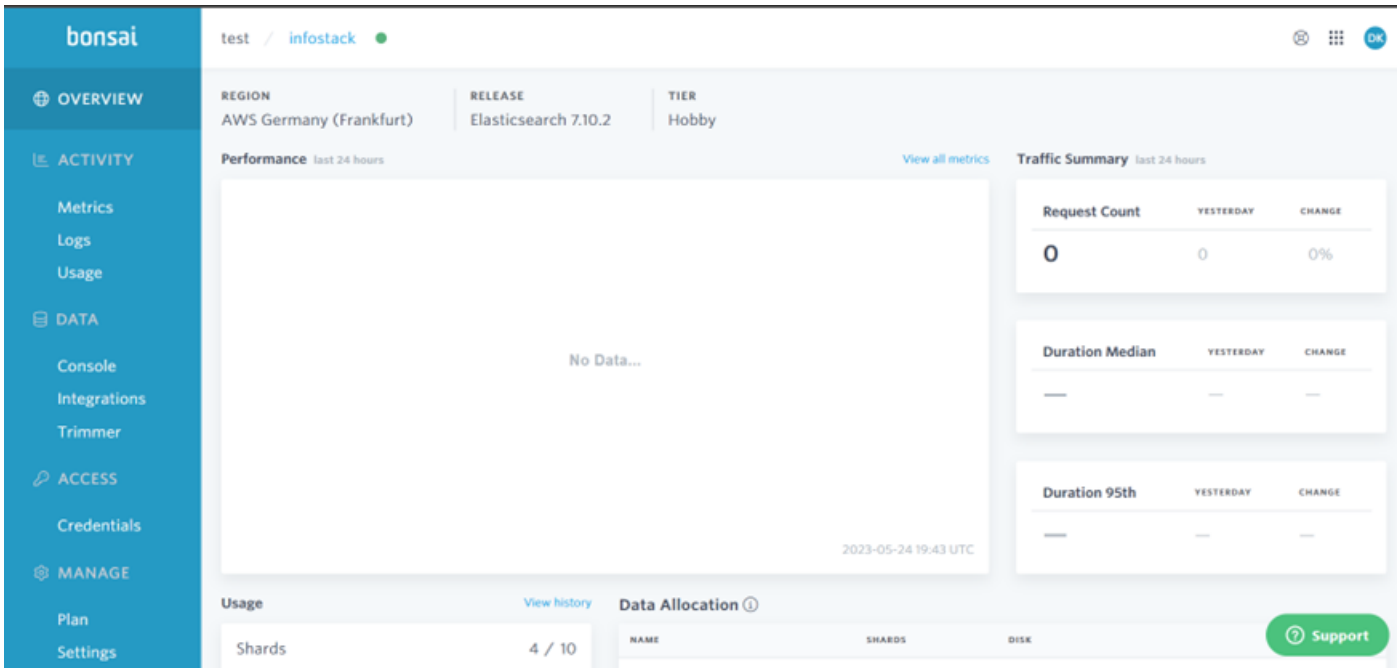

Рис. 3.31. Сторінка з інформацією про сервіс

Також збираємо дані для під'єднання веб сервісу до ElasticSearch (рис. 3.32).

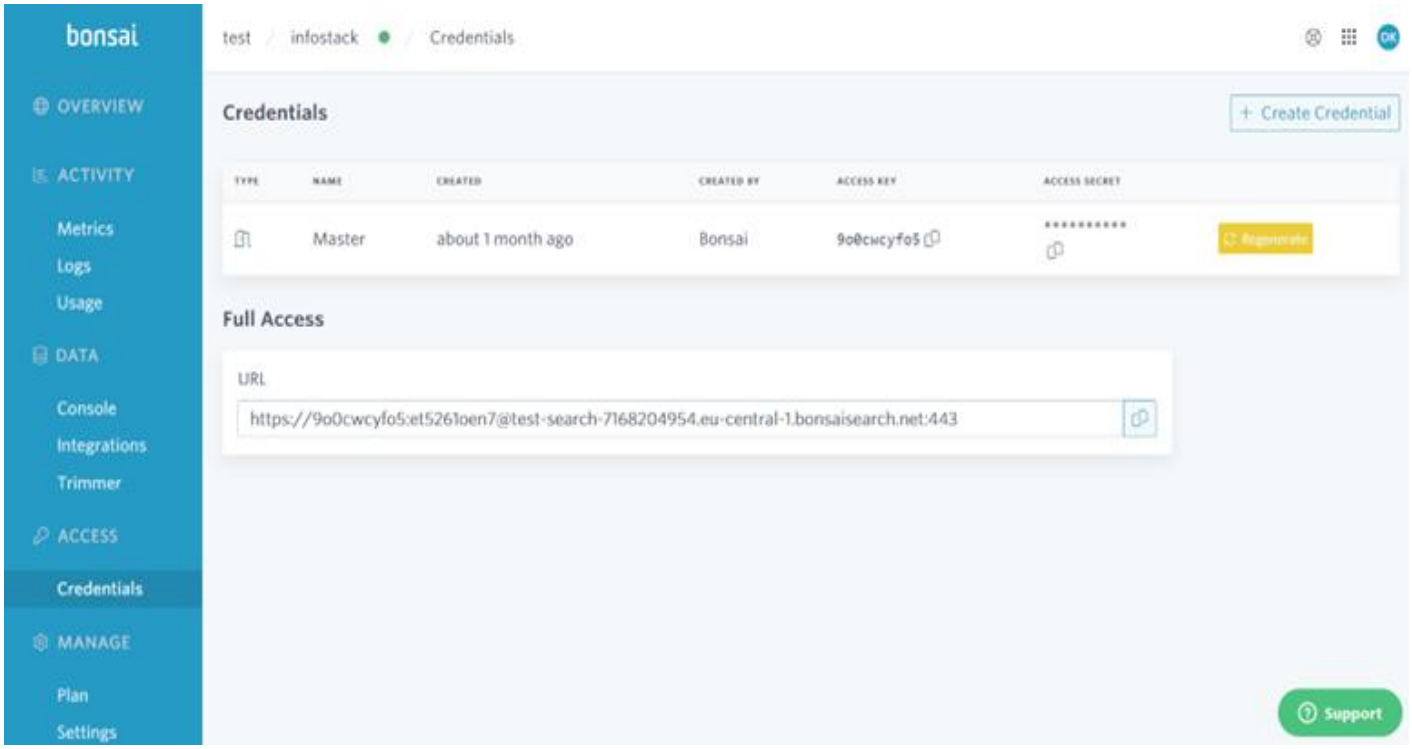

Рис. 3.32. Інформація для підключення до ElasticSearch

Знову повертаємося до сервісу, та у вкладці Environment вказуємо всі необхідні змінні оточення, а саме дані для підключення PostgreSQL, ElsaticSearch, GoogleAPI, GithubAP та іншу необхідну інформацію для коректної роботи сервісу (рис. 3.33).

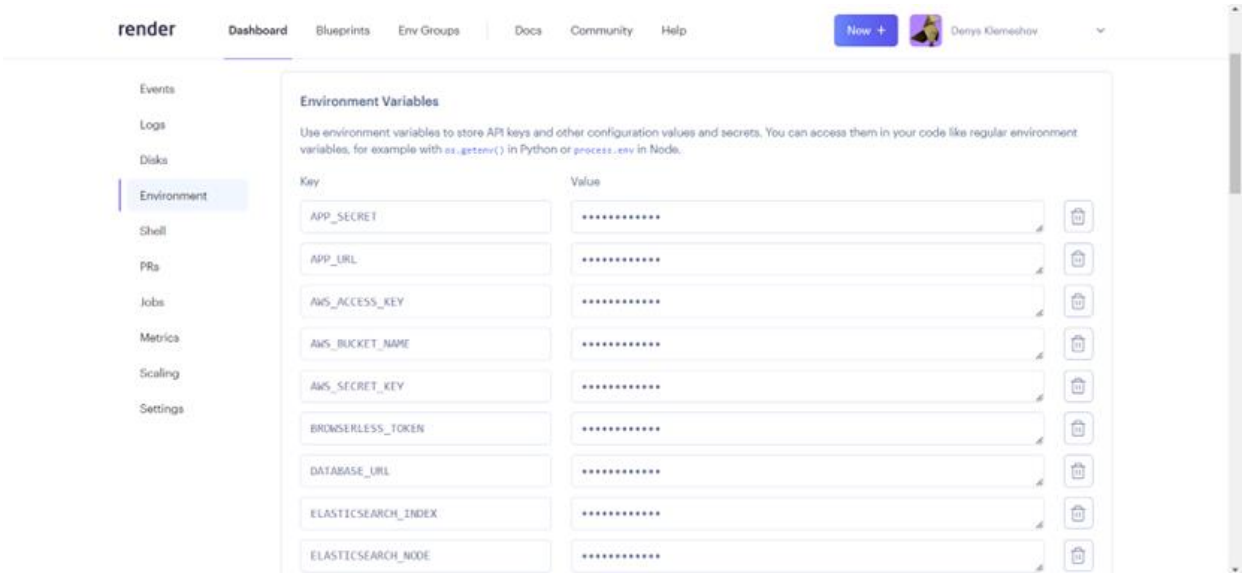

Рис. 3.33. Заповеннння змінних отоочення

Очкуємо, і бачимо що сервіс успішно розміщено на створеному сервері (рис. 3.34).

