Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти

на тему

“ Дослідження та програмна реалізація NoSQL баз даних для підвищення ефективності роботи систем обробки даних”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-23М
ОПП «Комп'ютерні науки»
спеціальності 122 «Комп'ютерні науки»
_____ Д.В.Погорілий.
«____» _____ 2024р.

Керівник проекту
кандидат технічних наук, доцент
_____ К.М. Марченко
«____» _____ 2024р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *магістр*
Галузь знань *12 "Інформаційні Технології"*
Спеціальність *122 "Комп'ютерні науки"*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерні науки"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
"____" _____2024_року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Погорілому Денису Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація NoSQL баз даних для підвищення ефективності роботи систем обробки даних*
2. Керівник роботи *Марченко Костянтин Миколайович, к.т.н.,доц*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу № 18-13 від 07.08.24
3. Строк подання роботи до захисту *08.1.2024 р.*
4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне дослідження та програмна реалізація застосування NoSQL баз даних в ІС*
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схеми алгоритму роботи додатку	*2 аркуша*
Діаграма процесів	*1 аркуш*
Маркетингове та економічне обґрунтування ІТ-проєкту	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Маркетингове та економічне обґрунтування ІТ-проєкту	Доренська А.О.	09.11.2024	17.11.2024
Охорона праці	Марченко К.М., к.т.н., доцент	03.11.2024	21.11.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	12.10.2024	
2.	Постановка задачі, оформлення ТЗ	18.10.2024	
3.	Розробка моделі компонента	23.10.2024	
4.	Розробка структур даних	25.10.2024	
5.	Розробка алгоритмів зв’язку та відображення	32.10.2024	
6.	Програмування алгоритмів	11.11.2024	
7.	Маркетингове та економічне обґрунтування іт-проєкту	13.11.2024	
8.	Розрахунки з охорони праці та техніки безпеки	16.11.2024	
9.	Оформлення ПЗ	18.11.2024	
10.	Попередній захист роботи	07.12.2024	

Дата видачі завдання
«__»_____2024

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Погорілий Д.В. Дослідження та програмна реалізація NoSQL баз даних для підвищення ефективності роботи систем обробки даних. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації застосувань NoSQL баз даних в розробці ПЗ.

Метою розробки є дослідження та програмна реалізація додатку з використанням NoSQL баз даних .

Об'єктом дослідження є процес реалізації ПЗ з застосуванням NoSQL баз даних.

Предметом дослідження є методи реалізації розробки ПЗ з застосуванням NoSQL СУБД.

Методи дослідження базуються на методах теорії кодування, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація проекту засобами фреймворку NodeJS та MongoDB.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися у браузері Chrome, Firefox, Safari. Програму розроблено в середовищі NodeJS з використанням БД MongoDB.

Ключові слова: комп'ютерна інженерія, веб-сайт, NodeJS, бази даних, ООП, NoSQL, MongoDB.

ANNOTATION

Pogorilyy D.V. Research and software implementation of NoSQL databases to increase the efficiency of data processing systems. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi 2024.

In this master's thesis, software was developed, which is intended for the implementation of NoSQL database applications in software development.

The purpose of the development is the research and software implementation of the application using NoSQL databases.

The object of the study is the process of software implementation using NoSQL databases.

The subject of the study is the implementation methods of software development using NoSQL databases.

Research methods are based on coding theory methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the project using the NodeJS and MongoDB frameworks.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used in Chrome, Firefox, and Safari browsers. The program was developed in the NodeJS environment using the MongoDB database.

Keywords: computer engineering, website, NodeJS, databases, OOP, NoSQL, MongoDB.

ЗМІСТ

					ВКРМ-122.24.0011.00.00.ПЗ	
Вим.	Арк.	№ докум.	Підпис	Дат		
Розроб.		Погорілий Д.В			*Дослідження та програмна реалізація NoSQL баз даних для підвищення ефективності роботи систем обробки даних*	Літ. / Арк. 1 / Аркушів
Перевір.		МарченкоК.М				
Н. Контр.		Коваленко А.С				ЦНТУ КН-23М
Затверд.		Смірнов О.А.				

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

ІС	–	інформаційна система.
ПЗ	–	Програмне забезпечення.
Хеш	–	Результат обробки даних хеш-функцією.
CSS3	–	Cascading Style Sheets 3 – каскадні таблиці стилів третього покоління.
HTML	–	HyperText Markup Language – мова розмітки гіпертексту.
HP	–	скриптова мова загального призначення.
БД	–	База даних.
JSON	–	JavaScript Object Notation – текстовий формат обміну даними.
SQL	–	Structured query language – мова структурованих запитів.
ПК	—	Персональний комп’ютер
NoSQL		not only SQL

ВСТУП

Актуальність теми. У сучасному інформаційному світі, де обсяги даних швидко збільшуються, ефективне управління та обробка інформації стає надзвичайно важливим завданням. Традиційні реляційні бази даних, які тривалий час були основою для зберігання й організації даних, мають певні обмеження в обробці великих обсягів інформації та в умовах високих навантажень.

Використання NoSQL (Not Only SQL) баз даних виходить за межі звичних підходів, пропонуючи нові можливості для зберігання, обробки та взаємодії з даними. Ці бази відзначаються гнучкістю схем, високою швидкістю роботи з великими обсягами даних і можливістю горизонтального масштабування для задоволення зростаючих потреб систем.

Магістерське дослідження спрямоване на аналіз можливостей і особливостей використання NoSQL баз даних в інформаційних системах. Зокрема, розглядаються переваги та обмеження NoSQL технологій, їхній вплив на продуктивність та масштабованість систем, а також потенціал для застосування в конкретних задачах. Також досліджуються практичні аспекти впровадження NoSQL рішень у реальних проектах і проводиться їх порівняння з традиційними реляційними базами даних.

Висновки мого дослідження стануть корисним ресурсом для розробників, системних архітекторів та спеціалістів з управління даними в умовах постійної еволюції технологій. Робота сприятиме поглибленню знань у сфері баз даних і допоможе визначити найбільш ефективні стратегії використання NoSQL підходів для розробки сучасних інформаційних систем.

Мета й завдання дослідження. Метою роботи є дослідження та створення програмної системи для аналізу ефективності застосування NoSQL баз даних у сучасних інформаційних системах. Зокрема, дослідження охоплює оцінку можливостей, переваг і обмежень цих технологій, а також аналіз їхнього впливу на якість та швидкість обробки даних у різних сценаріях.

Для досягнення цієї мети визначено план дослідження, що включає такі завдання:

Аналіз сучасного стану баз даних:

- Провести огляд поточного стану розвитку баз даних, виявити ключові тенденції та основні виклики, що виникають перед інформаційними системами в умовах швидких технологічних змін.

Дослідження технічних аспектів NoSQL баз даних:

- Глибоко дослідити технічні характеристики різних типів NoSQL баз даних, зокрема гнучкість у схемах, продуктивність, масштабованість і можливості інтеграції з іншими технологіями.

Оцінка переваг та обмежень NoSQL технологій:

- Провести критичний аналіз переваг і обмежень NoSQL баз даних у порівнянні з традиційними реляційними системами.

Дослідження впливу на швидкодію та масштабованість систем:

– Дослідити, як використання NoSQL баз даних впливає на продуктивність і масштабованість інформаційних систем за різних навантажень і обсягів даних.

Аналіз практичних аспектів впровадження:

– Розглянути реальні приклади впровадження NoSQL технологій у проєктах, досліджуючи переваги та виклики, що виникають під час їхнього використання.

Порівняння з традиційними реляційними базами даних:

На основі отриманих результатів розробити рекомендації для розробників та архітекторів інформаційних систем, зважаючи на специфіку завдань та потреб.

Об’єкт дослідження. Застосування NoSQL для створення інформаційної системи управління HR-процесами.

Предмет дослідження. NoSQL бази даних і їх використання в інформаційних системах.

Методи дослідження. включають методи зберігання даних, математичної статистики, теорії масового обслуговування, розробки програмного забезпечення, теорії алгоритмів та об’єктно-орієнтованого проектування.

Наукова новизна отриманих результатів. У ході дослідження отримано такі результати:

– удосконалено метод вибору NoSQL баз даних для роботи з сучасними інформаційними системами;

– розроблено інформаційну систему на основі NoSQL, що має розширені можливості та вищу швидкість доступу до даних порівняно з існуючими аналогами.

Практична цінність отриманих результатів. Включає висновки з впровадження NoSQL технологій у реальні проєкти, надаючи розробникам і архітекторам конкретні вказівки та рекомендації для ефективного застосування NoSQL.

Для практичного використання отриманих теоретичних результатів розроблено систему автоматизації HR-процесів на основі MongoDB, що використовує сучасні інформаційні технології.

Достовірність наукових результатів підтверджується теоретичними обґрунтуваннями, комп’ютерним моделюванням, дослідженнями параметрів у функціонуючій системі, а також відповідністю результатів з науковою літературою.

Таким чином, дослідження та розробка системи аналізу використання NoSQL баз даних у сучасних інформаційних системах є актуальним завданням, яке буде розглянуто в даній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

У сучасному цифровому світі, де величезні обсяги даних генеруються в реальному часі і стрімко зростають, питання зберігання, управління та аналізу інформації стає надзвичайно актуальним та складним завданням для багатьох галузей. Такий швидкий ріст обсягів даних супроводжується як значними викликами, так і новими можливостями, дослідження та вирішення яких є пріоритетним завданням для сучасних інформаційних систем.

Інформаційні системи (ІС) у сучасному світі виконують різноманітні й важливі функції, які безпосередньо впливають на ефективність і конкурентоспроможність підприємств та організацій. Вони охоплюють широкий спектр завдань, зосереджених на обробці, зберіганні та передачі інформації з метою підтримки прийняття рішень та забезпечення ефективного функціонування організаційних процесів.

Автоматизація бізнес-процесів

Однією з основних функцій інформаційних систем є автоматизація бізнес-процесів. Вони сприяють оптимізації як внутрішніх, так і зовнішніх операцій компанії, скорочуючи час на виконання завдань і підвищуючи точність процесів. За допомогою ІС можна автоматизувати фінансовий облік, управління запасами, обробку замовлень та інші важливі етапи діяльності.

Підтримка прийняття рішень

Інформаційні системи надають необхідну інформацію для ухвалення стратегічних і тактичних рішень. Вони забезпечують аналіз даних, підготовку звітів і прогнозування, що сприяє ефективному управлінню ресурсами та розробці стратегій розвитку.

Забезпечення доступу до інформації

Інформаційні системи надають структурований і швидкий доступ до даних для різних користувачів всередині організації. Це включає створення

зручних інтерфейсів, можливість обміну даними та забезпечення конфіденційності інформації.

Впровадження електронного бізнесу

Інформаційні системи мають важливе значення для впровадження електронного бізнесу, сприяючи взаємодії з клієнтами, партнерами та постачальниками через Інтернет. Вони дають змогу створювати та підтримувати електронні платформи, автоматизувати транзакції та адаптуватися до змін на ринку.

Забезпечення безпеки інформації

Інформаційні системи включають засоби захисту конфіденційності, цілісності та доступності даних. Вони допомагають виявляти та запобігати кіберзагрозам, забезпечуючи надійну безпеку інформації компанії.

Управління взаємодією з клієнтами

Інформаційні системи сприяють підтримці зв'язків із клієнтами, забезпечуючи роботу CRM-систем для зберігання та аналізу даних про клієнтів, а також автоматизацію процесів обслуговування та підтримки.

1.1. Призначення системи

База даних є ключовою і складною складовою інформаційних систем, які призначені для обробки та зберігання даних. У сучасному світі спостерігається експоненційний ріст обсягів даних, які генеруються в різних сферах, таких як медицина, фінанси, наукові дослідження, соціальні мережі тощо.

Крім того, дані стають усе різноманітнішими і можуть включати текст, графічні структури, геопросторові дані, зображення, відео та інші формати.

Метою цієї роботи є створення інформаційної системи для автоматизації HR-процесів із застосуванням сучасних технологій та нереляційних систем управління даними, зокрема з урахуванням критеріїв узгодженості у базах даних NoSQL на етапі проектування системи. Традиційні реляційні бази даних можуть

мати обмеження у роботі з різноманітними даними, тому NoSQL бази відкривають нові можливості для ефективної їх обробки.

1.2. Область застосування

NoSQL бази даних широко використовуються в багатьох галузях, таких як фінанси, медицина, телекомунікації, аналітика даних, Інтернет речей (IoT), веб-розробка та інші. Дослідження та впровадження NoSQL баз даних в інформаційних системах можуть допомогти розробникам та архітекторам систем зрозуміти можливості використання цих технологій для вирішення різноманітних задач. Нижче розглянемо основні напрямки застосування таких баз даних.

Веб-застосунки та соціальні мережі

Великі платформи та соціальні мережі, де важлива швидкість доступу та масштабованість, активно використовують NoSQL бази даних. Гнучкість схем дозволяє легко змінювати та розширювати дані, а можливість масштабування забезпечує обробку великих обсягів інформації.

Аналітика та обробка великих даних

NoSQL бази даних підходять для зберігання та обробки великих обсягів даних в аналітичних системах. Завдяки масштабуванню вони забезпечують швидкий аналіз і отримання важливих інсайтів з великих датасетів.

Інтернет речей (IoT)

NoSQL бази даних ефективно працюють з даними, зібраними з різноманітних сенсорів та пристроїв IoT. Гнучкі схеми дозволяють зберігати різні типи даних, а масштабованість допомагає впоратися з великим обсягом одночасних записів.

Електронна комерція

В індустрії електронної комерції, де необхідна обробка великих обсягів інформації про товари, клієнтів та транзакції, NoSQL бази даних забезпечують високу продуктивність та масштабованість.

Геопросторові дані

В сферах, пов’язаних із геолокацією, як-от геонавігація, картографія та агротехнології, NoSQL бази даних ефективно зберігають та обробляють великі обсяги геоданих.

Логістика та управління ланцюгом постачання

У сфері логістики та управління ланцюгом постачання, де необхідно контролювати та керувати потоками товарів та інформацією, NoSQL бази дозволяють оперативно виконувати операції з великими обсягами даних.

Наукові дослідження

У наукових дослідженнях, де часто обробляються великі та складні набори даних, NoSQL бази використовуються для зберігання та швидкого доступу до експериментальних даних.

Отже, дослідження та розробка системи для аналізу використання різних типів баз даних у сучасних інформаційних системах є актуальним завданням, що потребує вирішення в межах цієї магістерської роботи.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

Сучасний стан автоматизації кадрових процесів відзначається швидким розвитком, адже багато професій виходять на новий рівень. Системи для автоматизації та цифровізації HR-процесів набувають популярності й розширюють свої можливості. Діджиталізація поступово стає необхідною складовою сучасного HR, а комплексні фахівці активно допомагають компаніям у цьому процесі. Це важливий тренд у сфері управління людськими ресурсами, який сприяє усвідомленню важливості автоматизації частини HR-процесів, щоб звільнити ресурси для більш стратегічних завдань. Такі завдання, як ведення електронних таблиць чи навчальні матеріали, можна автоматизувати, проте варто залишити людський аспект у взаємодії з HR-фахівцем, щоб підтримувати мотивацію та професійний розвиток співробітників.

Потреба в автоматизації управлінських HR-задач з'явилася відносно недавно, тому ця галузь все ще знаходиться на етапі розвитку. Вибираючи систему для автоматизації управління, необхідно враховувати особливості кожної організації, орієнтуючись на максимальну відповідність алгоритмів аналізу інформації, що вже існують у компанії, та їх інтеграцію в систему.

Головною проблемою для HR-фахівців залишається складність управління великим обсягом завдань, функцій та процесів, які потрібно виконувати швидко та ефективно.

Спеціалізовані системи автоматизації управління персоналом дозволяють формалізувати й оптимізувати всі процеси, пов’язані з діяльністю співробітників, що робить управління людськими ресурсами значно ефективнішим. Автоматизована система управління кадровими процесами не лише впорядковує управління персоналом, але й забезпечує моніторинг і контроль, у яких зводиться до мінімуму ймовірність помилок, порушень та зловживань.