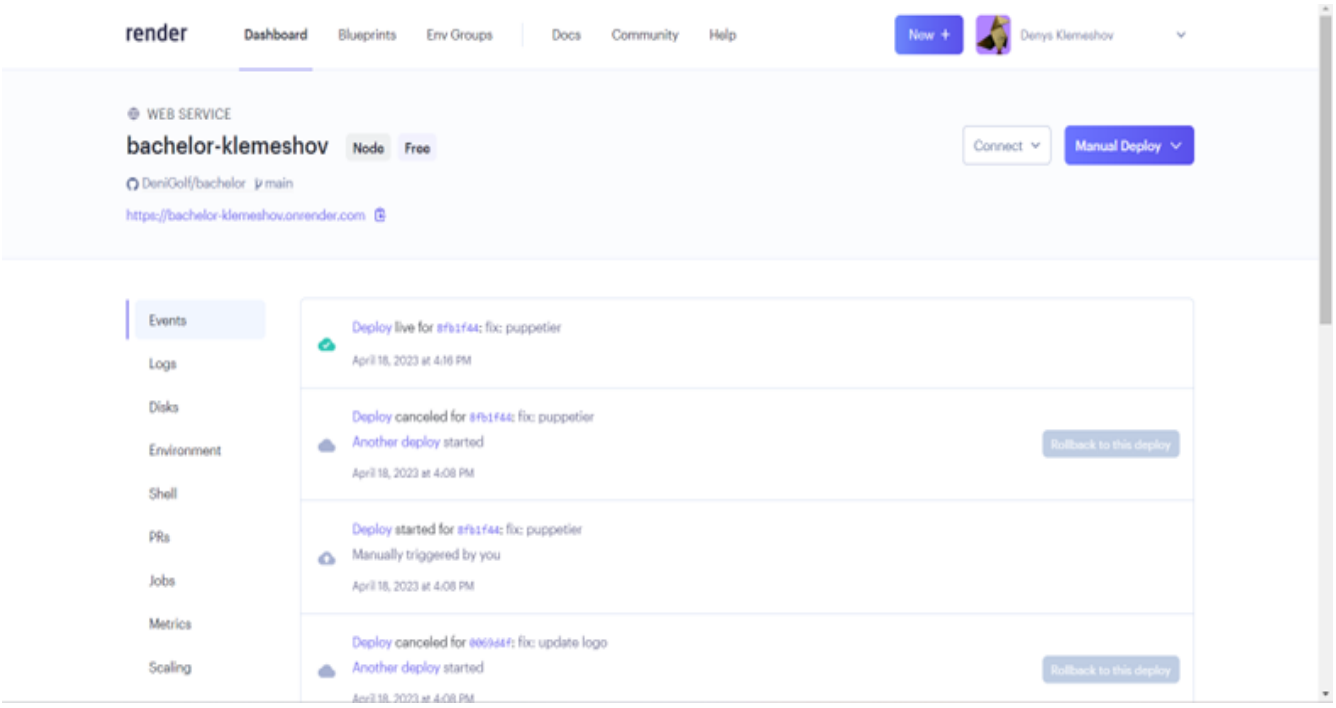

Рис. 3.34. Сторінка розгортань сервісу

3.4.4 Опис моделей

Моделі програми визначаються та описуються за допомогою бібліотеки TypeORM. TypeORM - це інструмент об'єктно-реляційного відображення (ORM), який полегшує взаємодію між реляційними базами даних та об'єктами додатків. Він пропонує практичний і зрозумілий спосіб пояснити організацію та зв'язки між різними компонентами даних системи.

Всі сутності наслідуються від абстрактого класу AbstractEntity (рис. 3.35).

```
export abstract class AbstractEntity extends BaseEntity {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;

  @DeleteDateColumn()
  deletedAt: Date;
```

Рис. 3.35. Модель AbstractEntity

Модель User (рис. 3.36).

```
@Entity()
export class User extends AbstractEntity {
  @Column({ length: 200 })
  fullName: string;
  @Column({ length: 50 })
  email: string;
  @Column({ length: 200, nullable: true })
  password: string;
  @Column({ nullable: true })
  avatar: string;
  @Column({ length: 200, nullable: true })
  title: string;
```

Рис. 3.36. Модель User

МодельWorkspace (рис. 3.37).

```
export class Workspace extends AbstractEntity {
  @Column({ length: 50 })
  name: string;

  @Column({ nullable: true })
  logo?: string;

  @OneToMany(() => UserWorkspace, (userWorkspace) => userWorkspace.workspace)
  userWorkspaces!: UserWorkspace[];

  @OneToMany(() => Team, (Team) => Team.workspace)
  teams: Team[];

  @OneToMany(() => Page, (Page) => Page.workspace)
  pages: Page[];

  @OneToMany(() => Notification, (Notification) => Notification.workspace)
  notifications: Notification[];

  @OneToMany(() => Tag, (Tag) => Tag.workspace)
  tags: Tag[];

  @OneToMany(() => Skill, (Skill) => Skill.workspace)
  skills: Skill[];
}
```

Рис. 3.37. Модель Workspace

Модель Page (рис. 3.38).

```
@Entity()
export class Page extends AbstractEntity {
  @RelationId((page: Page) => page.author)
  @Column()
  readonly authorId: string;

  @ManyToOne(() => User, (user) => user.pages)
  author: User;

  @RelationId((page: Page) => page.workspace)
  @Column()
  readonly workspaceId: string;

  @ManyToOne(() => Workspace, (workspace) => workspace.pages)
  workspace: Workspace;

  @RelationId((page: Page) => page.parentPage)
  @Column({ nullable: true })
  readonly parentPageId: string;

  @ManyToOne(() => Page, (page) => page.childPages, {
    onDelete: 'CASCADE',
  })
  parentPage: Page;

  @OneToMany(() => Page, (page) => page.parentPage)
  childPages: Page[];
```

Рис. 3.38. Модель Page

Модель Comment (рис. 3.39).

```
@Entity()
@Tree('closure-table')
export class Comment extends AbstractEntity {
  @Column()
  text: string;

  @Column({ nullable: true })
  voiceRecord: string;

  @RelationId((comment: Comment) => comment.author)
  @Column()
  readonly authorId: string;

  @ManyToOne(() => User, (user) => user.comments)
  author: User;

  @RelationId((comment: Comment) => comment.page)
  @Column()
  readonly pageId: string;

  @ManyToOne(() => Page, (page) => page.comments, {
    onDelete: 'CASCADE',
  })
  page: Page;

  @RelationId((comment: Comment) => comment.parentComment)
  @Column({ nullable: true })
  readonly parentCommentId: string;

  @TreeChildren()
  childComments: Comment[];

  @TreeParent({ onDelete: 'CASCADE' })
  parentComment: Comment;

  @OneToMany(() => Reaction, (reaction) => reaction.comment)
  reactions: Reaction[];
}
```

Рис. 3.39. Модель Comment

Модель Reaction (рис. 3.40).

```
@Entity()
export class Reaction extends AbstractEntity {
  @Column({ length: 50 })
  reaction: string;

  @RelationId((reaction: Reaction) => reaction.user)
  @Column()
  readonly userId: string;

  @ManyToOne(() => User, (user) => user.reactions)
  user: User;

  @RelationId((reaction: Reaction) => reaction.comment)
  @Column()
  readonly commentId: string;

  @ManyToOne(() => Comment, (comment) => comment.reactions, {
    onDelete: 'CASCADE',
  })
  comment: Comment;
}
```

Рис. 3.40. Модель Reaction

Модель Tag (рис. 3.41).

```
@Entity()
export class Tag extends AbstractEntity {
  @RelationId((tag: Tag) => tag.workspace)
  @Column()
  readonly workspaceId: string;

  @ManyToOne(() => Workspace, (workspace) => workspace.tags)
  workspace: Workspace;

  @Column()
  name: string;

  @ManyToMany(() => Page, (page) => page.tags)
  pages: Page[];

  @Column({
    type: 'enum',
    enum: TagType,
    default: TagType.APP,
  })
  type: TagType;
}
```

Рис. 3.41. Модель Tag

Цей словник даних описує основні компоненти даних, що використовуються в системі. Він пропонує детальний опис характеристик первинних сутностей, типів даних та їх зв'язків. Окремі сторінки додатків представлені об'єктом Page, тоді як користувачі системи представлені сутністю User. Користувачі можуть відповідати на коментарі на сторінках різними способами. До певних сторінок можна прив'язувати теги. Для того, щоб класифікувати та впорядкувати сторінки, існують зв'язки між користувачами, сторінками та тегами, які визначають права та взаємозв'язки.

3.5 Тестування розробленої системи на основі тестових даних та аналіз отриманих результатів

Перейдемо до показу користувацького інтерфейсу системи.

Спочатку розглянемо сторінку авторизації користувача (рис. 3.42). Тут видно поля для введення логіна та пароля користувача. Крім того, є кнопки для авторизації сторонніх сервісів.

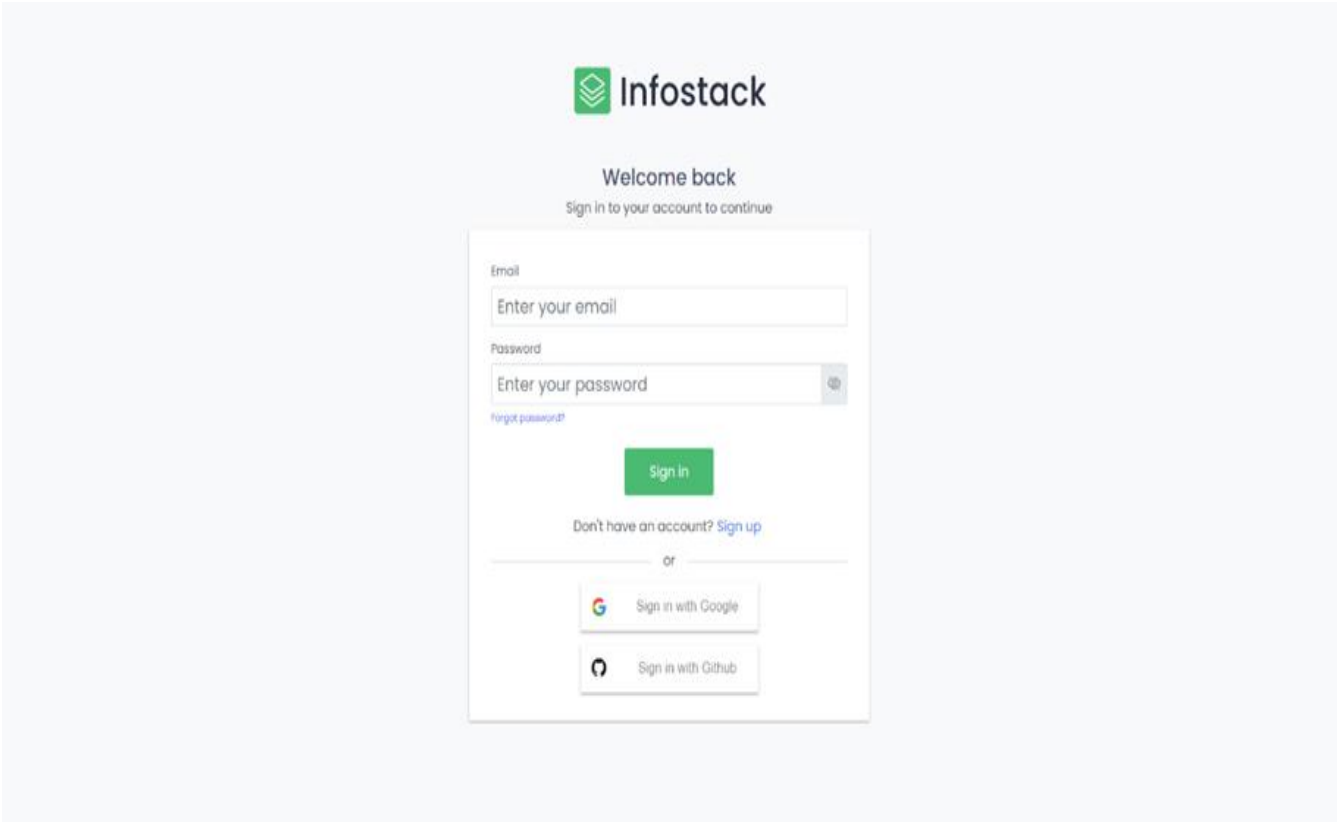

Рис. 3.42. Сторінка логіну

Також на цій сторінці пристутня кнопка для реєстрації. При кліку ми потрапляємо на сторінку де користувач може зареєструватись у системі (рис. 3.43).

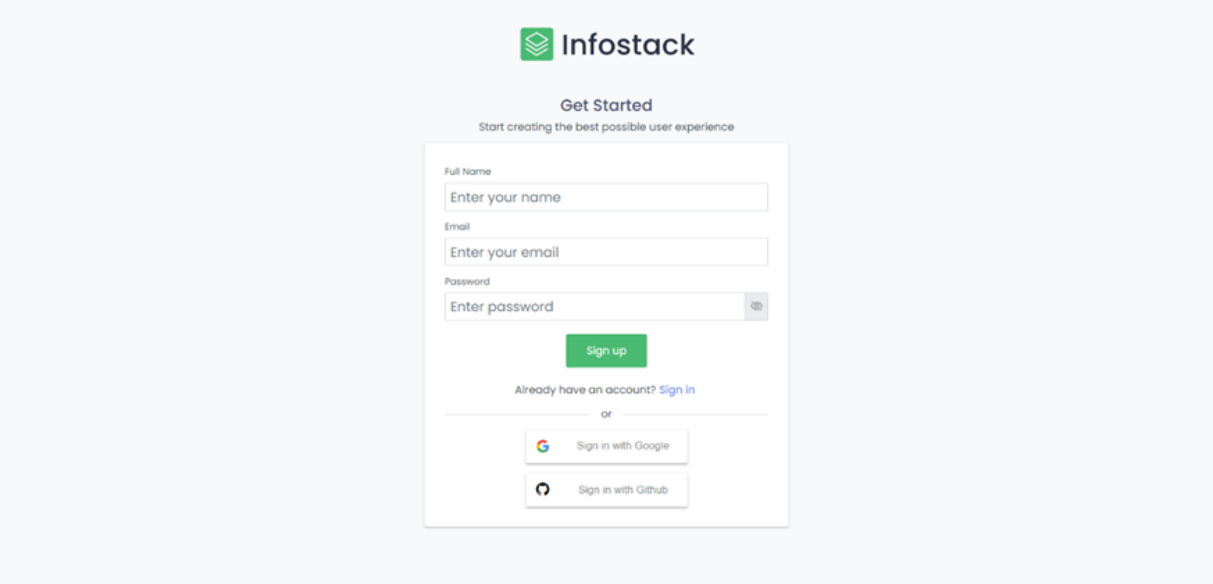

Рис. 3.43. Сторінка реєстрації

Після реєстрації, або авторизації користувач потрапляє на сторінку вибору робочого простору. Тут можна, або перейти у вже існуючий простір, або створити новий (рис. 3.44)

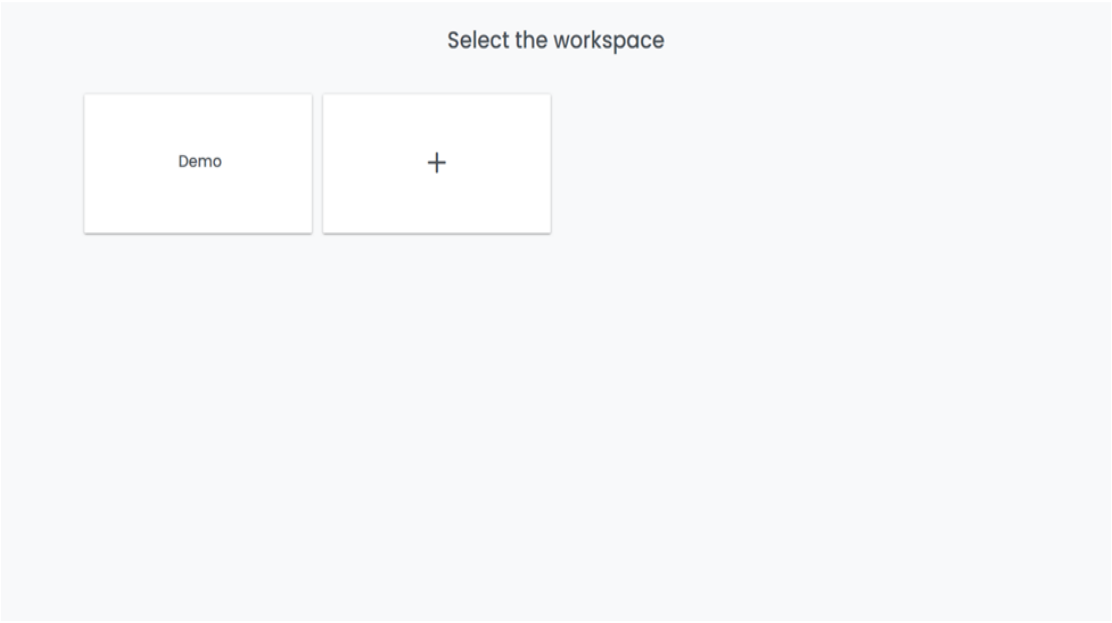

Рис. 3.44. Сторінка вибору простору

Після вибору робочої області з'являється головний екран програми (рис. 3.45). Ви можете вибрати одну з існуючих сторінок або створити нову за допомогою меню зліва. Сповіщення, рядок пошуку та інформація про користувача розташовані в шапці. Статистика представлена в основній частині, з графіками, що показують останні оновлення сторінок, сторінки з найбільшою кількістю переглядів за тиждень і сторінки, які найчастіше змінюються. Нижче наведено графік оновлення сторінок по днях.