Розробка інформаційної системи для автоматизації кадрових процесів є актуальним напрямом наукових досліджень. Під час аналізу кадрової системи було встановлено важливість серверної частини як сховища інформації, де зазвичай використовуються системи управління базами даних (СУБД). Зважаючи на сучасні тенденції, системи NoSQL наразі отримують особливу увагу.

У традиційних реляційних системах дані зберігаються у вигляді таблиць, а структура бази даних визначається заздалегідь. Проте при створенні великих систем (Big Data) з використанням реляційних СУБД розробники стикаються з труднощами, зокрема забезпеченням стійкості до системних помилок при великій кількості вузлів.

Щоб подолати ці обмеження реляційних баз даних, було створено альтернативні засоби для зберігання та обробки даних, відомі як NoSQL-бази даних. Піонерами в цій галузі стали компанії Google та Amazon.

У NoSQL-базах даних багаторазова реплікація записів використовується для забезпечення високої відмовостійкості. Проте ці системи мають недолік — відсутність підтримки транзакцій і блокувань, що створює проблему узгодженості під час реплікації.

До важливих показників координації реплікацій у NoSQL-базах належать ймовірність читання застарілих даних під час розповсюдження оновлень між вузлами, час очікування початку читання оновлених записів, кількість версій записів у системі, час їх обробки, а також імовірність заборони доступу до запису. Оцінка цих характеристик на етапі проєктування допомагає уникнути ручного налаштування численних параметрів для різних типів записів у базі даних і необхідності створювати повномасштабні моделі для тестування на екстремальних навантаженнях.

Оскільки технологія побудови інформаційних систем на основі NoSQL-баз даних є порівняно новою, відповідні математичні моделі для оцінки

показників узгодженості реплікацій або ще не розроблені, або не повною мірою відповідають вимогам.

2.1. Огляд існуючих систем

Існує кілька популярних систем автоматизації кадрових процесів. Розглянемо найбільш відомі рішення та визначимо їх основні переваги й недоліки. Однією з таких систем є BambooHR.

BambooHR - це платформа для управління персоналом, яка забезпечує зручний інтерфейс для організації кадрових процесів. Серед основних переваг цієї системи:

- інтуїтивний інтерфейс, що робить її зручною для використання навіть новачками;

- можливість налаштування звітів і аналітики, що допомагає в отриманні актуальної інформації для прийняття рішень;

- функціонал для управління життєвим циклом співробітників, включаючи прийом на роботу, адаптацію та звільнення;

Проте, у BambooHR є й певні недоліки:

- висока вартість для малого бізнесу, що може стати бар'єром для впровадження;

- обмежена кількість інтеграцій з іншими системами, що може ускладнити інтеграцію в існуючу IT-інфраструктуру компанії;

- функціональні обмеження для великих організацій - для деяких великих компаній система може не покривати всі необхідні HR-процеси.

Далі розглянемо інші системи, щоб порівняти їх переваги та недоліки з BambooHR головне вікно якої показано на рисунку 2.1.

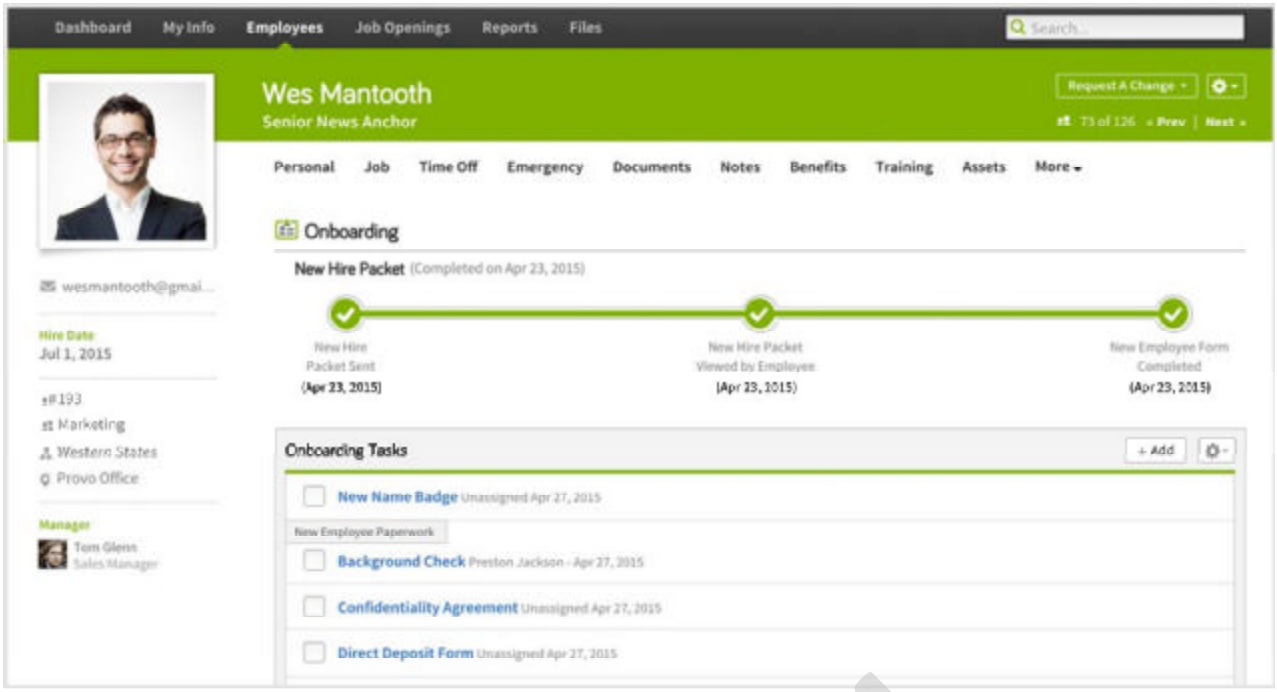


Рисунок 2.1 - Система BambooHR

BambooHR — це система для HR-фахівців, яка надає комплексні рішення для управління персоналом, що особливо підходить для малого та середнього бізнесу, який активно зростає. Основні можливості BambooHR включають:

- пошук і моніторинг кандидатів: інструменти для ефективного управління рекрутингом;

- відстеження винагород: контроль за процесом нарахування бонусів та інших винагород;

- інтеграція з іншими платформами: підтримує підключення до інструментів Zapier, Okta, OneLogin, Indeed, Greenhouse, а також має потужну аналітику;

- вбудований календар подій: допомагає стежити за важливими датами й подіями;

- управління навчанням персоналу: функції для моніторингу процесу навчання співробітників;

- документація та FAQ: детальна інформація для користувачів системи;

- мобільний доступ: підтримка роботи на мобільних пристроях, що підвищує зручність для користувачів;

До недоліків BambooHR можна віднести відсутність автоматизації деяких бізнес-процесів, таких як:

- система рейтингу серед співробітників: відсутня функція, що дозволяє працівникам виставляти рейтинги один одному;

- збереження записів про керівників команд: немає інструменту для управління інформацією щодо керівників;

- історія оцінок співробітників: відсутній механізм відстеження попередніх оцінок ефективності співробітників;

Таким чином, BambooHR є відмінним варіантом для компаній, які потребують зручної системи управління персоналом з базовими функціями, але можуть зіткнутися з обмеженнями, якщо потрібні глибші аналітичні або рейтингові можливості.

Hurma - це нове рішення на ринку, яке об'єднує в собі процеси рекрутингу, HR і управління цілями та ключовими результатами (OKR). Система має пакети, що підходять для компаній різних розмірів — від зростаючих команд до великих підприємств.

Основні функції системи Hurma включають:

- моніторинг настрою працівників: інструменти для відстеження рівня задоволеності команди;

- організація структури підприємства у вигляді дерева: можливість створення візуального дерева компанії;

- синхронізація з Google Calendar: нагадування про події та заходи компанії;

- особистий кабінет кожного співробітника: індивідуальні профілі з інформацією про працівника;

- автоматизація HR-процесів: зокрема, перевірка ефективності, вітання нових співробітників, коригування під час випробувального періоду, проведення вихідних інтерв'ю;

- телефонний зв'язок: можливість зв'язатися з відділом кадрів і керівництвом;

- підтримка OKR: управління цілями та ключовими результатами;

- управління відсутностями: інструменти для обліку відпусток, лікарняних, та віддаленої роботи;

- база вакансій і кандидатів: зручне зберігання та управління інформацією про вакансії та кандидатів;

- публікація вакансій на сайті компанії та інтеграція з LinkedIn для пошуку кандидатів;

- статистика процесу працевлаштування: аналітика щодо процесів рекрутингу та HR.

Недоліки системи Hurma:

- обмеження у фільтрах: неможливо відібрати кандидатів за певними курсами чи навичками;

- сортування вакансій тільки за назвою або датою створення;

- відсутність коментування завдань: немає можливості залишати коментарі до завдань, пов'язаних із затвердженням відпусток, дистанційної роботи та лікарняних;

Hurma пропонує комплексний підхід до управління персоналом, але має деякі обмеження, які можуть вплинути на зручність використання в окремих процесах.

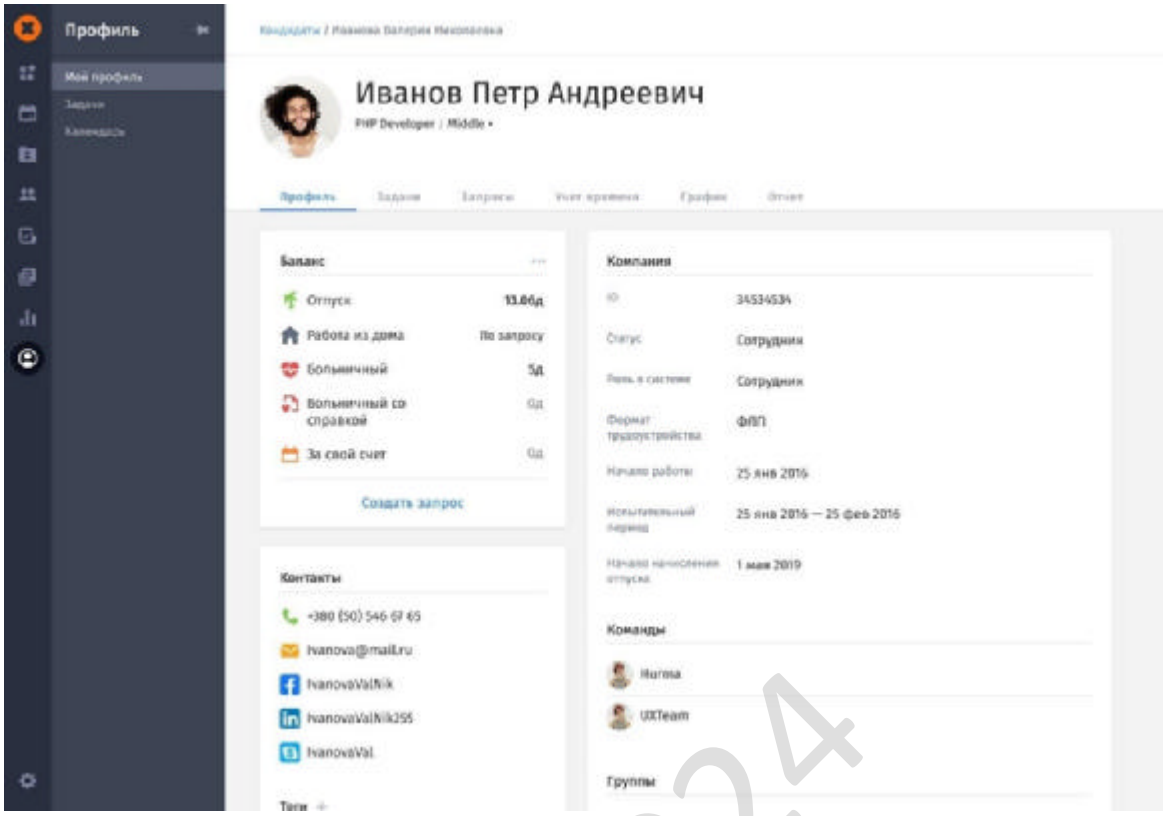


Рисунок 2.2 - Система Hurma System

Workable - це система для управління процесом підбору персоналу, яка об'єднує функції відстеження кандидатів (ATS) з платформою рекрутингу, що використовує потужні алгоритми на основі машинного навчання для пошуку відповідних кандидатів.

Основні можливості Workable включають:

- швидкий підсумковий аналіз: інструменти для оперативного огляду та аналізу даних про кандидатів;

- автоматизація пошуку співробітників: конструктор для створення анкет, резюме та вакансій, що спрощує процес відбору;

- інтеграція з порталом працевлаштування: підключення до популярних сайтів для розміщення вакансій;

- інтеграція з LinkedIn та можливість пошуку кандидатів у соціальних мережах для розширення охоплення;

- брендування інтерфейсу: можливість налаштування дизайну інтерфейсу відповідно до стилю компанії;

- синхронізація календаря: допомагає HR-відділу координувати розклад завдань та зустрічей;

- імпорт існуючих баз даних: можливість завантаження наявної інформації про кандидатів і працівників;

- персональний менеджер і служба підтримки: допомога у використанні системи через телефон і електронну пошту;

- мобільний додаток: для зручного доступу до функцій системи з мобільних пристроїв;

- розширення Google Chrome: швидкий доступ до функцій пошуку кандидатів безпосередньо з браузера.

Workable забезпечує зручний інтерфейс та потужні інструменти для рекрутингу, що дозволяють ефективно керувати процесом підбору персоналу в компанії різного розміру.

Zoho People - це програмне забезпечення для управління персоналом, спеціально розроблене для малого та середнього бізнесу. Завдяки простому інтерфейсу система готова до використання відразу після налаштування.

Основні можливості Zoho People включають:

- управління відсутністю: функціонал для обліку відпусток, лікарняних та інших відсутностей;

- шаблони типових кадрових документів: стандартні шаблони, які спрощують документообіг у відділі кадрів;

- мобільний додаток та веб-версія: зручність доступу як на мобільних пристроях, так і через браузер;

- моніторинг часу: можливість відстеження робочого часу працівників;

- аналітика для HR: інструменти для аналізу ключових показників HR-процесів;

- функції перевірки ефективності: оцінка продуктивності співробітників за допомогою різних інструментів;

- електронний підпис: забезпечує зручність і юридичну силу електронних підписів документів;

- внутрішня аналітика та звітність: можливість створення звітів та аналізу даних для управлінських рішень.

Zoho People є зручним рішенням для компаній, які шукають просту та функціональну систему для управління персоналом.

Роль HR-менеджера надзвичайно важлива для успіху компанії. Він виконує функцію посередника між працівниками та керівництвом, працює над формуванням бренду компанії, удосконаленням корпоративної культури, мотивацією та адаптацією співробітників. Окрім цього, на HR-менеджера покладається ведення документації, відстеження лікарняних, відпусток, відряджень, проведення співбесід, планування заходів тощо. Усі ці завдання потребують високої точності та відповідальності, оскільки від роботи HR-спеціаліста часто залежить ефективність інших співробітників.

Автоматизація дозволяє значно зменшити обсяг рутинних завдань, оптимізувати їх виконання та сконцентруватися на більш важливих аспектах, таких як підвищення лояльності команди, мотивація та продуктивність співробітників, адаптація новачків, розвиток корпоративної культури та інші завдання стратегічного значення.