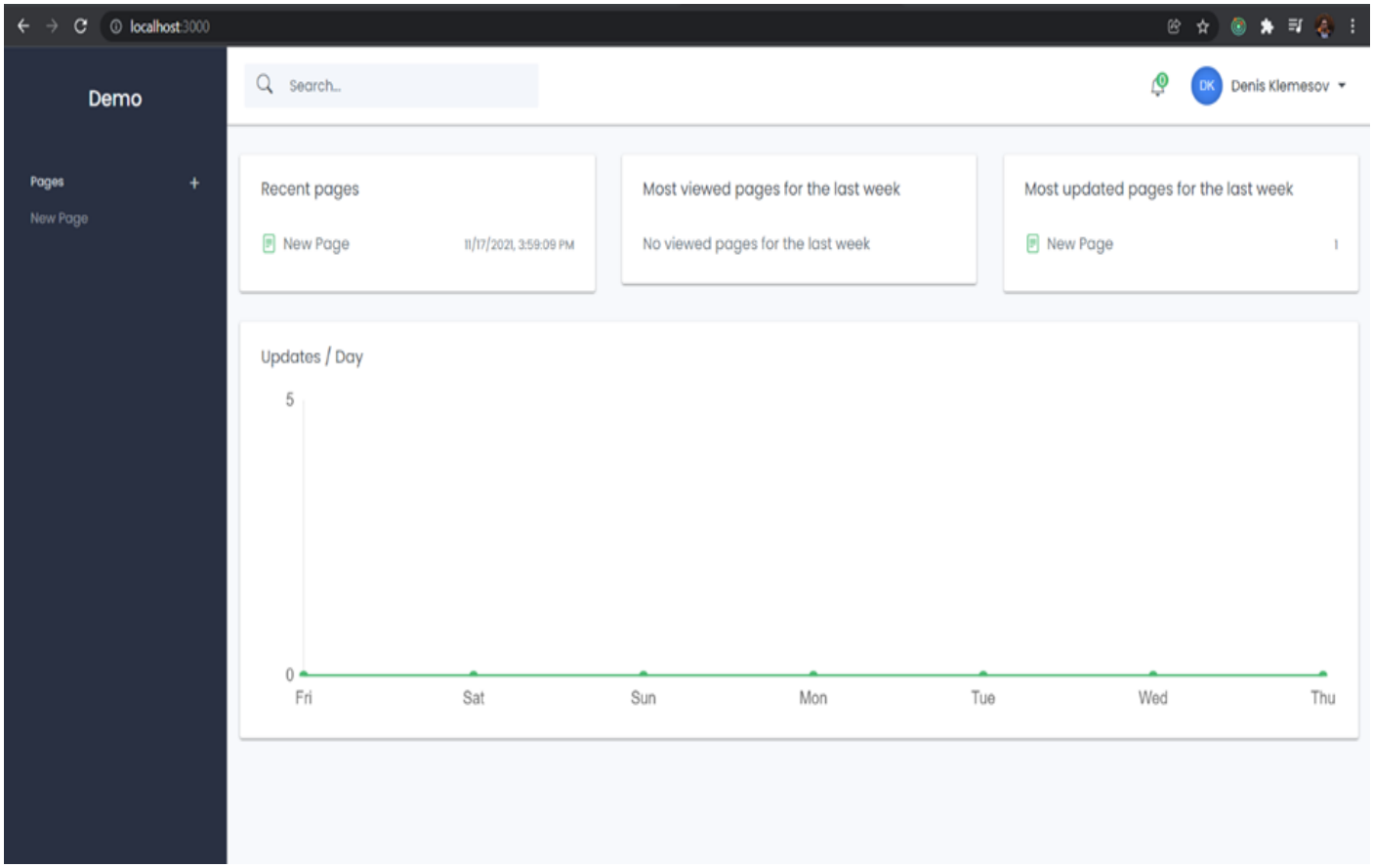

Рис. 3.45. Головна сторінка

На сторінці профілю користувача присутня вся інформація про почточного користувача (рис. 3.46). Можна побачити навички користувача, сторінки на які він підписаний.

В секції активностей присутні дані про оновлення сторінок. Також можна показувати всі активності, або тільки створені поточним користувачем.

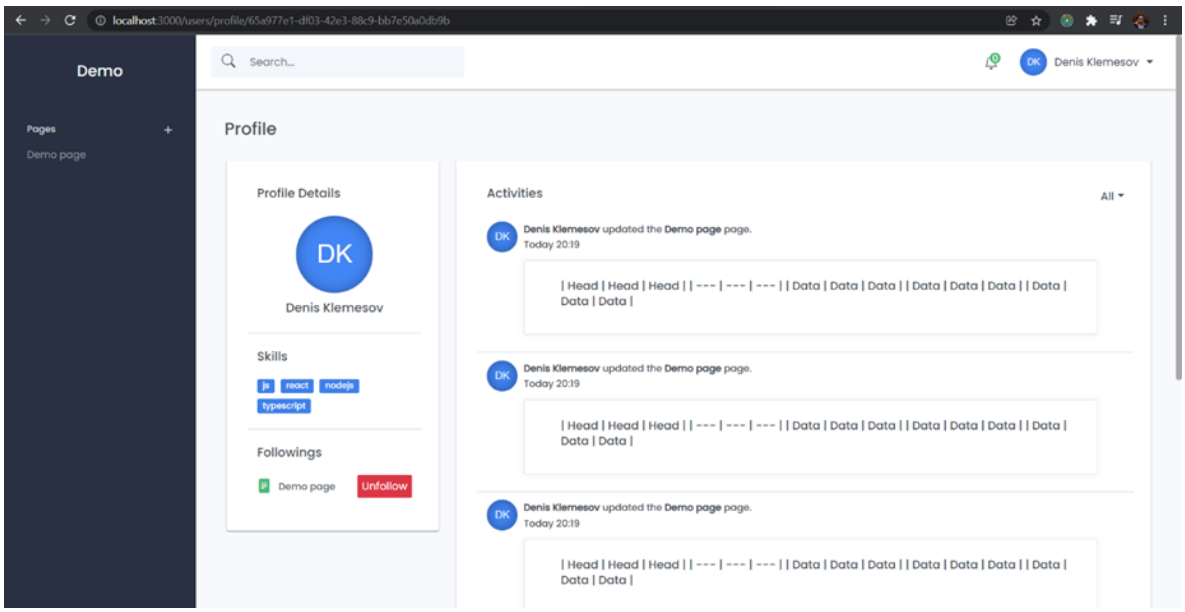

Рис. 3.46. Сторінка профілю користувача

Редагування особистих даних, таких як ім'я, посада та навички, є частиною налаштувань користувача (рис. 3.47). Зліва є розділ для завантаження аватара. Якщо користувач не додасть аватарку, замість неї з'являться його ініціали. В інтерфейсі є модальне вікно для завантаження зображення, поля для введення даних і кнопка для збереження змін.

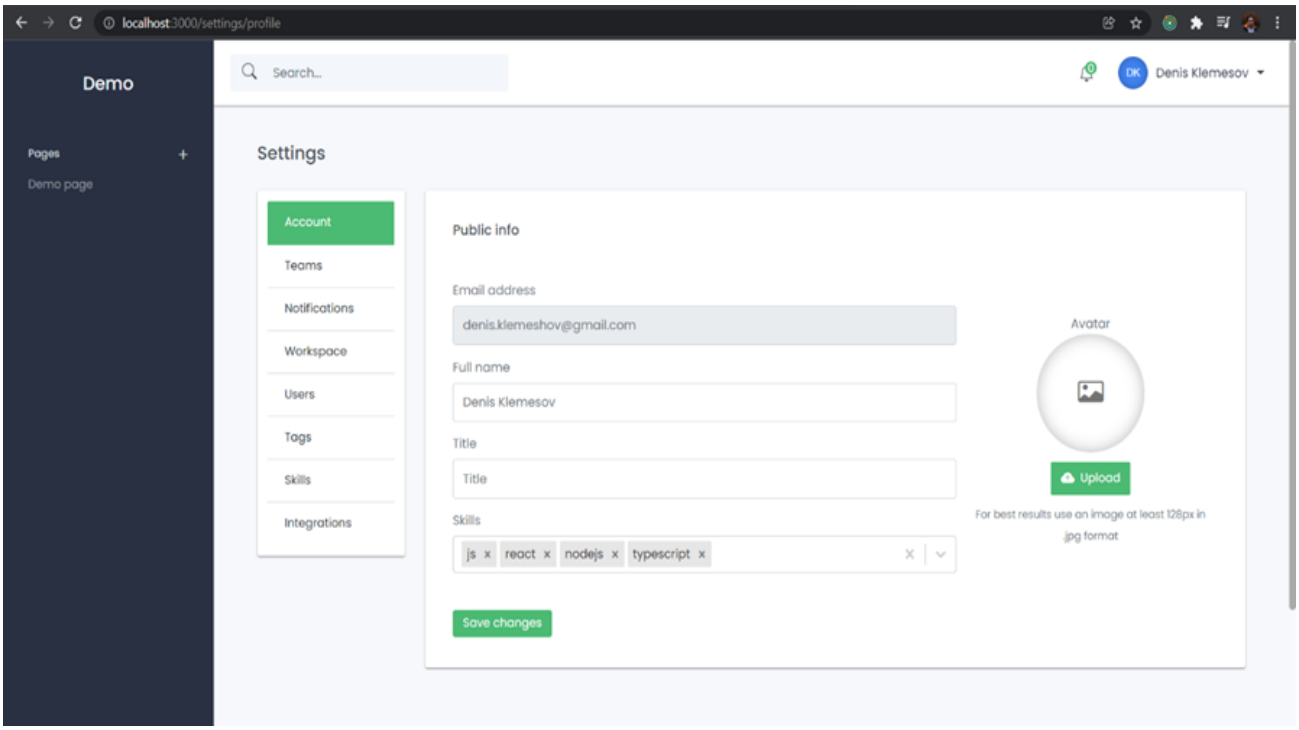

Рис. 3.47. Сторінка налаштувань акаунта

Ви можете переглянути вміст сторінки, вибравши її з меню ліворуч (рис. 3.48). На ній міститься вся необхідна користувачеві інформація. Ви можете переглянути авторів, підписників, теги та розділи в розділах ліворуч. Вміст сторінки, кнопка вибору версії та всі налаштування показані праворуч. Нижче знаходиться розділ коментарів.

Можна залишати текстові та голосові коментарі. Також присутня можливість додавання реакцій для швидкого висловлення думки.