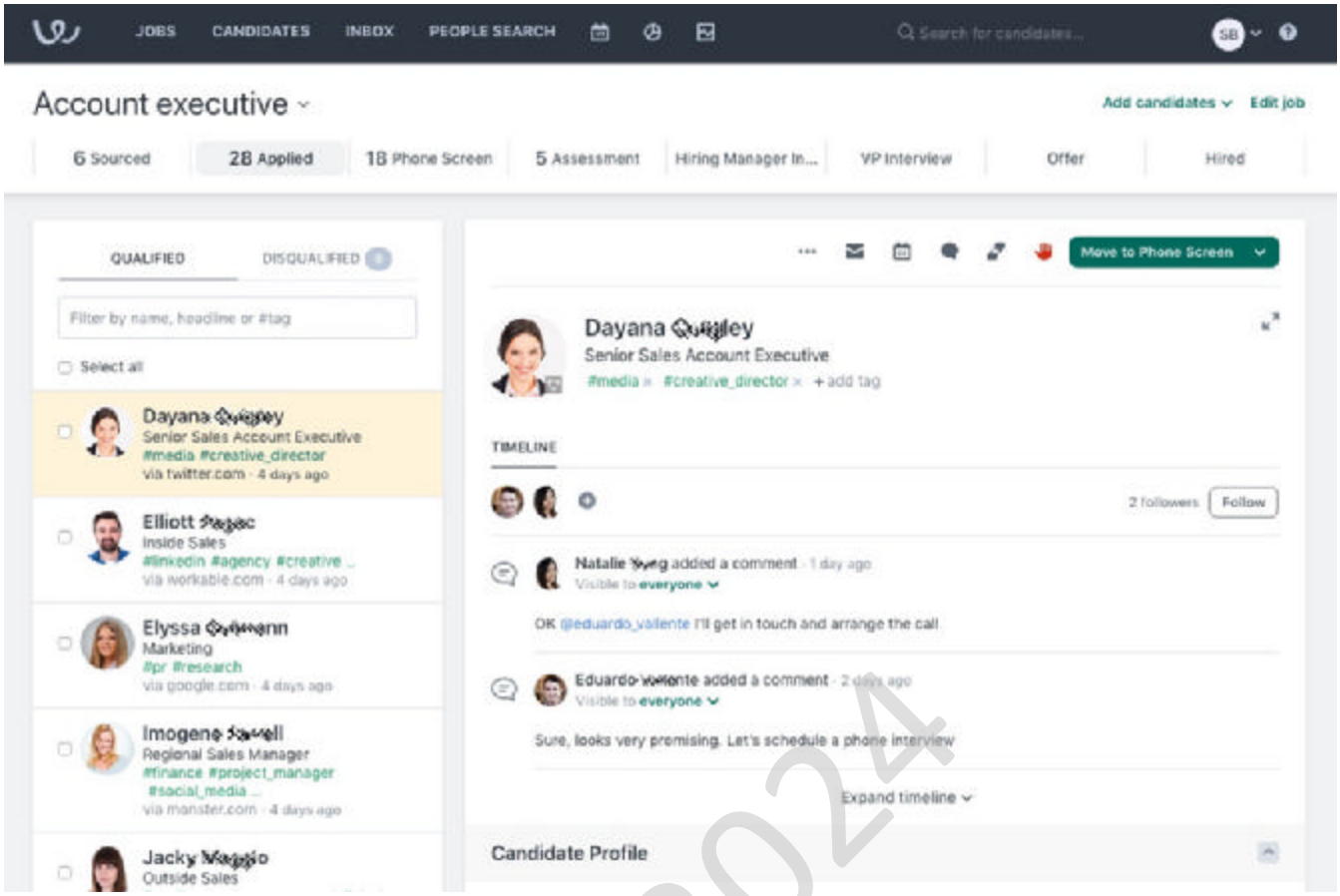


Рисунок 2.3 - Система Workable

2.2 Обґрунтування вибору методів розробки

У процесі аналізу HR-системи було виявлено критичну роль серверної частини, яка забезпечує зберігання інформації. Для цього використовується система управління базами даних (СУБД), і в останні роки особлива увага приділяється системам NoSQL.

Протягом останніх кількох років реляційні СУБД домінували у сфері обробки даних, де інформація зберігається у вигляді таблиць з чітко визначеною схемою. Однак під час створення великих систем (Big Data) з реляційними СУБД розробники стикнулися з кількома серйозними викликами:

- ускладнена агрегація даних: Процедура агрегації ускладнюється через необхідність зчитування записів з великої кількості зв'язаних таблиць, що призводить до проблем з відсутністю відповідності;

- протиріччя в обробці неструктурованих даних: Існує конфлікт між потребою зберігати великі обсяги неструктурованих даних і необхідністю їх структуризації через розробку схеми бази даних;

- високі витрати на обладнання: для зберігання великих обсягів інформації потрібні дорогі спеціалізовані апаратно-програмні комплекси паралельних систем баз даних, такі як Teradata або Sun Oracle Database Machine.

- проблеми з відмовостійкістю: При наявності великої кількості вузлів виникають труднощі з забезпеченням необхідної відмовостійкості системи.

Відповіддю на виклики, що виникли у традиційних реляційних базах даних, стали альтернативні рішення для зберігання та обробки даних, відомі як «бази даних NoSQL». Піонерами в цій галузі виступили компанії Google і Amazon.

У системах NoSQL реалізується багаторазова реплікація записів, що забезпечує високу відмовостійкість. Однак ці системи мають свої обмеження: вони не підтримують механізми транзакцій і блокувань, що створює проблеми з відповідністю реплікації.

Ключовими показниками координації репліки в базах даних NoSQL є:

- ймовірність читання застарілого запису: під час оновлень, які розподіляються по вузлах системи;

- час очікування початку читання записів: з оновлених серверів;

- кількість версій записів: у базі даних NoSQL та час їх обробки;

- ймовірність заборони доступу до запису: у базі даних.

Ці аспекти повинні бути оцінені на етапі проектування системи, щоб уникнути ручного вибору необхідних значень параметрів для великої кількості типів записів на етапі налагодження та потреби в масштабному моделюванні при екстремальному навантаженні на систему.

Оскільки технологія розробки інформаційних систем на основі баз даних NoSQL є досить новою, існуючі математичні моделі для оцінки показників відповідності реплік або відсутні, або неадекватні. Це створює додаткові труднощі в процесі проектування та оптимізації таких систем.

Типи NoSQL баз даних

Незважаючи на різноманітність баз даних NoSQL, вони виконують певні спільні функції, пов’язані з координацією реплік:

- розміщення реплік: Забезпечення узгодженості записів у кластері під час оновлення;

- узгодження версій реплік: Об'єднання кількох версій записів в один під час відстеження змін;

- звірка реплік: відновлення реплік після усунення несправностей у вузлі.

Бази даних NoSQL зазвичай використовують два основних методи налаштування та оновлення реплік: головний-підлеглий і кільцевий.

- головний-підлеглий метод: У цьому випадку дані зберігаються на головному вузлі, а їх реплікація відбувається на підлеглі вузли. Усі зміни вносяться на головному вузлі і зберігаються в його пам’яті. Підлеглі вузли періодично опитують головний вузол, щоб отримати накопичені зміни і зберегти їх у своїй пам’яті. Прикладами таких баз даних є MongoDB, HBase, Neo4j тощо.

- кільцевий метод: У цьому випадку дані розміщуються на секціях (вузлах), впорядкованих за серверами у вигляді кільця. При розміщенні даних «на кільці» необхідно визначити, скільки секцій (вузлів) буде в системі. Це забезпечує ефективну маршрутизацію запитів і балансування навантаження.

Обидва ці методи дозволяють досягти високої доступності і надійності систем, але вимагають ретельного проектування та налаштування для забезпечення відповідності між репліками.

На рисунку 2.4 представлено приклад розміщення розділів у кільцевій архітектурі. У цьому прикладі визначено 64 розділи, які розподілені по колу на

трьох фізичних серверах: A, B та C. Кожному серверу буде виділено приблизно 21 або 22 розділи (64 розділи поділені на 3 сервера).

Цей підхід дозволяє рівномірно розподілити навантаження між серверами, що сприяє підвищенню відмовостійкості системи. Кожен сервер зберігає частину даних, що дозволяє уникнути перевантаження одного вузла та підвищує загальну ефективність роботи бази даних. Кільцева архітектура також спрощує процес додавання нових серверів до системи, оскільки нові вузли можуть бути легко інтегровані у вже існуючу структуру без значних змін у конфігурації.

Коли запис додається до бази даних, для нього обчислюється хеш-ключ, який визначає номер розділу (віртуального вузла, v-node). Цей v-node вказує на сервер, де буде зберігатися запис. Інші репліки записів (кількість яких становить N-1) розміщуються на наступних (N-1) серверах, які розташовані в кільцевій архітектурі за годинниковою стрілкою від першого сервера. Прикладами розподілених кільцевих баз даних є Riak, Amazon Dynamo та інші.

Використання віртуальних вузлів (v-nodes) має кілька переваг:

- відмовостійкість: Якщо один з вузлів стає недоступним (через поломку або планове обслуговування), навантаження на систему розподіляється рівномірно між залишеними активними вузлами. Це дозволяє зберегти працездатність системи та зменшити ризик перевантаження інших вузлів;
- автоматичне балансування навантаження: Коли недоступний вузол знову стає активним або до системи додається новий вузол, він приймає приблизно таке ж навантаження від інших активних вузлів. Це сприяє більшій стабільності роботи системи;
- гнучкість у налаштуванні: Кількість віртуальних вузлів, які підпорядковуються реальному вузлу, може бути визначена залежно від потужності сервера. Це дозволяє ефективно балансувати навантаження в неоднорідній інфраструктурі, де різні сервери можуть мати різну продуктивність.

Таким чином, віртуальні вузли забезпечують ефективне управління даними в розподілених системах, підвищуючи їхню надійність і продуктивність.

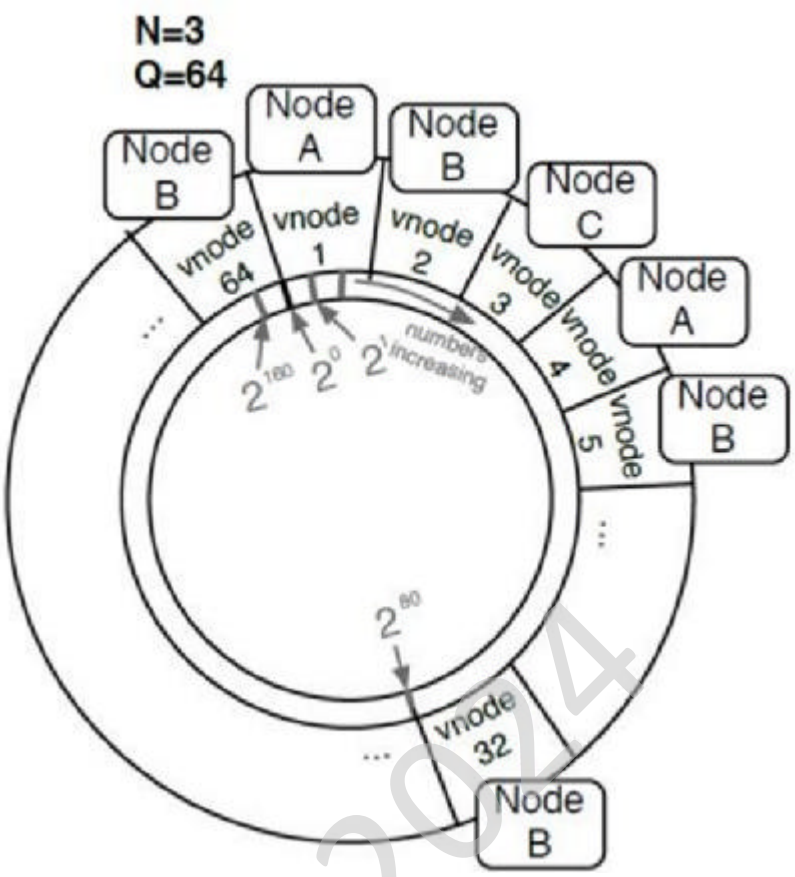


Рисунок 2.4 - Кільце з 64 v-вузлами і трьома серверами

Реплікація є важливим аспектом у системах NoSQL, оскільки вона сприяє підвищенню продуктивності та відмовостійкості. Однак, коли запис оновлюється, оновлення не завжди поширюється миттєво на всі репліки. Зазвичай існує так зване "вікно неузгодженості" — це період часу, протягом якого дані на різних вузлах можуть не бути синхронізованими. У цьому контексті існують різні рівні узгодженості, зокрема:

- сувора послідовність: У цьому випадку кожен запит на читання завжди повертає останнє оновлення запису. Це забезпечує максимальний рівень узгодженості, але може знижувати доступність системи;
- погана узгодженість: Ця форма узгодженості не гарантує, що запит на читання завжди поверне останнє оновлення запису. Тобто, читач може отримати застарілі дані;

- можлива узгодженість: Це особливий випадок слабкої узгодженості, де система гарантує, що якщо нові оновлення не відбуваються, то, зрештою, запит на читання поверне останнє оновлення.

Узгодженість тісно пов'язана з доступністю системи та стійкістю до втрати з'єднання, що було сформульовано в теоремі CAP (консистенція, доступність, толерантність до розділів) Еріком Брюером. Теорема стверджує, що можна побудувати розподілену систему, яка буде відповідати лише двом з трьох властивостей одночасно:

- узгодженість (C): Система забезпечує, що всі вузли бачать одні й ті ж дані в один і той же час;

- доступність (A): Кожен запит отримує відповідь, навіть якщо одна або кілька частин системи не відповідають;

- толерантність до розділів (P): Система продовжує працювати, навіть якщо з'єднання між її частинами втрачається;

Додатково, при розгляді реплікації та узгодженості, важливо позначити деякі параметри:

- N: Кількість вузлів, на які в кінцевому підсумку буде репліковано оновлення (з певною затримкою);

- V: Кількість вузлів, в які повинні бути записані дані до того, як користувач отримає підтвердження про успішне завершення операції. Якщо V<NV < NV<N, то система продовжує реплікувати дані на решту N−VN - VN−V вузлів;

- R: Кількість вузлів, від яких база даних очікує отримання відповідей для успішного завершення читання записів.

Важливо зазначити, що відсутність блокувань в системах NoSQL дозволяє одночасно виконувати читання та запис одних і тих же даних на різних вузлах. Це може призводити до конфліктів, які, в свою чергу, потрібно вирішувати за допомогою спеціалізованих інструментів NoSQL або вручну.

Вирішення конфліктів у системах NoSQL є важливим завданням, оскільки одночасні оновлення записів можуть призводити до несумісностей. Ось кілька стратегій, які можна використовувати для управління конфліктами:

- використання міток часу: Це найпростіший метод, при якому кожен запис отримує мітку часу. У разі конфлікту, перевага надається запису з найновішою міткою часу. Хоча цей метод є простим у реалізації, його важко впровадити в кластері вузлів, оскільки мітки часу можуть бути отримані з різних джерел, що призводить до проблем з синхронізацією;
- векторні годинники (Vector Clocks): Це більш складний метод, який дозволяє відстежувати версії записів, зберігаючи інформацію про порядок оновлення. Векторний годинник є масивом пар <користувач, номер версії запису для цього користувача>. Кожен вузол у кластері підтримує свій векторний годинник, що дозволяє відстежувати, які зміни були внесені і в якому порядку;
- коли вузол оновлює запис, він інкрементує свій лічильник у векторі, що представляє цей запис;
- при конфлікті два різні записи можуть мати різні вектори годинника, що вказує на те, що один з них був оновлений пізніше або обидва мали окремі зміни;
- таким чином, векторні годинники дозволяють розрізняти незалежні оновлення та вирішувати конфлікти, зберігаючи всю історію змін.

Цей підхід не лише дозволяє визначити, який з записів є "останнім", але й зберігає інформацію про всі зміни, що можуть стати корисними для подальшого аналізу або відновлення даних.

Обидва методи мають свої переваги та недоліки, і вибір залежить від специфіки системи, вимог до узгодженості та необхідності вирішення конфліктів.

На рисунку 2.5 - опис ситуації з веденням вектора годинника та конфліктами при оновленні запису D у базі даних дуже добре ілюструє, як працює механізм управління версіями в NoSQL системах.

Нижче наведено сценарій оновлення запису D:

- початковий стан: Запис D знаходиться в базі даних у початковому стані;

- оновлення від A1;

- користувач A1 виконує перше оновлення запису D, в результаті чого виникає версія D1;

- після першого оновлення вектор годинника може виглядати так: A1: 1;

- потім A1 виконує ще одне оновлення, що створює нову версію D1, оновлюючи вектор годинника: A1: 2;

- оновлення від A2 та A3;

- паралельно (випадковий збіг) користувачі A2 та A3 читають запис D, який містить версію D1 (з вектором годинника A1: 2);

- користувач A2 вносить зміни і створює версію D2, при цьому його вектор годинника стає: A2: 1;

- користувач A3 також вносить зміни, створюючи версію D2, з вектором: A3: 1.

- Конфлікти.

У базі даних тепер є дві версії запису:

- D1: з вектором годинника A1: 2;

- D2: з векторами годинника A2: 1 та A3: 1;

Читання та об'єднання версій:

- коли користувач A1 намагається зчитати запис D, база даних повертає йому обидві версії: D1 та D2.

Для об'єднання оновлень, зроблених A2 і A3, база даних використовує вектор годинника. Вона об'єднує зміни з D2, створюючи нову версію, скажімо D3, з вектором:

- A1: 2, A2: 1, A3: 1;

Узгоджена версія:

- база даних зберігає узгоджену версію D3, яка містить всі ідентифікатори користувачів, що вносили зміни, що дозволяє зберегти історію змін.

Переваги використання векторного годинника:

- відстеження версій: Можливість бачити всі версії запису та ідентифікувати, які зміни були внесені різними користувачами;

- уникнення конфліктів: У разі конфлікту система може спробувати автоматично об'єднати оновлення на основі векторів годинника.

Цей механізм забезпечує певний рівень узгодженості у дистрибуційних системах, дозволяючи користувачам працювати паралельно, не блокуючи записи, але вимагаючи подальшого управління конфліктами при злитті змін.

Переваг і недоліків годинникового вектора, а також підходу до розв'язання конфліктів за допомогою атипового тегу, добре висвітлює ключові аспекти роботи з конфліктами в NoSQL системах. Давайте розглянемо ці пункти детальніше.

Переваги годинникового вектора

Відсутність єдиної точки відмови:

Завдяки дистрибуційній природі системи, використання векторів годинника не вимагає централізованої синхронізації часу. Це означає, що якщо один з вузлів виходить з ладу, інші вузли можуть продовжувати працювати без зупинки системи.

У традиційних системах, де використовується єдиний стандарт часу, будь-яка проблема з синхронізацією може призвести до серйозних збоїв.

Недоліки годинникового вектора

Неможливість автоматичного вирішення конфліктів:

- Коли виникають конфлікти між запитами, система не може автоматично визначити, яка версія запису є "правильною". Це потребує ручного втручання або додаткових механізмів для об'єднання даних.

Збільшення довжини тактового вектора

При численних оновленнях запису довжина вектора може збільшуватися, що ускладнює його обробку і зберігання.

Довгі вектори можуть займати більше пам'яті і потребують більше часу для порівняння та обробки.

Механізми скорочення векторів

У системах, таких як Riak, можна налаштувати частоту підстроювання вектора та максимальний розмір, щоб зменшити вплив цих недоліків. Це дозволяє підтримувати ефективність системи при масштабуванні.

Атемпоральний тег

Визначення та функція

Атемпоральний тег може бути представлений у вигляді хешу вмісту, GUID або лічильника, який генерується для кожного запису. Цей тег супроводжує дані при їх поверненні до користувача.

Перевірка при оновленні

Коли користувач вносить зміни, система порівнює тег, отриманий від користувача, з тегом, що зберігається в базі даних. Якщо теги не збігаються, це означає, що інші зміни були внесені після отримання даних користувачем, і операція відхиляється.

Користувачеві необхідно знову прочитати та оновити запис, що може додати до затримки, але забезпечує узгодженість даних.

Використання в CouchDB

Система CouchDB застосовує цей підхід, що дозволяє їй залишатися дистрибуційною та гнучкою. Користувачі можуть працювати з копіями даних, але система контролює узгодженість за рахунок атемпоральних тегів.

Цей механізм ілюструє важливість балансу між доступністю, узгодженістю і стійкістю системи, а також демонструє, як NoSQL бази даних намагаються вирішити ці проблеми через інноваційні підходи.Такий підхід використовується в системі CouchDB [30].

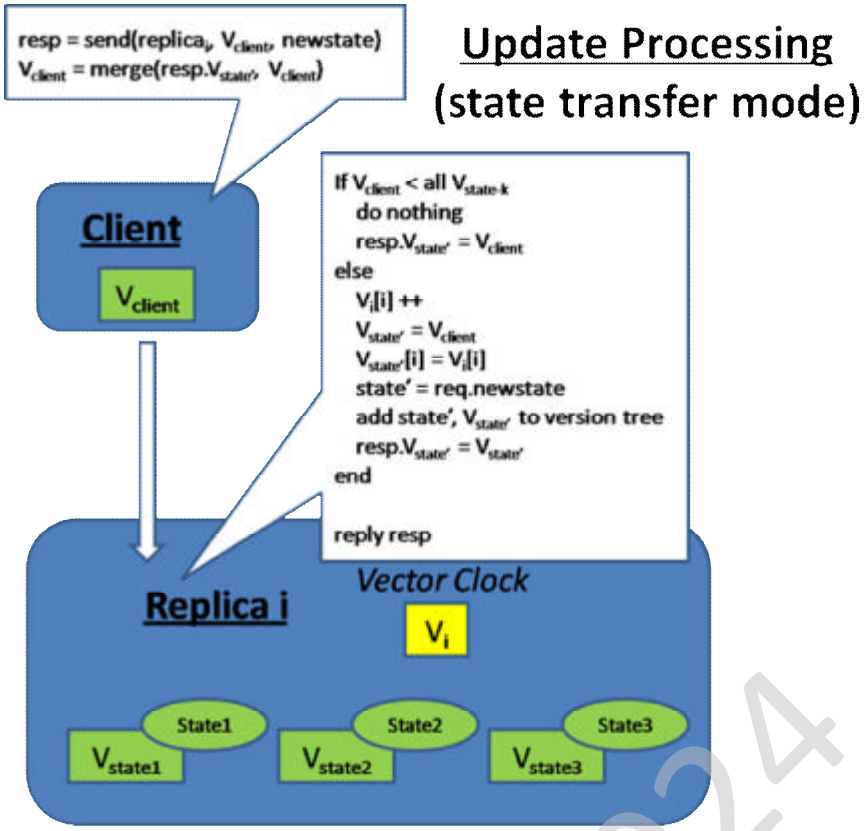


Рисунок 2.5 - Приклад ведення вектора годин

Ситуацій, які призводять до появи кількох версій записів у NoSQL базах даних, і способів усунення цих невідповідностей є вичерпним.

1) Виправлення одного документа кількома співробітниками

Ситуація: Кілька співробітників одночасно вносять зміни до одного документа, наприклад, пропозицій щодо розвитку компанії.

Спосіб усунення невідповідностей:

- Уніфікація: Керівник групи відповідає за вибір найбільш доцільних коригувань і їх об'єднання в один фінальний документ.

Це вимагає встановлення механізму рецензування або обговорення між співробітниками, щоб забезпечити узгодженість і уникнути конфліктів.

2) Оцінка документа кількома експертами

Ситуація: Кілька експертів одночасно оцінюють статтю, представлену на конференції.

Спосіб усунення невідповідностей

Вибір оцінки: Президент комісії може обрати остаточну оцінку на основі коментарів і пропозицій від інших експертів.

Такий підхід дозволяє врахувати різні точки зору і досягти консенсусу.

3) Спільна робота над документом

Ситуація: Кілька співробітників працюють над однією статтею і можуть бачити результати роботи один одного.

Спосіб усунення невідповідностей

Контроль з боку керівника: Керівник авторської групи має забезпечити узгодженість між різними версіями, обираючи найбільш релевантні зміни для остаточного варіанту.

Це може бути досягнуто шляхом регулярних зустрічей та обговорень.

4) Обговорення події в блозі

Ситуація: Користувачі обговорюють подію, де дані однієї фази обробки стають вхідними даними для наступної фази.

Спосіб усунення невідповідностей

Узгодженість на основі потенційної причинності: NoSQL системи можуть впроваджувати механізми, які забезпечують узгодженість даних шляхом обліку послідовності операцій.

Це може включати відстеження історії змін і автоматичне усунення конфліктів на основі часу або послідовності.

5) Серія пов'язаних записів

Ситуація: Додаток генерує серію записів, які зберігаються в різній часовій послідовності.

Спосіб усунення невідповідностей

Узгодженість на основі явної причинності: NoSQL системи можуть використовувати механізми, які відстежують і встановлюють зв'язки між записами, щоб усунути неузгодженості.

Це особливо важливо в сценаріях, де результати однієї операції є вхідними даними для наступної, забезпечуючи коректність і послідовність даних.

Ці приклади ілюструють, як складність спільної роботи та паралельного редагування може призвести до виникнення конфліктів в даних. В NoSQL базах даних механізми вирішення конфліктів, такі як уніфікація, вибір, узгодженість на основі причинності, дозволяють підтримувати цілісність даних, незважаючи на їх дистрибутивну природу. Це підкреслює важливість розуміння особливостей роботи з даними в умовах багатокористувацького середовища та необхідність впровадження ефективних методів управління змінами.

Щодо міркуваннь важливості узгодженості в системах NoSQL та способів вирішення конфліктів у ситуаціях з паралельною роботою дуже чіткі.

Значення узгодженості

Узгодженість є критично важливим елементом для стабільного функціонування системи, оскільки вона забезпечує, що всі користувачі працюють з актуальною та точною інформацією. Без узгодженості можуть виникати конфлікти, які ускладнюють або роблять неможливим виконання завдань. Це особливо важливо в контексті дистрибутивних систем, де дані можуть бути розподілені по різних вузлах.

Паралельна робота та вектор годин

У ситуаціях, коли кілька користувачів одночасно вносять зміни до одного запису, зростає ймовірність виникнення конфліктів. Використання механізму векторного годинника для відстеження версій запису є ефективним способом для вирішення цих конфліктів. Керівник групи може приймати рішення на основі інформації з вектора годин, щоб узгодити зміни, внесені різними користувачами.

Вікно незгоди

Знання про вікно незгоди — це важливий аспект у плануванні роботи з даними. Це вікно визначає час, протягом якого різні версії запису можуть існувати в системі до того, як конфлікти будуть вирішені. Керуючи цим вікном, системи можуть оптимізувати час відгуку та забезпечити кращу продуктивність.

Багатофазна обробка

У випадку багатофазної обробки, де вихід однієї фази є вхідними даними для наступної, відсутність невідповідностей є критично важливою. Знання затримки відповіді системи під час забезпечення суворої узгодженості дозволяє оцінити час виконання всього процесу. Це допомагає у плануванні та розподілі ресурсів, щоб забезпечити плавність виконання завдань.

Оцінка навантаження

Коли багато користувачів працюють з одним записом, кількість версій може значно зростати. Оцінка навантаження на користувачів стає важливою задачею. Ця оцінка включає:

- кількість версій запису;
- час обробки цих версій кожним користувачем;
- потенційні конфлікти, які можуть виникнути через паралельну роботу.

На основі цих оцінок можна створити рекомендації щодо максимальної кількості користувачів, які можуть одночасно брати участь у обговоренні. Це допоможе уникнути перевантаження системи і забезпечити більш ефективне використання ресурсів.

Висновки: управління узгодженістю, конфліктами та навантаженням у NoSQL системах є складним, але критично важливим завданням. Це вимагає чіткої стратегії, яка включає вектор годин, оцінку вікна незгоди та розуміння затримок у багатофазних процесах. Оптимізація цих аспектів може значно поліпшити продуктивність і стабільність системи.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання магістерської роботи буде реалізовано програмне забезпечення, що відображає роботу ІС з концепцією NoSQL.

У процесі написання магістерської роботи необхідно виконати наступний обсяг робіт:

а) провести аналіз існуючих аналогових систем з метою виявлення їх позитивних і негативних якостей. Результати аналізу будуть враховані при подальшій розробці;

б) вибрати та обґрунтувати спосіб побудови системи контролю за роботою технологічного обладнання виробництва в автоматизованому режимі. Розробити функціональні та структурні схеми системи;

в) розробити системне програмне забезпечення, яке дозволить реалізувати поставлене технічним завданням завдання. Будувати блок-схеми програмних алгоритмів і підпрограм;

г) організувати інтерфейс користувача з метою формування та відображення повідомлень про некоректні дії користувача та нестандартні ситуації;

г) надати рекомендації щодо організаційно-методичних заходів, які забезпечать введення системи в промислову експлуатацію та її подальше успішне функціонування;

г) виконати розрахунки для визначення економічної ефективності розробленої системи;

ж) розробити заходи з охорони праці під час впровадження та експлуатації системи, а також розробляє заходи цивільного захисту;

з) Сформувати висновки про обсяги виконаної роботи та отримані результати

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Аналіз і вибір типу СУБД та мови програмування

Етап аналізу передбачає вивчення системних вимог та існуючих проблем. Поняття «аналіз» конкретизується через терміни «аналіз вимог» (тобто вивчення системних вимог) і «об'єктно-орієнтований аналіз» (дослідження об'єктів предметної області). У процесі об'єктно-орієнтованого аналізу основна увага приділяється визначенню й опису об'єктів (або понять), що є важливими в межах предметної області.

Аналіз вимог може охоплювати опис процесів або сценаріїв використання програми, які можна навести у вигляді прикладів. Об'єктно-орієнтований аналіз концентрується на описі предметної області через класифікацію об'єктів. Декомпозиція предметної області полягає у виділенні понять, атрибутів і асоціацій, що мають значення для вирішення поставленої задачі. Підсумок аналізу представлений моделлю предметної області, яка ілюструється набором діаграм для демонстрації понять або об’єктів.

Найдоцільнішим є виділення процесів у прив'язці до наявних структурних елементів. Існуючі елементи побудовані на функціональній основі - виконання певної функції або створення готового продукту. Відповідно, розподіл процесів здійснюватиметься в рамках наявної системи управління. Опис процесів у домені може бути виконаний у вигляді прецедентних структурованих словесних описів. Прецедент - це набір взаємопов'язаних сценаріїв успішного і невдалого виконання, які демонструють, як користувач системи вирішує певне завдання. Запити, сформовані на основі побажань зацікавлених сторін [11], подано нижче.

Функціональний. Функції з назвою «керування» передбачають наявність опцій: додати, змінити, видалити тощо.

Опис інформаційної системи

У магістерській роботі реалізовано інформаційну систему, що зберігає дані про людські ресурси, освіту співробітників, їхній останній досвід, графіки роботи, управління відпустками та поточні розподілені проекти. Також буде створено адміністративну панель для управління всією інформацією про співробітників. Панель міститиме статистику щодо рівня зайнятості працівників різних установ і середній показник плинності кадрів для кожної установи.

Проєкт буде розроблено на базі Node Express, веб-додатка для Node.js, з інтерфейсом, побудованим за допомогою HTML, Bootstrap, CSS і JS.

Система складається з основних структурних модулів:

- окремі облікові записи для всіх адміністраторів і співробітників;

- різні види даних і контрольований доступ до них на основі привілеїв користувачів;

- реєстрація нових співробітників і адміністраторів;

- тайм-менеджмент для контролю робочого часу співробітників;

- управління заробітною платою;

- ведення обліку загального стажу та практичного досвіду співробітників;

- управління поточними розподіленими проєктами в межах організації.

У системі вибираються такі групи користувачів:

- **Адміністратор.** Має повний доступ до системи, включаючи можливість реєструвати нових співробітників, призначати привілеї іншим користувачам, переглядати та коригувати відвідуваність, керувати зарплатою, видаляти записи або профілі співробітників, призначати та перепризначати проєкти, а також схвалювати або відхиляти заявки на відпустку.

- **Персонал.** Може відзначати свій робочий графік, переглядати власну історію, поточні зарплати, профіль (включаючи освіту та досвід роботи), бачити всі свої проєкти та колег, які працюють над тими ж проєктами, подавати заявки на відпустку та переглядати їх статус.

- **Лідер проєкту.** Має доступ до навичок співробітників у проєкті, яким керує, може оцінювати продуктивність співробітників і бачити список всіх учасників проєкту.