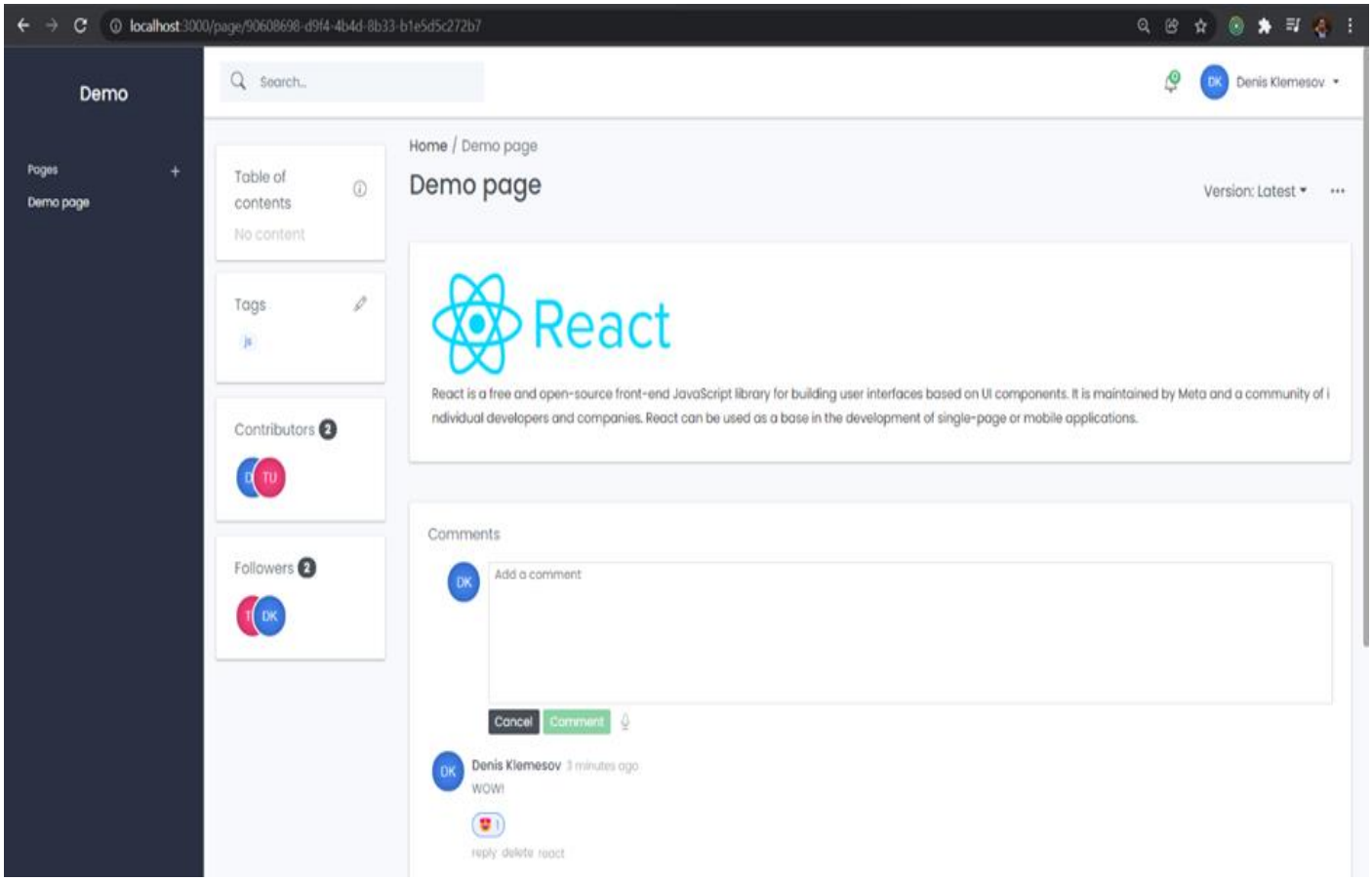

Рис. 3.48. Сторінка для перегляду вмісту сторінки

На сторінці для редагування вмісту сторінки відображається назва сторінки, редактор та кнопки керування (рис. 3.49).

Використовується редактор який працює за допомогою особливого синтаксису. Користувачі можуть легко форматувати текст, вставляти цитати, зображення, або таблиці.

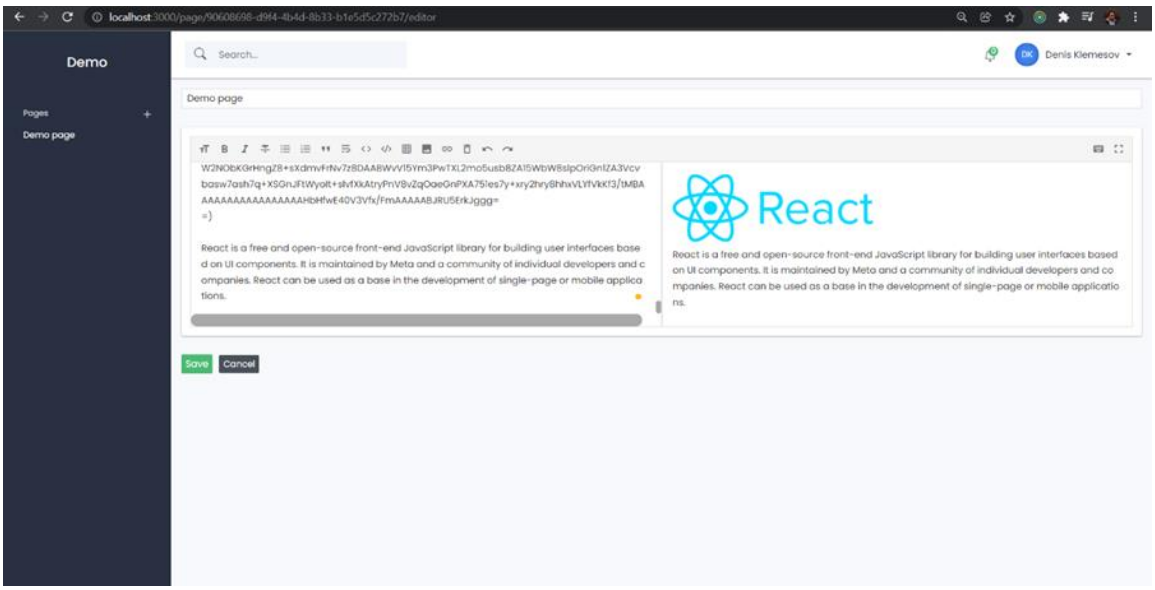

Рис. 3.49. Сторінка для редагування вмісту сторінки

Система пропонує універсальну систему керування сповіщеннями (рис. 3.50). Ви самі вирішуєте, які сповіщення ви хочете отримувати. Сповіщення бувають двох типів. Сповіщення електронною поштою надходять на електронну адресу користувача, тоді як системні сповіщення відображаються в заголовку сторінки. Ви можете вибрати зручні для вас налаштування отримання сповіщень за допомогою опцій.

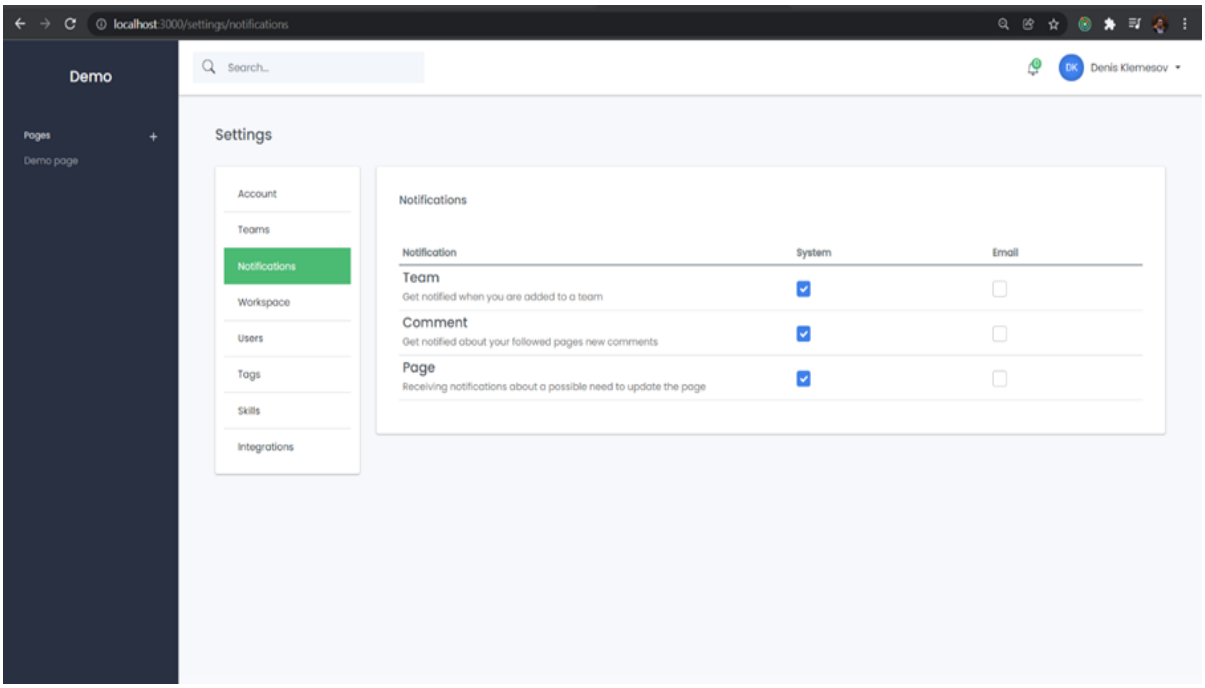

Рис. 3.50. Сторінка налаштувань сповіщень

Сповіщення в реальному часі користувач може переглянути в розділі Сповіщення (рис. 3.51). Сповіщення бувають різних типів, кожен з яких має власну іконку та інформацію. Таким чином, користувач може швидко визначити, що саме відбулося в системі. В інтерфейсі є кнопки для попереднього перегляду кожної сторінки, кнопка для позначення всіх повідомлень як прочитаних, а також інформація про кількість сторінок. Також видно час з моменту отримання повідомлення.