- **Бухгалтерський працівник.** Може створювати платіжні відомості для співробітників, призначати бонуси, встановлювати та коригувати зарплати, підвищувати заробітну плату, а також надсилати зарплатні відомості електронною поштою кожному працівнику.

Розробка архітектури інформаційної системи

Архітектура програми визначає її основні компоненти, функції та способи взаємодії між ними. Створений додаток реалізує клієнт-серверну модель, підтримує ідентифікацію та авторизацію користувачів, а також забезпечує зберігання даних. Основні запити програми передаються по мережі, що вимагає наявності як клієнтської, так і серверної частини.

Функціонування клієнт-серверної програми відбувається наступним чином:

- клієнт генерує та надсилає запит на сервер;

- сервер виконує необхідну обробку даних та звертається до бази даних;

- сервер отримує результат від бази даних, обробляє його і надсилає відповідь клієнту;

- клієнт приймає результат, відображає його користувачу та очікує на подальші дії.

Цей процес повторюється, доки користувач не завершить роботу з сервером. Клієнт-серверна архітектура була обрана для цієї системи (рисунок. 3.2), оскільки вона забезпечує масштабованість і дозволяє великій кількості користувачів отримувати доступ до системи. Важливою перевагою є також можливість віддаленого доступу до системи (22).

Проєкт на основі клієнт-серверної архітектури складається з трьох основних частин:

- зв'язок з базою даних – обробка запитів і управління збереженням даних;
- представлення даних клієнту – інтерфейс, що дозволяє користувачу переглядати та взаємодіяти з інформацією;
- бізнес-логіка – частина, яка обробляє запити користувачів та надає їм саме ті дані, яких вони потребують.

Обробка та зберігання даних здійснюються на сервері, а відображення інформації та запити на її зміну – на клієнті.

У процесі проєктування основна увага зосереджена на концептуальному рішенні (у вигляді програмного чи апаратного забезпечення), яке відповідає основним вимогам системи. На цьому етапі проєктування можуть бути описані програмні об’єкти або схеми баз даних.

Концепцію проєктування можна розділити на два основні напрямки: об’єктно-орієнтоване проєктування та проєктування бази даних. У процесі об’єктно-орієнтованого проєктування визначаються програмні об’єкти та способи їх взаємодії, щоб забезпечити виконання системних вимог.

Об’єктно-орієнтоване проєктування передбачає визначення об’єктів, їхніх обов’язків і методів взаємодії. Для ілюстрації цих зв’язків часто використовують діаграми послідовності – тип діаграм взаємодії UML, які демонструють потоки повідомлень між програмними об’єктами та виклики методів. Поряд із динамічним відображенням зв’язків між об’єктами, представленим на діаграмі взаємодії, корисним також є створення фрагмента системи у вигляді діаграми класів, яка описує проєктувальні класи системи.

3.2 Розробка структурної схеми

Структурна схема сайту представляє собою сукупність об'єктів та частин сайту, а також їхні взаємозв'язки. Метою створення такої схеми є наочне відображення основних складових частин сайту, його блоків, вузлів і зв’язків між ними. Це допомагає чітко уявити загальну архітектуру сайту та зрозуміти, як

окремі елементи взаємодіють для досягнення функціональних вимог.

Структурна схема розробленої системи зображена на рисунку 3.1. На ній показано структуру додатку.

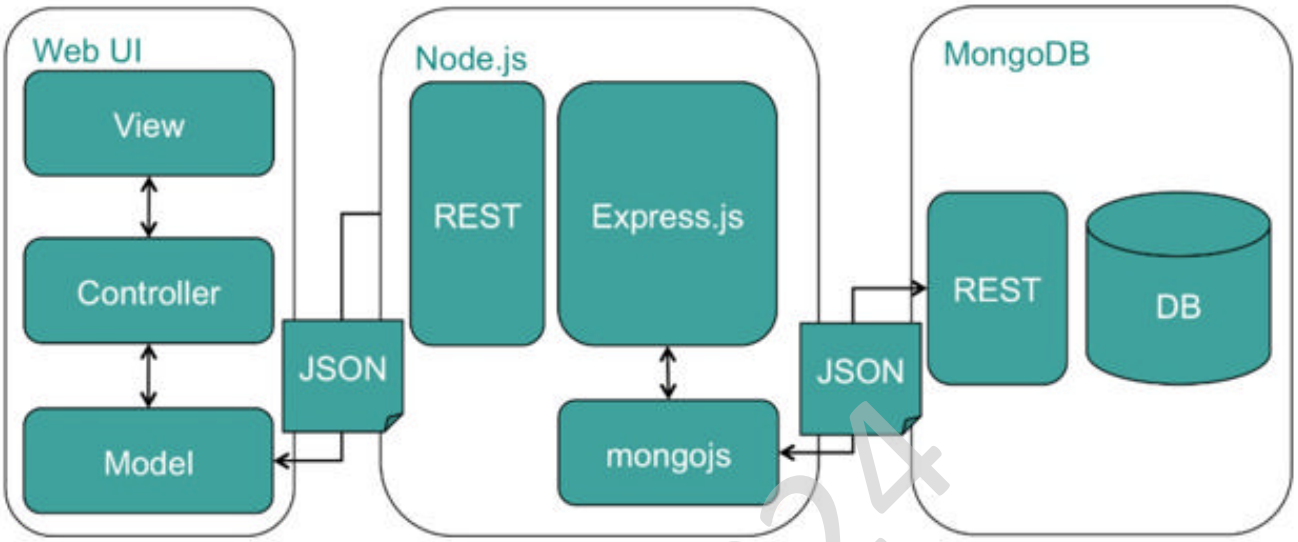


Рисунок 3.1 – Структурна схема

Структурна схема системи

Структурна схема системи відображає, як кожен компонент взаємодіє з іншими, створюючи єдину систему, що забезпечує роботу веб-додатка із серверною частиною, базою даних та ключовими функціональними блоками. Нижче наведено короткий опис кожного компонента та їхню взаємодію:

WEB UI (користувацький інтерфейс):

- забезпечує візуальну взаємодію з користувачем;
- взаємодіє із сервером через HTTP-запити, щоб отримати або оновити дані.

NodeJS:

- використовується як середовище виконання серверної частини;
- обробляє HTTP-запити від WEB UI та керує взаємодією з іншими компонентами системи.

Express.js:

- веб-фреймворк для Node.js, який спрощує створення API;
- виконує маршрутизацію та обробляє HTTP-запити.

Controller:

- відповідає за бізнес-логіку, опрацьовуючи дані, отримані від Express.js;
- керує взаємодією між WEB UI та базою даних.

MongoDB:

- нереляційна база даних, де зберігаються всі дані;
- Controller звертається до MongoDB для збереження, читання та оновлення інформації.

REST:

- використовується для створення API, що забезпечує взаємодію між WEB UI та серверною частиною системи;
- встановлює стандарти для обміну даними між компонентами через HTTP-протокол.

JSON:

- формат, що використовується для передачі даних між компонентами;
- дані, що передаються між WEB UI, NodeJS та базою даних, можуть бути представлені у форматі JSON.

Ця структура забезпечує ефективну роботу системи, де кожен компонент виконує свою роль у забезпеченні функціональності та безперервної роботи веб-додатка.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.1.

Функціональна схема системи

Функціональна схема системи допомагає розібрати ролі кожного

компонента та його взаємодію для забезпечення цілісної функціональності системи. Розглянемо основні блоки схеми:

UseInterface (Користувацький інтерфейс)

Цей блок відповідає за інтерфейс веб-додатка, надаючи користувачам доступ до функцій системи через зручний та естетичний дизайн.

Складові блоку:

HTML: Мова розмітки, що визначає структуру веб-сторінки та елементи, які будуть відображені;

JavaScript (JS): Відповідає за динамічну взаємодію на сторінці, обробку подій, зміну контенту та асинхронні запити через AJAX;

CSS: Стилізує елементи інтерфейсу, забезпечуючи привабливий та адаптивний дизайн;

Fonts (Шрифти): Підвищують зручність читання та загальний стиль інтерфейсу;

Images (Зображення): Використовуються для ілюстрацій, фонових елементів та логотипів, підвищуючи візуальну привабливість.

Request Management (Управління запитами)

Цей блок описує серверну частину веб-додатка, що використовує Express.js, менеджер маршрутів, контролери, API та бібліотеку Mongoose для взаємодії з базою даних MongoDB.

Основні компоненти блоку

Express.js: Фреймворк для створення веб-додатків та API на Node.js. Виконує маршрутизацію та обробку HTTP-запитів.

Route Manager (Менеджер маршрутів): Керує маршрутами в Express.js, обробляючи запити та реєструючи шляхи.

Controllers (Контролери): Обробляють бізнес-логіку, виконують запити та взаємодіють з моделями для роботи з базою даних.

API (Інтерфейс програмування додатків): Визначає стандарти обміну даними між компонентами, забезпечуючи взаємодію клієнтської та серверної

сторін.

Mongoose: Бібліотека для взаємодії з MongoDB, створення схем даних, валідації та операцій CRUD.

Models (Моделі): Описують структуру даних у базі, використовуючи методи Mongoose для обробки запитів до MongoDB.

Module (Модуль)

Цей блок складається з різних функціональних компонентів, таких як Email, ExcelExport, Auth і Tasks, які виконують специфічні завдання в системі.

Email (Електронна пошта): Надсилає повідомлення електронною поштою, забезпечує сповіщення користувачів.

ExcelExport (Експорт в Excel): Генерує файли у форматі Excel, надаючи зручне представлення даних у табличному форматі.

Auth (Автентифікація та авторизація): Перевіряє ідентифікацію користувача та надає доступ до відповідних ресурсів.

Tasks (Завдання): Управляє завданнями, забезпечуючи можливість створення, редагування та відстеження статусу виконання завдань.

Ця функціональна схема чітко відображає ролі кожного компонента, забезпечуючи повну інтеграцію всіх елементів у рамках цілісної системи. Кожен блок виконує конкретні функції, необхідні для ефективної роботи веб-додатка.

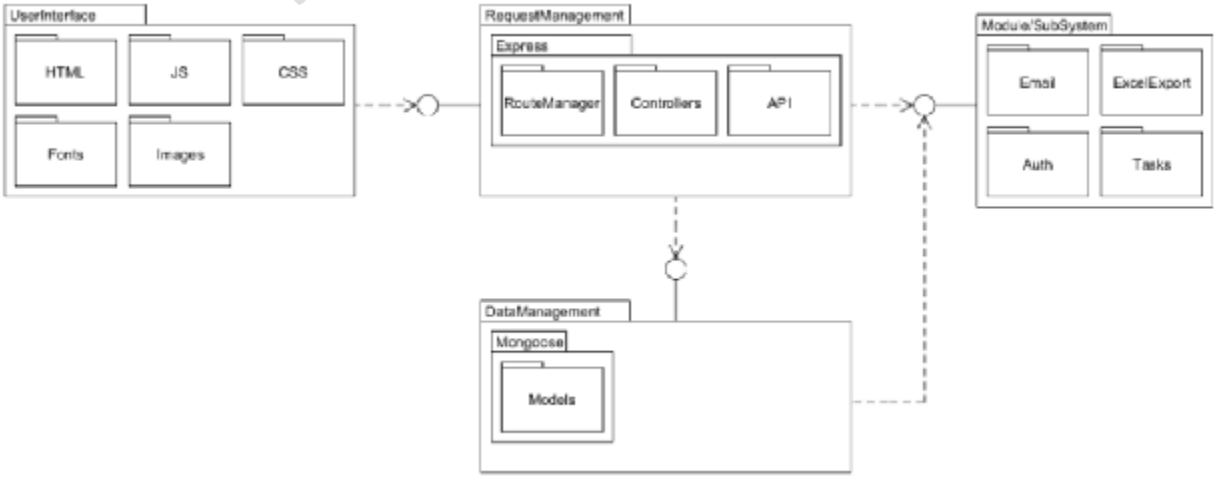


Рисунок 3.2 – Функціональна схема

3.4 Розробка діаграми процесів

Діаграми процесів, як і діаграми послідовності, належать до діаграм взаємодії. Вони ілюструють взаємодію об’єктів у форматі графіка чи сітки, відображають потік подій і зосереджують увагу на зв’язках між об’єктами.

Моделювання виконання операції виконується наступним чином:

1) Ідентифікація параметрів, повернених значень та інших об'єктів, видимих для операції.

2) Якщо операція тривіальна, візуалізуйте її реалізацію безпосередньо в коді, який можна розмістити на задньому плані моделі або явно візуалізувати в анотації.

3) Якщо операція є алгоритмічно складною, ми змоделюємо її виконання за допомогою діаграми діяльності.

4) Якщо операція потребує великого обсягу детального проектування, розгляньте можливість спільної реалізації. У майбутньому ви зможете розширити структурні та поведінкові компоненти співпраці за допомогою діаграм класу та взаємодії відповідно. Відношення - це зв'язок між двома примірниками класів, які визначають певну форму руху та видимості між ними. Посилання є прикладом асоціації. Повідомлення, що передаються між об’єктами, представлені як імена цих повідомлень над лініями з’єднання зі стрілками. Над однією лінією зв’язку можна перерахувати будь-яку кількість повідомлень. Щоб показати послідовність повідомлень у поточному потоці керування, поруч із повідомленням вказується порядковий номер.

Діаграми співпраці, як і діаграми послідовності, можна використовувати для відображення взаємодії як між об’єктами домену, так і між об’єктами програмного забезпечення.

Існують різні способи організації взаємодії між клієнтом і сервером. Одними з найпопулярніших технологій є WebSocket і AJAX. Суть технології AJAX полягає в зміні вмісту завантаженої веб-сторінки без повного перезавантаження, завдяки чому досягається висока динамічність сайтів.

Технологія заснована на обміні даними та навчанні певних компонентів за потреби. AJAX з'явився в 1998 році і тому підтримується старими версіями різних браузерів [9].

WebSocket - це стандартна технологія HTML5, яка дозволяє встановлювати повнодуплексне TCP-з'єднання між сервером і веб-браузером (клієнтом). Ця технологія покликана вирішити проблему зняття обмежень на обмін даними між браузерами і серверами. Браузер повинен мати підтримку WebSocket.

Розроблено діаграму процесів, щоб розглянути деякі передумови для вибору технічних або інших інструментів і технологій, а також дизайну програми. Визначимо кілька варіантів використання за схемою.

Діаграму процесів зображено на рисунку 3.3.