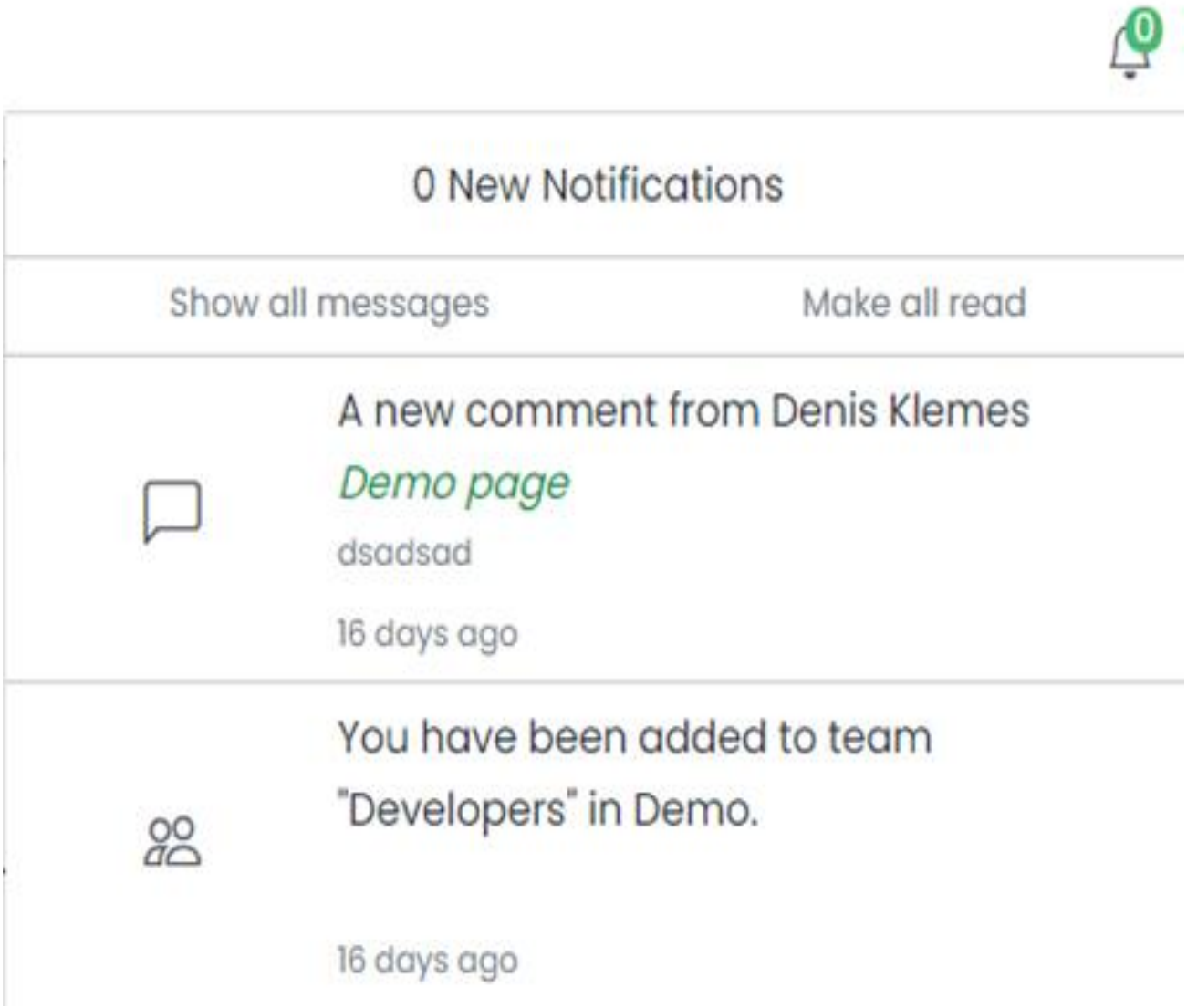

Рис. 3.51. Блок сповіщень

При кожній зміні сторінки зберігається також старіша версія її вмісту (рис. 3.52). Користувач може з легкістю повернутись до попередніх змін за необхідності.

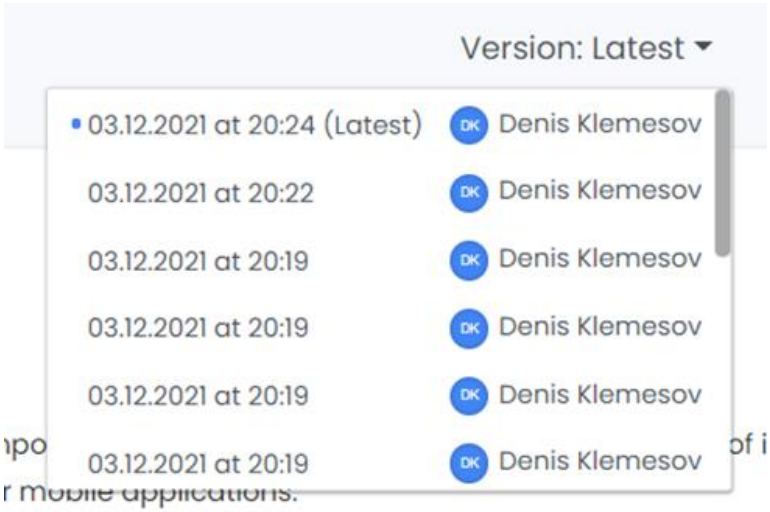

Рис. 3.52. Блок вибору версії сторінки

Вам може знадобитися поділитися вмістом сторінки з іншими людьми. Крім того, бувають випадки, коли вам потрібно надати інформацію комусь, хто не є користувачем системи. Ви можете створити посилання на певний період часу за допомогою сторінки створення посилання (рис. 3.53). Людина може скористатися ним, щоб переглянути вміст сторінки за певний період часу. Ви можете надіслати запрошення на електронну пошту. За бажанням ви можете продовжити термін дії посилання.

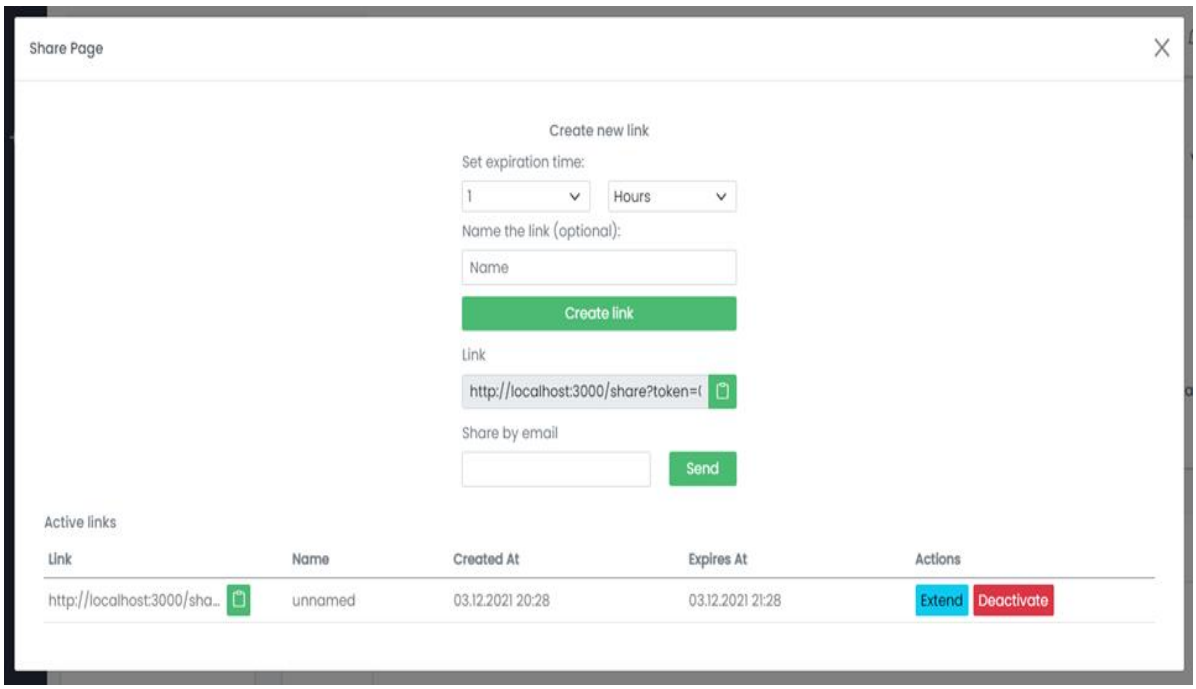

Рис. 3.53. Сторінка для створення посилання

Щоб упорядкувати користувачів, потрібні налаштування команди (рис. 3.54). Ви можете легко керувати правами доступу, якщо об'єднаєте кількох користувачів у команду. На цій сторінці ви можете змінити існуючу команду або створити нову. Інтерфейс містить кнопки для керування командами, аватарки користувачів і назву команди.

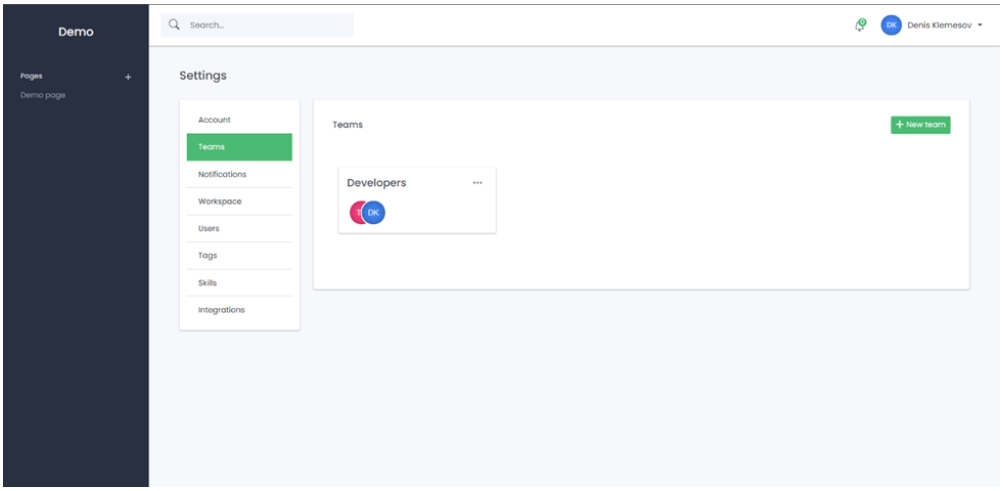

Рис. 3.54. Сторінка налаштування команд

Різні сторінки класифікуються за допомогою тегів (рис. 3.55). Ви можете створити скільки завгодно тегів. Список тегів і кнопки для оновлення або додавання нових тегів включено до інтерфейсу користувача.