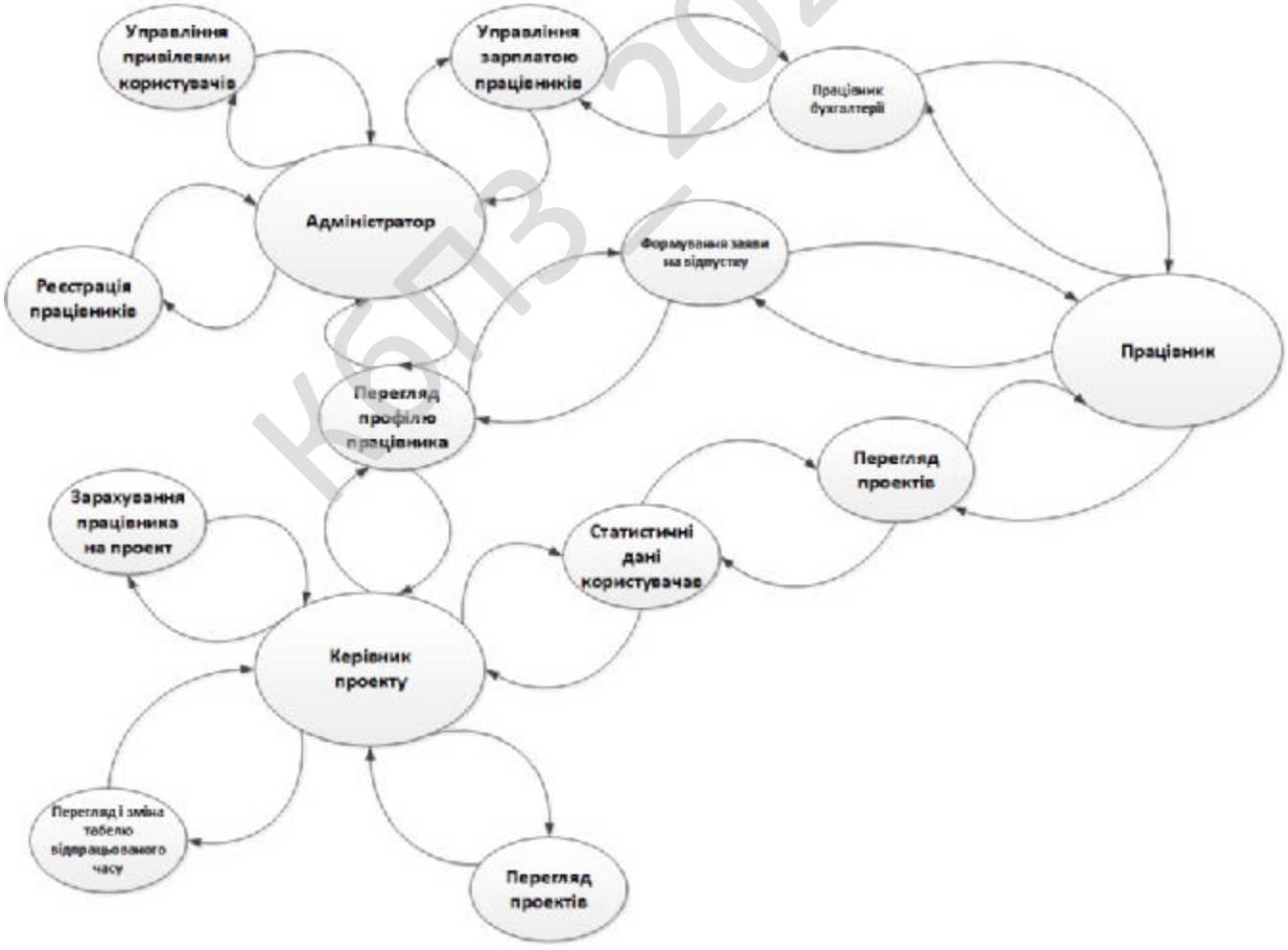


Рисунок 3.3 – Діаграма процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ

Опис технологій для створення системи

Аналіз сучасних веб-технологій показав, що платформа Node.js є ефективним вибором для створення системи. Node.js — це середовище для JavaScript, яке базується на движку Chrome V8. Воно застосовує неблокуючу, подієво-орієнтовану модель введення-виведення, що сприяє легкості та високій ефективності. Модель Node.js (рисунок. 4.1) кардинально відрізняється від традиційних серверних платформ, де масштабованість досягається завдяки багатопоточності. Завдяки архітектурі, заснованій на подіях, знижується використання пам'яті та підвищується пропускна здатність.

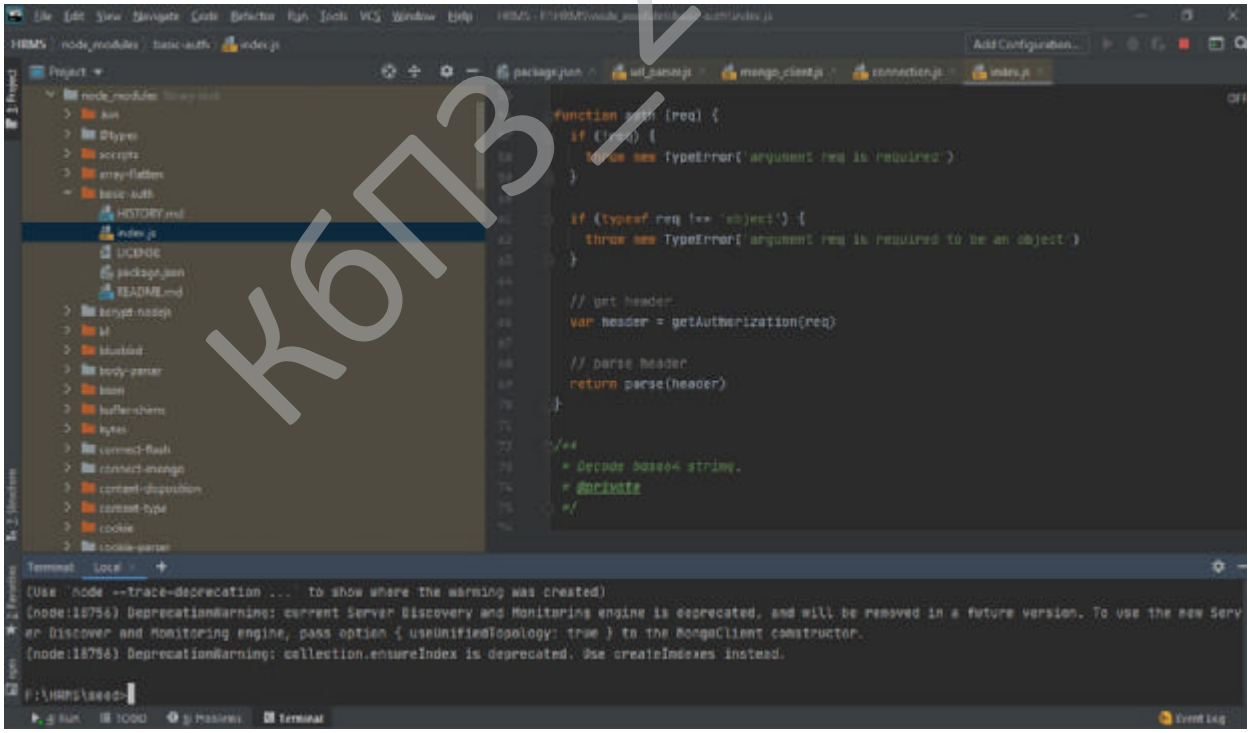


Рисунок 4.1 - Середовище розробки з Node.js

У Node.js під час запуску створюється єдиний програмний потік, у якому й виконується весь код. Це означає, що одночасне виконання кількох частин додатка неможливе. Однак “паралельність” досягається за рахунок асинхронної моделі, побудованої на основі циклу подій (event-loop), який обробляє завдання в порядку черги. На рисунку 4.2. зображена діаграма розгортання.

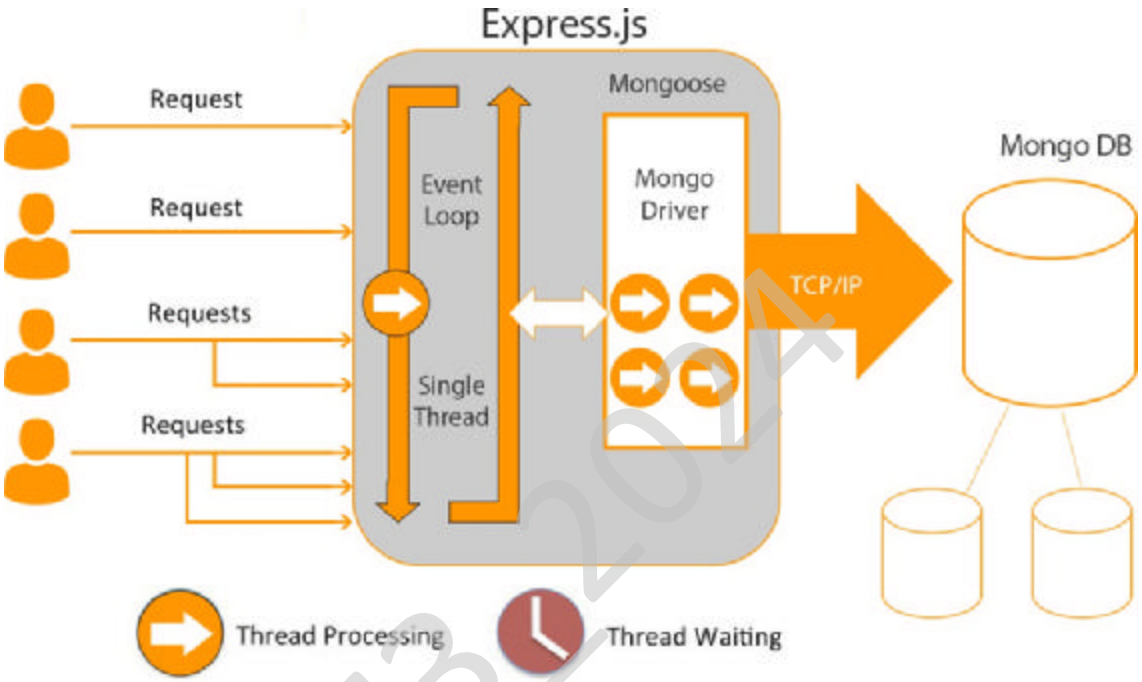


Рисунок 4.2 - Діаграма розгортання, включаючи Node.js Express, MongoDB

Базу даних MongoDB було обрано як СУБД з кількох причин:

- можливість роботи з неструктурованими даними;
- слабкий або відсутній зв'язок між колекціями;
- зберігання даних не потребує високої продуктивності.

Кожен екземпляр даних зберігається у впорядкованій колекції в базі (рисунок. 4.3).

Загальна архітектура веб-додатка побудована за принципом клієнт-серверної моделі. Клієнт надсилає запити серверу для виконання певних операцій або отримання інформації, сервер обробляє ці запити та повертає відповіді.

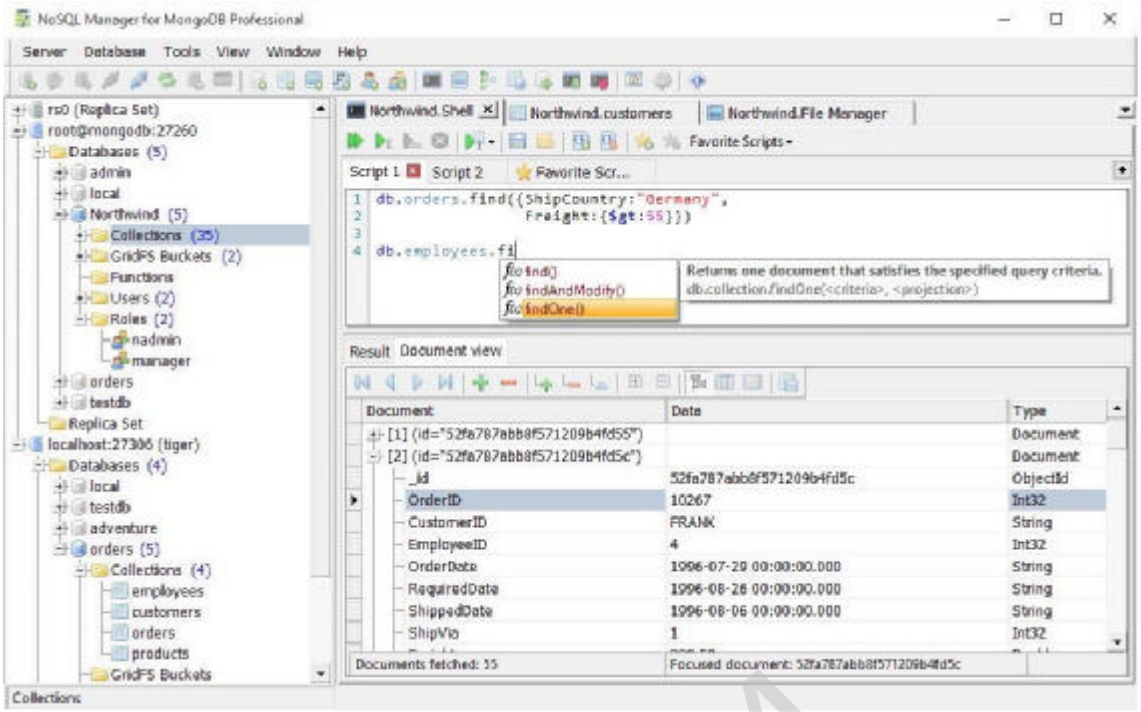


Рисунок 4.3 - MongoDB Manager

Серверна частина системи управління кадровими процесами для обробки запитів повинна включати три основні компоненти, що виконують такі функції: обробку запиту, попередню обробку отриманих даних та маршрутизацію. Компонент, відповідальний за обробку запиту, створює об’єкт запиту і передає його компоненту попередньої обробки даних, після чого звільняє створений екземпляр для обробки запиту.

У Node.js, функції, які забезпечують проміжну обробку даних у «Express», називаються «проміжним програмним забезпеченням» (middleware). Вони організовуються в ланцюжки, і після проходження всього ланцюжка обробки управління передається компоненту маршрутизації. Звільнення екземпляра обробки запиту і передача управління маршрутизаційному компоненту відбувається за допомогою функції зворотного виклику.

Використання Node.js разом із документо-орієнтованою базою даних MongoDB для побудови серверної частини дозволяє створювати високошвидкісні програми, які підтримують велику кількість одночасних підключень і вимагають невеликого обсягу оперативної пам’яті.

Програмна реалізація системи

Для розробки проєкту було обрано мову JavaScript разом із платформою Node.js. Це безкоштовна платформа, яка дозволяє створювати програми на JavaScript і включає численні корисні модулі, що спрощує написання коду без потреби починати «з нуля».

Node.js складається із середовища виконання та бібліотек. У розробці застосовувалися такі бібліотеки JavaScript:

Express.js — фреймворк, який спрощує створення веб-додатків [12];

Socket.io — бібліотека для реалізації Web-Socket з'єднань у веб-додатках, що дозволяє обмін даними в реальному часі. Вона складається з двох частин: клієнтської, яка працює в браузері, і серверної для Node.js. Обидві частини мають схожий API [14];

WebStorm — редактор коду з підсвічуванням синтаксису, підтримкою тем і гарячих клавіш, що спрощує написання JavaScript коду, який можна легко інтегрувати у веб-сторінки [10];

Monk.js — обгортка для MongoDB, що дозволяє зручно взаємодіяти з базою даних [13].

Для забезпечення реального часу між клієнтом і сервером було обрано WebSocket. Цей тип з'єднання надає обом сторонам — клієнту та серверу — рівні права, дозволяючи перевірку з'єднання за допомогою PING і PONG кадрів. Ініціатор перевірки надсилає кадр PING із довільним вмістом, і приймач повинен відповісти кадром PONG із тим самим вмістом протягом визначеного часу.

Для обміну даними між клієнтською стороною та сервером розроблені і реалізовані обробники подій. Список обробників подій для клієнтської сторони наведено на рисунку 4.4.

```
socket
.on('message', function (username, message) {/*…*/})
.on('leave', function(username, userList) {/*…*/})
.on('join', function(username, userList) {/*…*/})
.on('connect', function() {/*…*/})
.on('disconnect', function() {/*…*/})
.on('change_code', function(code){ /*…*/})
```

Рисунок 4.4 - Список обробників подій на стороні клієнта

Обробники подій на стороні клієнта в цьому веб-додатку можна описати так:

- повідомлення — спрацьовує, коли користувач надсилає повідомлення в чаті;

- вихід — активується, коли користувач закриває всі вкладки браузера або виходить із системи;

- приєднання — виконується при підключенні нового учасника до системи;

- connect — подія, що сигналізує про активне з'єднання з сервером;

- disconnect — подія, яка спрацьовує при втраті з'єднання з сервером;

- change — відбувається при зміні доступу до проекту;

- change lang — активується при зміні мови інтерфейсу;

- зміна прав — спрацьовує, коли змінюються права доступу конкретного учасника.

Кожен із цих обробників забезпечує гнучку взаємодію клієнтської сторони з сервером і підтримку відповідних змін у реальному часі.

Також були розроблені та реалізовані обробники подій на стороні сервера, які показані на рисунку 4.5.

Рисунок 4.5 - Список обробників подій на стороні сервера

Важливо розуміти, що хоча назви деяких подій на клієнті й сервері можуть збігатися, реалізація їхньої логіки принципово відрізняється. Наприклад, при обробці події **повідомлення** на клієнті отримане від сервера повідомлення просто відображається в чаті, тоді як на сервері ця подія обробляється через розсилку повідомлення всім учасникам чату.