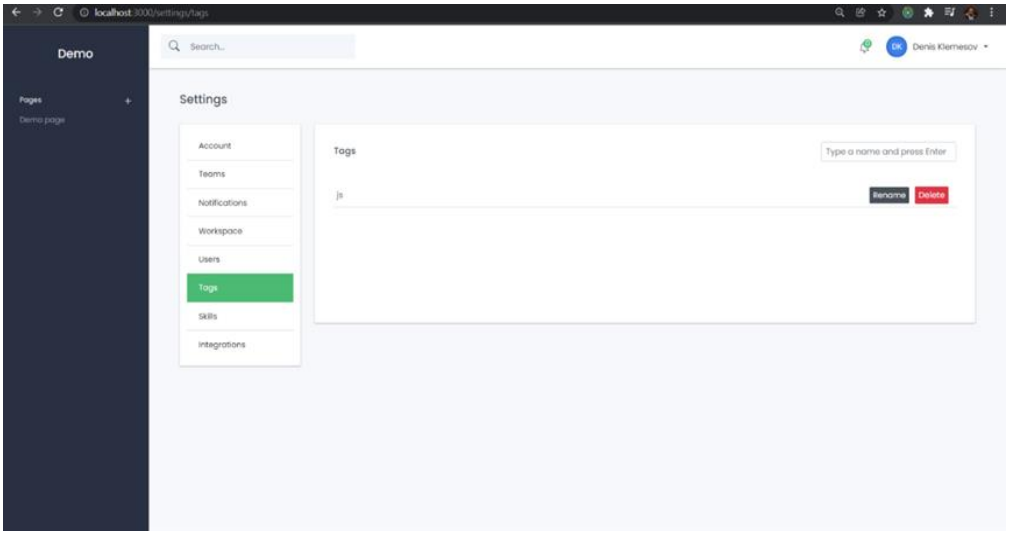

Рис. 3.55. Сторінка налаштувань тегів

Якщо користувач вкаже не існуючу адресу сторінки його буде переаправлену на сторінку 404 (рис. 3.56).

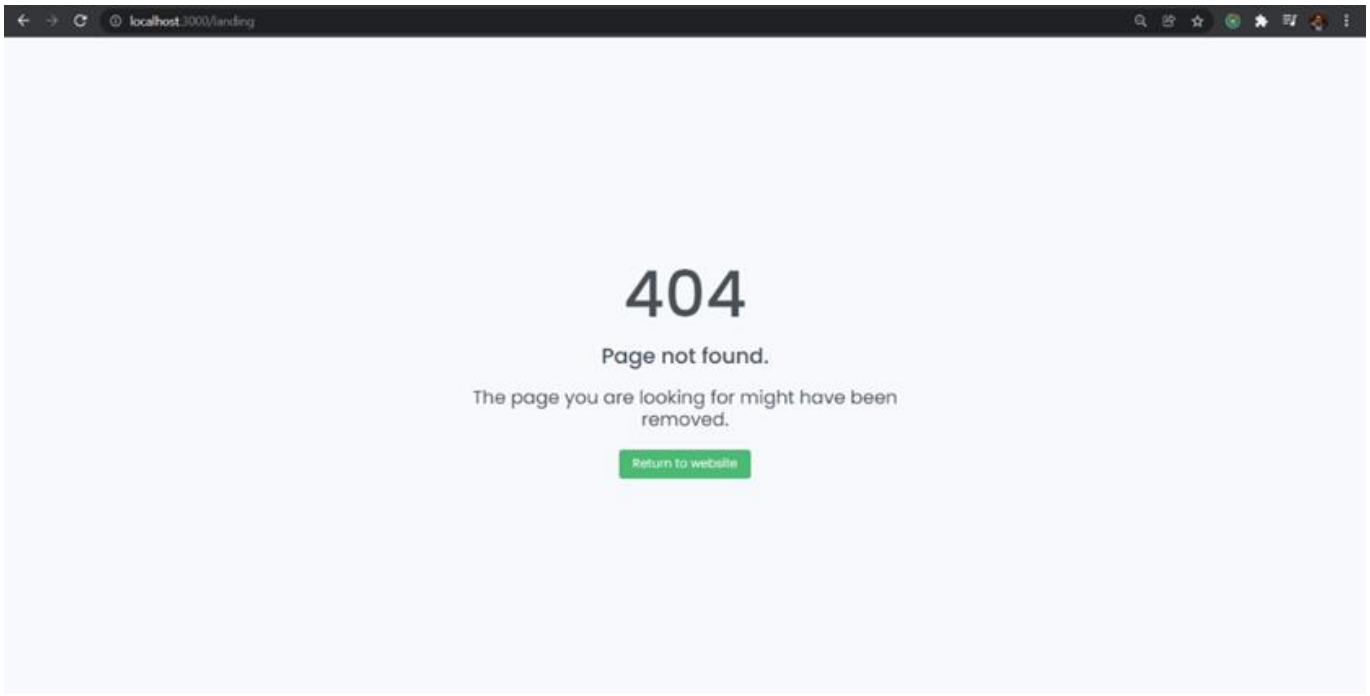

Рис. 3.56. Сторінка помилки

Користувач може побачити причину помилки, та повернутись на головну сторінку.

3.6 Висновки до розділу

У цьому розділі я висвітлив основні проблеми системи упрівління та організації інформації. Проаналізував вплив управління інформацією на організаційну ефективність та прийняття рішень. Споєктував та розробкив архітектури централізованої системи управління інформацією. Створив Canvas модель системи, розробив UML діаграми реєстрації та авторизації в системі. Описав процес створення клієнтської, серверної частин та розгортання системи на сервері. Продемонстрував та протестував роботу основних функцій системи. Крім того розроблено схему бази даних. У цій частині ми розглянули кроки, пов'язані з проектуванням та створенням інтерфейсу користувача. Ми обговорили фреймворки, прототипи та тестування

інтерфейсу з точки зору цінності дизайну, орієнтованого на користувача, що передбачає розуміння його потреб.

Інтерфейс використовує адаптивний дизайн, щоб забезпечити найкращий досвід перегляду та взаємодії на різних пристроях і розмірах екранів. Фреймворк Bootstrap пропонує міцну основу для створення надійних та естетично привабливих компонентів інтерфейсу.

Також наведено основні фрагменти коду який відповідає за функціонування ключових особливостей системи. Продемонстровано роботу описаних функцій які вирішують проблеми зберігання та організації інформації.

Алгоритм збереження інформації охоплює багатоетапний процес, де кожен компонент (React, Node.js, PostgreSQL, Elasticsearch) відіграє ключову роль у забезпеченні ефективного зберігання, пошуку та захисту даних. Завдяки шифруванню, індексації та управлінню ролями система є надійною, безпечною та продуктивною.

Організація даних у системі поєднує ефективне збереження в PostgreSQL та швидкий пошук у Elasticsearch, з урахуванням шифрування, валідації даних та кешування. Таке рішення забезпечує високу продуктивність, гнучкість та надійність системи, а також дозволяє користувачам швидко знаходити необхідну інформацію.

ВИСНОВКИ

У цій магістерській роботі було проведено ретельне дослідження галузі систем зберігання та організації інформації, створення теоретичних моделей та алгоритмів, а також планування та реалізація централізованої системи управління інформацією. Дослідження показало, наскільки гостро сучасні підприємства потребують ефективного та зручного рішення для вирішення проблем фрагментарного зберігання даних, відсутності можливостей для спільної роботи та підтримки актуальності документації. Завдяки додаванню нових функцій і методів, запропоноване рішення заповнило важливі прогалини в існуючих системах.

В теоретичну частині описав сучасні методи та алгоритмів організації, пошуку та захисту даних. Після ретельного аналізу існуючих моделей та фреймворків з метою визначення їхніх переваг та недоліків було розроблено індивідуальну стратегію для запропонованої системи. Особливу увагу було приділено гарантуванню конфіденційності, узгодженості та цілісності даних при збереженні зручності використання та масштабованості. Було досліджено, як штучний інтелект може бути інтегрований в управління інформацією, щоб підвищити продуктивність системи і запропонувати більш інтелектуальні рішення для автоматизації та аналізу даних.

В рамках практичної частини дослідження було спроектовано та розроблено повністю функціональний прототип системи. Ця процедура включала розробку надійної архітектури на основі сучасних ідей, впровадження алгоритмів для ефективної обробки даних та проведення ретельного тестування для оцінки продуктивності системи. Розроблене рішення продемонструвало значні переваги над існуючими системами, включаючи кращі інструменти для моніторингу та збереження актуальності збережених даних, централізований метод зберігання даних та розширені можливості для спільної роботи.

Наукова унікальність цієї роботи полягає в особливому поєднанні передових алгоритмів, моделей безпеки та інструментів на основі штучного інтелекту, які використовуються для вирішення реальних проблем управління інформацією.

Запропонований підхід не тільки задовольняє потреби компанії зараз, але й закладає основу для розширення та майбутніх удосконалень.

Виходячи з досліджених концепцій та алгоритмів, можна зробити висновок, що сучасні системи управління інформацією мають бути побудовані на поєднанні традиційних підходів (як реляційні бази даних) з новітніми технологіями, зокрема, штучним інтелектом і алгоритмами безпеки. Це забезпечить не лише вирішення поточних задач, але й дозволить системі адаптуватися до нових, мінливих вимог і забезпечити її стабільність, безпеку та високу ефективність на довготривалий період. Тому, наступними етапами в розробці цієї системи будуть інтеграція цих алгоритмів у практичну реалізацію, тестування їх функціональності та постійне вдосконалення на основі реальних відгуків користувачів і випадків застосування.

Алгоритм зберігання даних в системі включає кілька етапів, де кожен компонент, такий як React, Node.js, PostgreSQL та Elasticsearch, грає важливу роль у забезпеченні ефективного збереження, обробки та захисту інформації. Використання шифрування, індексації та управління доступом гарантує безпеку, надійність і високу продуктивність системи.