При проєктуванні системи було прийнято рішення використовувати документо-орієнтовану NoSQL СУБД. Серед доступних варіантів, таких як CouchDB, Couchbase, MarkLogic, MongoDB, eXist тощо, важливо було вибрати систему, яка б забезпечувала зручну інтеграцію з веб-додатком. MongoDB є найбільш популярною для платформи Node.js завдяки своїй розширюваності, високій продуктивності, відсутності жорсткої схеми і відкритому вихідному коду. Вона зберігає дані у форматі BSON (двоїчний формат JSON), що робить її гнучкою та зручною для веб-додатків.

У MongoDB база даних складається з колекцій, які, зазвичай, представляють певну сутність предметної галузі. Кожна колекція містить **документи** (далі - записи), що є JSON-об'єктами з обов'язковим ідентифікатором **ObjectId**. MongoDB автоматично генерує цей ідентифікатор, який є унікальним, включно з глобальною унікальністюянні з реляційною СУБД, колекція аналогічна таблиці, а запис - рядку таблиці.

Типи даних, які можуть зберігатися в записах MongoDB, включають рядки, масиви (структури даних з доступом за індексом), логічні значення, дати, цілі числа, null, а також об’єкти (структури даних, що дозволяють доступ за ключем).

Оскільки MongoDB не підтримує реляційну цілісність, забезпечення цілісності даних виконується на рівні серверного коду. Для цього використовується ODM (Object Data Modeling) бібліотека **Mongoose**, яка спрощує взаємодію з MongoDB та підтримує обробку зв'язків між даними.

Рисунок 4.6 - Модель документів бази MongoDB

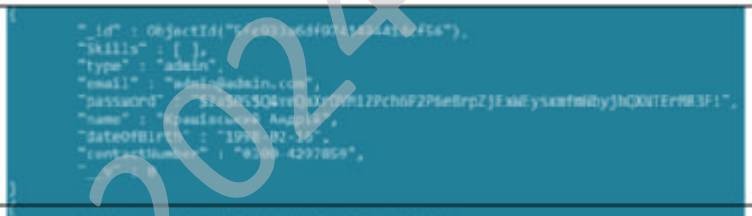
У колекції зберігається кілька ключових елементів, серед яких:

- **ідентифікатор користувача**, який створив деталь;
- **ідентифікатори проєктів**;
- деталі поточного отримання реквізитів платежу;
- **ідентифікатор запису** в колекції.

Вирішено, що нова вкладка документа розглядається як сесія того самого користувача. При цьому з'єднання з клієнтом припиняється тільки після закриття всіх вкладок, відкритих цим користувачем.

Складність полягає в тому, що згідно з WebSocket-стандартом кожна нова вкладка браузера відкриває окреме повноцінне з'єднання між клієнтом і сервером. Для відстеження всіх активних WebSocket-з'єднань користувача для конкретного проєкту створюється таблиця підключених користувачів. Це дозволяє коректно зберігати стан користувача, відстежуючи всі його активні з'єднання. Структура збережених даних описана в таблиці 4.1.

Таблиця 4.1 - Приклад списку формування списку працівників

Ідентифікатор користувача	Роль	Реалізований Json-документ
5fc933a6df97 4149441dcf56	Адміністратор	
5fcbd91313dbc f3e443ebe08	Бухгалтер	

Щоб мати чітке уявлення про зв’язки між сутностями, було розроблено схему бази даних з використанням псевдозв’язків (рисунок. 4.7) за нотацією Гордона Евереста [21]. Важливо зазначити, що імена колекцій і псевдозв’язків у схемі можуть мати дещо інше значення, ніж у реляційних базах даних, оскільки тут можливе використання вкладених об'єктів. Наприклад, колекція **product_type** демонструє один із таких випадків.

Типи псевдозв’язків документа з іншими сутностями, з урахуванням неструктурованості даних, такі:

HasOne: має один тип документа.

HasOne: має одну мову.

ManyToMany: належить до одного або кількох продуктів (**BelongsTo**), оскільки є загальним. З іншого боку, один продукт може мати багато документів різними мовами (**HasMany**).

HasMany: містить багато змін.

BelongsTo: належить до одного завантаження, електронної пошти або замовлення.

Ця структура зв'язків дозволяє більш гнучко відображати стосунки між даними, враховуючи неструктурованість і вкладеність документів у NoSQL базах даних, таких як MongoDB.

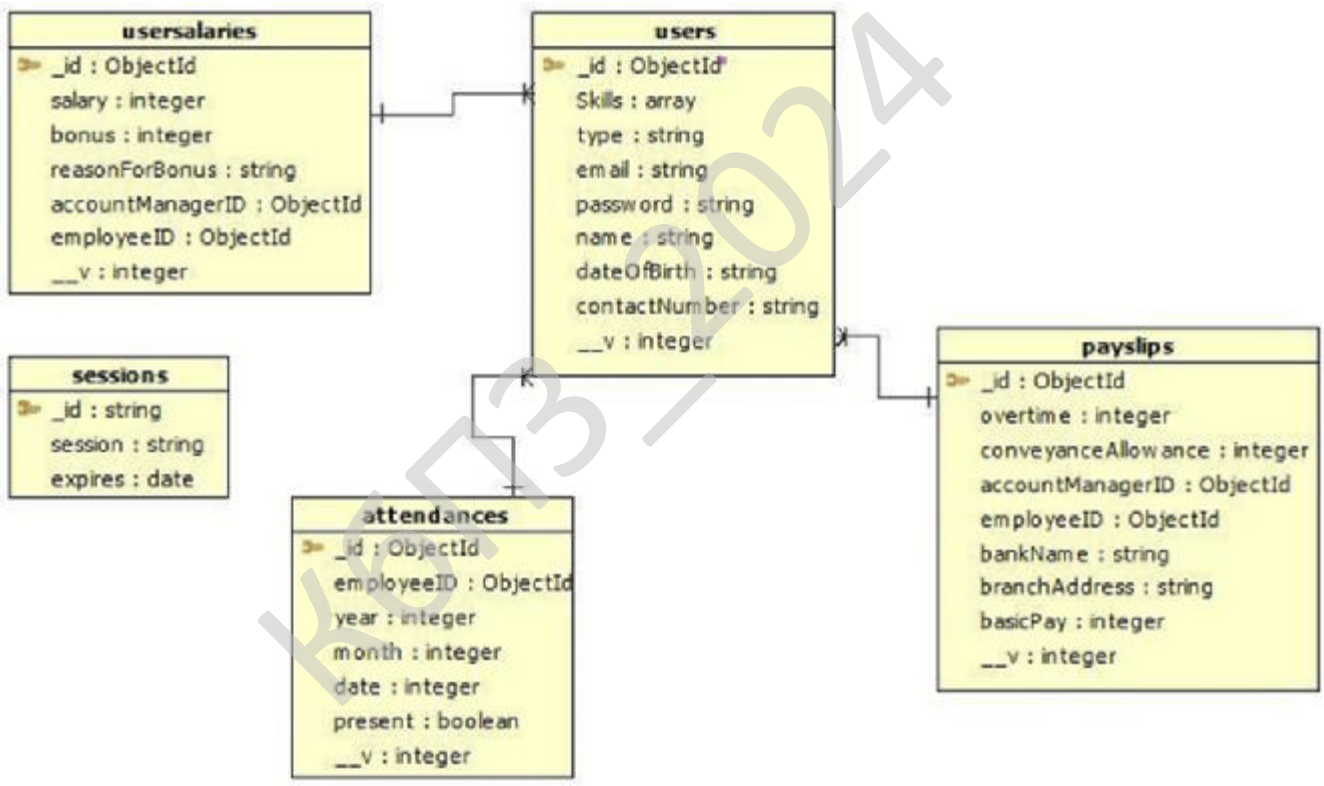


Рисунок 4.7 - Схема бази даних з позначенням псевдо-зв'язків

У контексті проектування бази даних MongoDB використовуються такі ключові поняття для опису функцій дизайну схеми:

Вбудовування (embedding) - це збереження об'єкта однієї сутності всередині іншої. Це корисно, коли об'єкт має тісно пов'язану інформацію, яку потрібно зберігати разом із основною сутністю;

Посилання (referencing) - використання унікального ідентифікатора **ObjectId** для зовнішньої сутності.

У MongoDB посилання дозволяють зв'язувати документи між різними колекціями.

Інструмент **Mongoose** автоматично генерує **ObjectId** під час вставки вкладеного об'єкта. Наприклад, при додаванні продукту до типу товару.

При проектуванні схеми бази даних у MongoDB визначено кілька конструктивних особливостей:

- **гнучкість**: Використання **композитних псевдоключів** (наприклад, комбінованих ідентифікаторів) знижує гнучкість роботи з даними, оскільки це ускладнює подальше масштабування та модифікацію структури;
- **запити**: Якщо зберігається лише **ідентифікатор об'єкта зовнішньої сутності**, це вимагатиме додаткового запиту для отримання всіх даних цієї сутності;
- **вкладеність об'єкта сутності** (embedding) має сенс, коли об'єкт, який вбудовує іншу сутність, завжди пов'язаний із цією сутністю, і їх треба часто використовувати разом. Це відображає зв'язок **"один до багатьох"** в реляційній схемі. Наприклад, один товар може мати багато відгуків, і ці відгуки можуть зберігатися всередині самого товару;
- **посилання** (referencing) дають більшу гнучкість при частих змінах об'єктів, оскільки замість того, щоб оновлювати вкладені об'єкти в кожному батьківському документі, можна просто змінити один документ. Це відображає зв'язок "багато до багатьох" у реляційній схемі. Наприклад, один користувач може мати багато проектів, і для кожного проекту створюється окремий запис, що містить посилання на користувача.

Атрибути основної сутності Users наведені в таблиці 4.2, а структура проекту інформаційної системи зберігання даних показана на рисунку 4.8.

Таблиця 4.2 - Атрибути основної сутності Users

Найменування атрибуту	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор Users
Skills	array	Практичні навички
type	string	Тип
email	string	email
password	string	Пароль
name	string	Прізвище, ім’я, по батькові
dateOfBirth	string	Дата народження
contactNumber	string	Контактний номер телефону
__v	date	Дата нагадування актуальності
department	string	Підрозділ
designation	string	Посада

Для реалізації запиту пошуку за атрибутами **ім'я**, **контактний телефон**, **відділ** та **ярлик** в MongoDB, вам потрібно створити запит, який шукає користувача з відповідними даними.

У **userSchema** зберігаються основні атрибути користувача: ім'я, телефон, відділ та ярлик.

У функції **searchUser** створюється об'єкт **searchConditions**, в який додаються умови пошуку (якщо відповідні атрибути передані в запит).

Запит **User.find(searchConditions)** виконується для пошуку користувачів, які задовольняють умови пошуку.

Результати пошуку повертаються у вигляді масиву об'єктів користувачів.

Цей підхід дозволяє гнучко здійснювати пошук за будь-якими з атрибутів, при цьому, якщо один з атрибутів не передано, він просто не буде враховуватися у пошукових умовах.

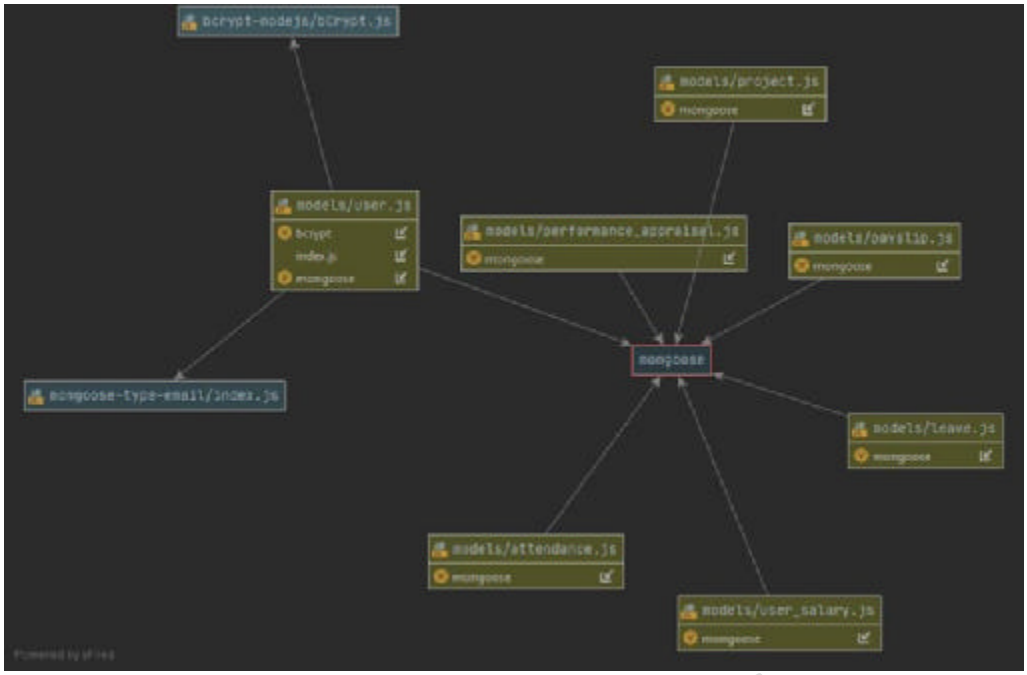


Рисунок 4.8 - Структура проекту інформаційної системи для зберігання даних

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Алгоритм роботи основної програми, зокрема процес реєстрації та активації користувача, можна описати наступними кроками:

Алгоритм реєстрації та активації користувача:

- **вхід на сайт**. Користувач потрапляє на головну сторінку сайту;
- **вибір між реєстрацією та авторизацією**;
- користувач вибирає опцію **реєстрації** (якщо в нього немає аккаунта) або **авторизації** (якщо аккаунт вже існує).

Реєстрація нового аккаунта

На сторінці реєстрації користувач вводить свої дані:

- фотографія (завантажується файл);
- ім’я;
- поштова адреса;
- пароль.

Після введення даних система перевіряє:

- чи не порожні поля;
- чи завантажено зображення;
- якщо є незаповнені поля або відсутня фотографія — виводиться помилка.

Створення запису в базі даних

Якщо всі поля заповнені правильно:

- створюється **перший запис** в таблиці користувачів (що містить інформацію про користувача, наприклад, ім'я, email, пароль);
- створюється **другий запис** в таблиці акаунтів на активацію, що містить:
- ідентифікатор користувача;
- код активації (унікальний для кожного користувача).

Надсилання листа з кодом активації:

- система надсилає листа на вказану поштову адресу користувача;
- лист містить посилання з кодом активації, який необхідно ввести на наступному етапі для активації аккаунта;
- користувач отримує повідомлення, що лист надіслано на вказану адресу.

Перехід по посиланню з кодом активації:

- користувач переходить за посиланням, що містить код активації.

Перевірка коду активації:

- система перевіряє код активації, який міститься в URL;
- якщо код активації збігається з тим, що зберігається в таблиці акаунтів на активацію;
- видаляється запис про неактивований аккаунт;
- статус аккаунта змінюється на **активовано** в таблиці користувачів.

Перенаправлення на сторінку авторизації;

- користувач перенаправляється на сторінку **авторизації**.

- користувач може ввести свої дані для входу.

Авторизація та доступ до сайту:

- користувач вводить ім’я користувача та пароль для авторизації;
- якщо дані правильні, користувач отримує доступ до закритої частини сайту та може почати роботу.

Блок-схема алгоритму:

- користувач переходить на сайт → Вибір між реєстрацією та авторизацією.
- реєстрація;
- введення даних (фото, ім’я, email, пароль);
- перевірка заповнення полів → помилка або успіх;
- створення запису в базі даних;
- відправлення листа з кодом активації;
- користувач переходить по посиланню → перевірка коду активації;
- якщо код правильний → активуємо аккаунт;
- перенаправлення на сторінку авторизації;
- авторизація користувача → доступ до сайту;

Цей алгоритм дозволяє забезпечити безпечний процес реєстрації та активації користувачів для доступу до закритих частин веб-сайту.