Структура даних у системі оптимізує використання PostgreSQL для надійного зберігання та Elasticsearch для швидкого пошуку, з урахуванням шифрування, перевірки коректності даних та кешування. Це рішення забезпечує відмінну продуктивність, гнучкість і стабільність системи, дозволяючи користувачам швидко отримувати доступ до потрібної інформації.

Таким чином, пропонуючи свіжий підхід, який забезпечує баланс між теоретичною строгістю і реальним застосуванням, ця магістерська робота значно просуває дисципліну інформаційного менеджменту. Результати показують, як система може підвищити організаційну ефективність, прискорити співпрацю та полегшити прийняття рішень. Крім того, робота пропонує міцну основу для подальшого вивчення та розвитку в цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Elmasri, R.; Navathe, S. B. Fundamentals of Database Systems. 7th Edition. Pearson. – 2016. – С. 110-275.
2. Date, C. J. An Introduction to Database Systems. 8th Edition. Addison-Wesley. – 2005. – С. 152-404.
3. Baeza-Yates, R.; Ribeiro-Neto, B. Modern Information Retrieval: The Concepts and Technology Behind Search. 2nd Edition. Addison-Wesley. – 2011. – С. 64-99.
4. Tanenbaum, A. S.; van Steen, M. Distributed Systems: Principles and Paradigms. 2nd Edition. Pearson. – 2007. – С. 91-350.
5. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to Algorithms. 3rd ed., MIT Press. – 2009. – С. 168-650.
6. "PostgreSQL Documentation". PostgreSQL.org.
7. Fielding, R. T.; Taylor, R. N. "Principled Design of the Modern Web Architecture". ACM Transactions on Internet Technology. – 2000. – С. 42-58.
8. Langville, A. N., & Meyer, C. D. Google’s PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press. – 2006. – С. 110-450.
9. "React Documentation". Reactjs.org.
10. "Next.js Documentation". Nextjs.org.
11. "React Flow Documentation". Reactflow.dev.
12. "Material-UI Documentation". Mui.com.
13. "CSS-Tricks: A Complete Guide to Flexbox". Css-tricks.com.
14. Crockford, D. JavaScript: The Good Parts. O'Reilly Media. – 2008. – С. 10-128.
15. Osmani, A. Learning JavaScript Design Patterns: A JavaScript and TypeScript Developer's Guide. 2nd Edition. O'Reilly Media. – 2021. – С. 58-290.
16. Keith, J. HTML5 for Web Designers. A Book Apart. – 2009. – С. 23-195.
17. "Web Accessibility Guidelines (WCAG)". W3C. Available online: https://www.w3.org/WAI/.
18. "Web Performance Optimization Guide". Google Developers.
19. "Frontend Masters: Advanced React Patterns". Frontendmasters.com.

20. Krug, S. Don't Make Me Think: A Common Sense Approach to Web Usability. 2nd Edition. New Riders. – 2006. – С. 100-237.
21. "Node.js Documentation". Nodejs.org. "Express.js Documentation". Expressjs.com.
22. "REST API Design Guidelines". Restfulapi.net.
23. "MongoDB Documentation". MongoDB.com.
24. "Redis Documentation". Redis.io.
25. "ElasticSearch Documentation". Elastic.co.
26. "DynamoDB Documentation". Amazon Web Services.
27. Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley. – 2002. – С. 68-280.
28. Freeman, E.; Robson, E. Head First Design Patterns. 2nd Edition. O'Reilly Media. – 2020. – С. 12-317.
29. Salton, G., & Buckley, C. "Term-weighting Approaches in Automatic Text Retrieval." Information Processing & Management. – 1988. – С. 513–523.
30. "JWT Documentation". Jwt.io.
31. "Nginx Documentation". Nginx.org.
32. "Docker Documentation". Docker.com.
33. "Kubernetes Documentation". Kubernetes.io.
34. "Linux Server Administration Guide". Linux.org.
35. Spraul, V. Think Like a Programmer: An Introduction to Creative Problem Solving. No Starch Press. – 2019. – С. 23-320.
36. "AWS Documentation". Amazon Web Services.
37. "Web Performance Optimization Guide". Google Developers.
38. Dean, J.; Ghemawat, S. "MapReduce: Simplified Data Processing on Large Clusters". – 2004. – С. 30-85.
39. Robbins, J. Debugging Applications for Microsoft .NET and Microsoft Windows. 2nd Edition. Microsoft Press. – 2004. – С. 55-220.
40. Freeman, E.; Robson, E. Head First Design Patterns. 2nd Edition. O'Reilly Media. – 2020. – С. 68-185.

ДОДАТКИ

Додаток А

Фрагмент коду сервера

```
import 'reflect-metadata';
import cors from 'cors';
import path from 'path';
import { createServer } from 'http';
import { Server } from 'socket.io';
import express, { Express } from 'express';
import { createConnection } from 'typeorm';
import cookieParser from 'cookie-parser';
import { env } from './env';
import routes from './api/routes';
import ormconfig from './config/ormconfig';
import { handlers as socketHandlers } from './socket/handlers';
import { logger } from './common/utils/logger.util';
import {
  errorHandlerMiddleware,
  auth as authorizationMiddleware,
  socketInjector as socketMiddleware,
} from './api/middlewares';
import { SocketEvents } from './common/enums/socket';

const { nodeEnv, url, port } = env.app;

const app: Express = express();
const httpServer = createServer(app);

const io = new Server(httpServer, {
  cors: {
    origin:    nodeEnv    ===    'production'    ?    url    :
'http://localhost:3000',
    methods: ['GET', 'POST'],
  },
});

io.on(SocketEvents.CONNECTION, socketHandlers);

app.use(cors());
app.use(express.static(path.join(__dirname, './public')));
app.use(express.json({ limit: '50mb' }));
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());

app.use('/api/', authorizationMiddleware, socketMiddleware(io));

routes(app);

app.use(errorHandlerMiddleware);
```

```
app.use('*', (_req, res) => {
  res.sendFile(path.join(__dirname, '/public/index.html'));
});

httpServer.listen(port, async () => {
  try {
    await createConnection(ormconfig);
  } catch (error) {
    logger.info(`App started with error: ${error}`);
  }
  logger.info(`Server is running at ${port}.`);
});

export default app;
```

```
import { Express } from 'express';
import userRoute from './user-route';
import authRoute from './auth-route';
import pageRoute from './page-route';
import workspaceRoute from './workspace-route';
import tagRoute from './tag-route';
import teamRoute from './team-route';
import skillRoute from './skill-route';
import commentReactionRoute from './comment-route';
import notificationRoute from './notification-route';
import githubRoute from './github-route';
import linkRoute from './link-route';
import notificationSettingsRoute from './notification-settings-route';

const routes = (app: Express): void => {
  app.use('/api/users', userRoute);
  app.use('/api/pages', pageRoute);
  app.use('/api/auth', authRoute);
  app.use('/api/workspaces', workspaceRoute);
  app.use('/api/tags', tagRoute);
  app.use('/api/teams', teamRoute);
  app.use('/api/skills', skillRoute);
  app.use('/api/comments', commentReactionRoute);
  app.use('/api/notifications', notificationRoute);
  app.use('/api/github', githubRoute);
  app.use('/api/links', linkRoute);
  app.use('/api/notifications-settings', notificationSettingsRoute);
};
export default routes;
```

Додаток Б

Фрагменти коду сервісу pages

```
export const createPage = async (
  userId: string,
  workspaceId: string,
  body: IPageRequest,
): Promise<IPage> => {
const                      userWorkspaceRepository                    =
getCustomRepository(UserWorkspaceRepository);
  const userWorkspace =
    await userWorkspaceRepository.findByUserIdAndWorkspaceIdDetailed(
      userId,
      workspaceId,
    );
  if (userWorkspace.status === InviteStatus.JOINED) {
    const { parentPageId, ...pageContent } = body;
    const { title, content } = pageContent;

    const pageRepository = getCustomRepository(PageRepository);
    const { id } = await pageRepository.save({
      authorId: userId,
      workspaceId,
      parentPageId,
      pageContents: [pageContent],
    });
    const userRepository = getCustomRepository(UserRepository);
    const user = await userRepository.findById(userId);
const                       pageContentRepository                     =
getCustomRepository(PageContentRepository);
    await pageContentRepository.save({
      title,
      content,
      authorId: userId,
      pageId: id,
    });
    await elasticPageContentRepository.upsert({
      title,
      content,
      pageId: id,
      workspaceId,
    });
    const page = await pageRepository.findByIdWithContents(id);

    const userPermissionRepository = getCustomRepository(
      UserPermissionRepository,
    );
    await userPermissionRepository.createAndSave(
      user,
```

```
      page,
      PermissionType.ADMIN,
    );                const        usersPermissions        =        await
userPermissionRepository.findByPageId(
      parentPageId,
    );
    for (const userPermission of usersPermissions) {
      await userPermissionRepository.createAndSave(
        userPermission.user,
        page,
        userPermission.option,
      );
    }

    const teamPermissionRepository = getCustomRepository(
      TeamPermissionRepository,
    );                    const      teamsPermissions      =      await
teamPermissionRepository.findByPageId(
      parentPageId,
    );
    for (const teamPermission of teamsPermissions) {
      await teamPermissionRepository.createAndSave(
        teamPermission.team,
        page,
        teamPermission.option,
      );
    }
   const              workspaceAdmins              =              await
userWorkspaceRepository.findWorkspaceAdmins(
      workspaceId,
    );

    for (const workspaceAdmin of workspaceAdmins) {
      await userPermissionRepository.createAndSave(
        workspaceAdmin.user,
        page,
        PermissionType.ADMIN,
      );
    }
    return { ...mapPageToIPage(page), permission: PermissionType.ADMIN
};
  } else {
    throw new HttpError({
      status: HttpCode.NOT_FOUND,
      message: HttpErrorMessage.DELETED_FROM_WORKSPACE,
    });
  }
};
```