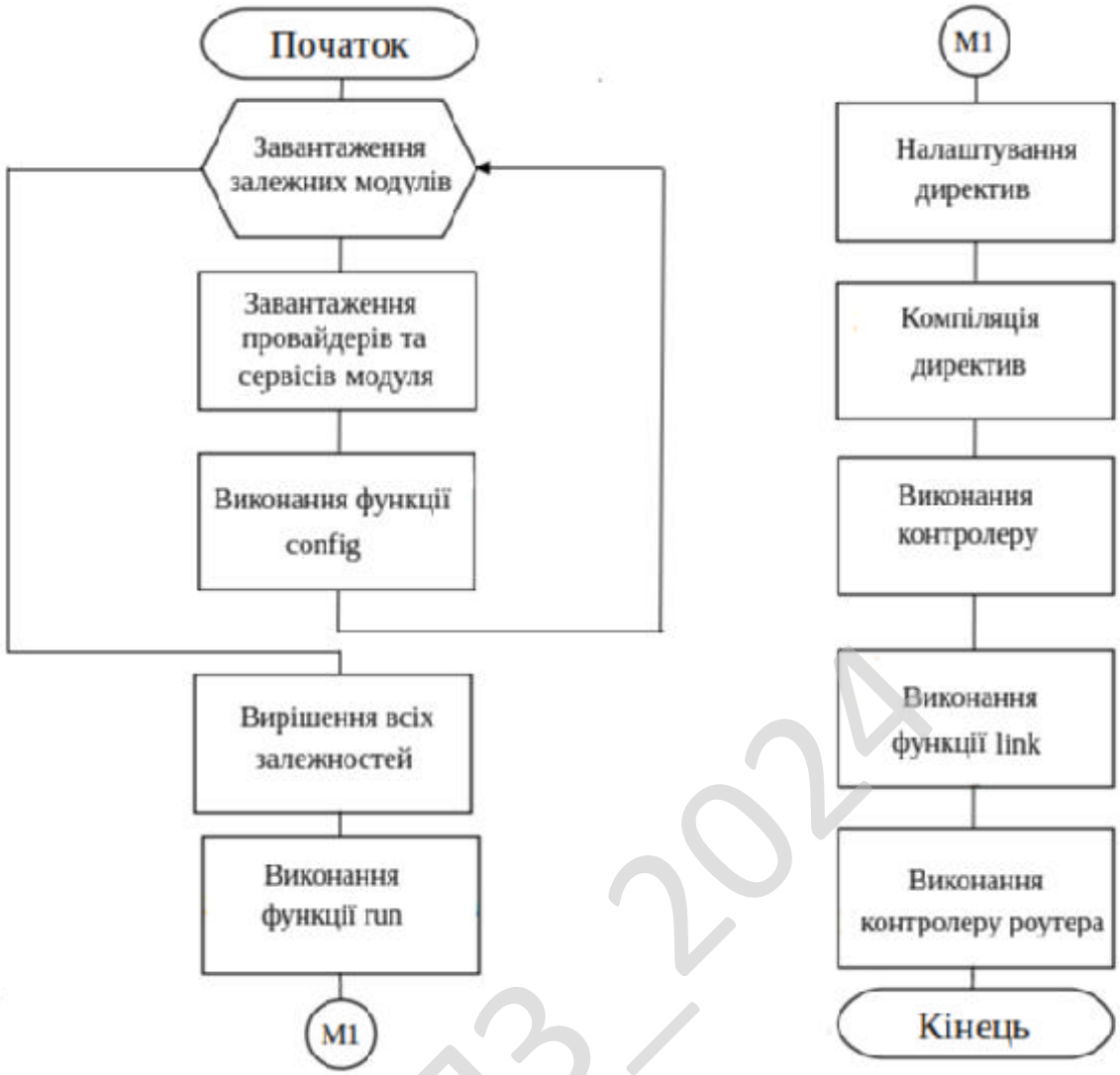


Рисунок 4.9 – Блок-схема алгоритму роботи ІС HR

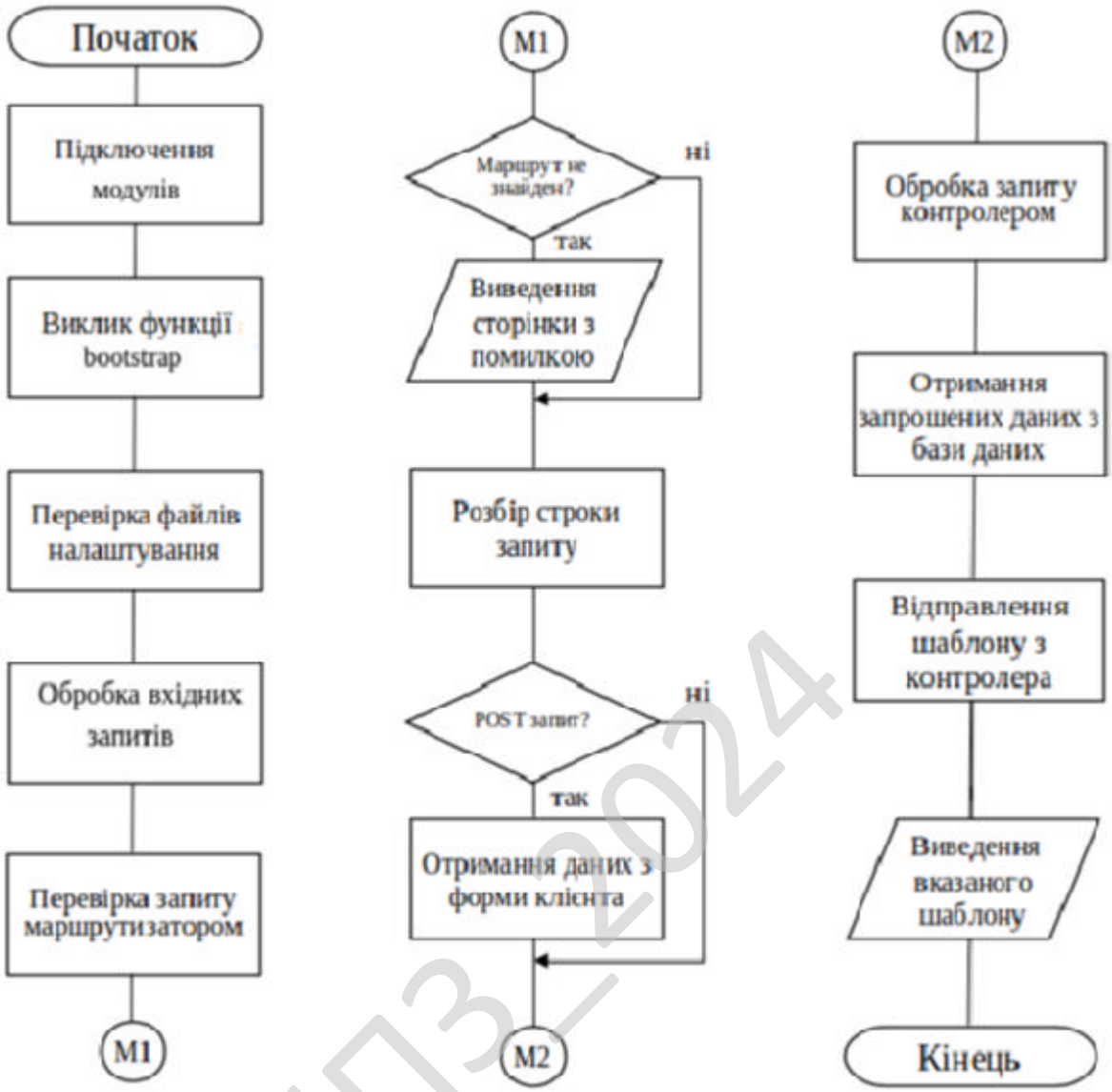


Рисунок 4.10 –Блок-схема алгоритму роботи основної програми

4.2 Захист розробленого програмного забезпечення

Для захисту коду обрано алгоритм bcrypt, його обрано через його перевірену часом безпеку, стійкість до атак та простоту використання. Цей алгоритм забезпечує надійний захист паролів навіть у разі витоку бази даних, що робить його стандартом для сучасних веб-додатків.

Переваги bcrypt у порівнянні з іншими алгоритмами

MD5 та SHA-256:

Ці алгоритми не враховують salt і швидше виконуються, що робить їх менш захищеними від атак brute force.

PBKDF2:

Хоча PBKDF2 теж захищає паролі, bcrypt має вбудовану підтримку salt і простішу інтеграцію.

Argon2:

Argon2 - сучасний конкурент bcrypt із додатковими функціями, проте bcrypt залишається стандартом через широку підтримку та перевірену часом надійність.

Особливості алгоритму bcrypt

Стійкість до атак "грубої сили" (Brute Force)

bcrypt використовує параметр cost factor (work factor), який визначає кількість ітерацій для хешування. Чим вищий work factor, тим більше обчислювальної потужності потрібно для кожної спроби підбору.

Навіть якщо зловмисник отримає доступ до хешів, спроба підібрати паролі буде дуже повільною.

Використання "солоної" хеш-функції

bcrypt додає унікальний salt (випадковий набір символів) до кожного пароля перед хешуванням. Це запобігає використанню попередньо підготовлених таблиць (rainbow tables), які полегшують розшифрування хешів.

Захист від атак на паралельних системах (GPU/ASIC)

bcrypt був розроблений для роботи з обчисленнями на CPU, що робить його менш ефективним для виконання на графічних процесорах (GPU), які часто використовуються зловмисниками для масових атак.

Адаптивність

У майбутньому, якщо обчислювальні потужності зростуть, параметр work factor можна збільшити, щоб алгоритм залишався безпечним.

Зворотна сумісність

Хеші, створені з використанням bcrypt, завжди матимуть стандартну структуру, яка легко перевіряється. Це полегшує інтеграцію з існуючими системами.

Реалізовано аутентифікацію за допомогою JWT

```
const jwt = require('jsonwebtoken');

// Генерація токена
function generateToken(user) {
    const payload = { id: user._id, role: user.role };
    return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '1h' });
}

// Middleware для перевірки токена
function authenticateToken(req, res, next) {
    const token = req.headers['authorization']?.split(' ')[1];
    if (!token) return res.status(401).json({ message: 'Token missing' });

    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
        if (err) return res.status(403).json({ message: 'Token invalid'
});
        req.user = user;
        next();
    });
}

module.exports = { generateToken, authenticateToken };
```

Захист бази даних MongoDB за допомогою ролей

```
db.createUser({
    user: "appUser",
    pwd: "securePassword123",
    roles: [{ role: "readWrite", db: "yourDatabase" }]
});
const mongoose = require('mongoose');

mongoose.connect('mongodb://appUser:securePassword123@localhost:27017/your
Database', {
```

```
});
```

Валідація даних для захисту від ін'єкцій

```
const Joi = require('joi');

const userSchema = Joi.object({
    username: Joi.string().min(3).max(30).required(),
    password: Joi.string().min(8).required(),
    email: Joi.string().email().required(),
});

// Перевірка:
app.post('/register', async (req, res) => {
    const { error } = userSchema.validate(req.body);
    if (error) return res.status(400).json({ message:
error.details[0].message });

});
```

Шифрування паролів

```
const bcrypt = require('bcrypt');

// Хешування пароля перед збереженням
async function hashPassword(password) {
    const salt = await bcrypt.genSalt(10);
    return bcrypt.hash(password, salt);
}

// Перевірка пароля при логіні
async function comparePassword(inputPassword, hashedPassword) {
    return bcrypt.compare(inputPassword, hashedPassword);
}
```

Налаштовано захист від XSS-атак

```
const xssClean = require('xss-clean');

// Middleware для очищення запитів
app.use(xssClean());
```

Впроваджено HTTPS

```
const https = require('https');
const fs = require('fs');

const options = {
    key: fs.readFileSync('path/to/private-key.pem'),
    cert: fs.readFileSync('path/to/certificate.pem'),
};

https.createServer(options, app).listen(443, () => {
    console.log('Secure server running on port 443');
});
const bcrypt = require('bcrypt');

async function hashPassword(password) {
    const saltRounds = 10; // Cost factor
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    return hashedPassword;
}

async function verifyPassword(inputPassword, storedHash) {
    const isMatch = await bcrypt.compare(inputPassword, storedHash);
    return isMatch;
}

// Використання:
(async () => {
    const password = 'securePassword123';
    const hashed = await hashPassword(password);
    console.log('Хеш пароля:', hashed);

    const isValid = await verifyPassword('securePassword123', hashed);
    console.log('Пароль правильний:', isValid);
})();
```

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Щоб WEB-сайт став доступний для відвідувачів, його потрібно розмістити на Internet сервері, підключеному до Мережі і дати йому доменне ім'я.

Дуже важливо, щоб WEB-сайт був розміщений на цілодобово функціонує потужному сервері з високою пропускною здатністю каналу.

Після запуску браузера введіть адресу нашої системи (якщо вона не налаштована системним адміністратором на автоматичне завантаження). У нашому випадку - http://localhost:3000/. У вікні ми бачимо початкову сторінку для авторизації в системі (рисунок 5.1). Після авторизації користувач потрапляє в систему, зовнішній вигляд якої залежить від призначених прав доступу, а також від ролі, наданої користувачеві під час його реєстрації в системі. Як було сказано раніше, система передбачає чотири рівні доступу до системи: адміністратор, керівник проекту, співробітник, бухгалтер.

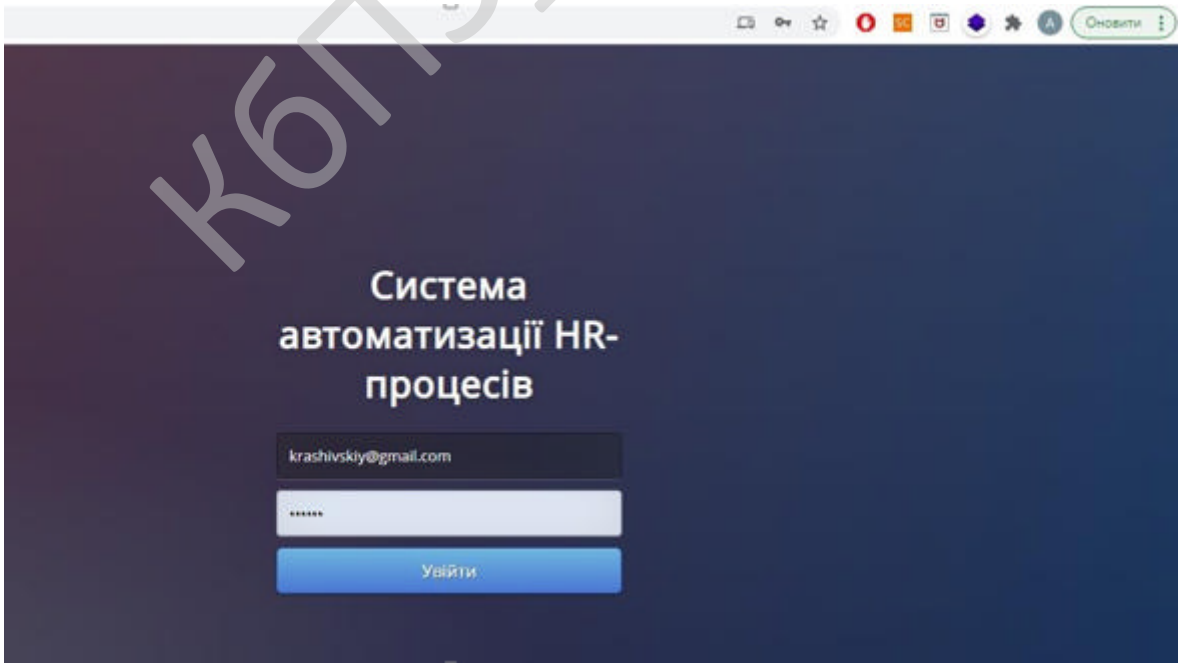


Рисунок 5.1 – Головна сторінка для входу в систему

На рисунку 5.2 зображено головну сторінку системи для адміністратора. У лівій частині сторінки знаходиться головне меню, яке використовується для виклику всіх функцій системи.

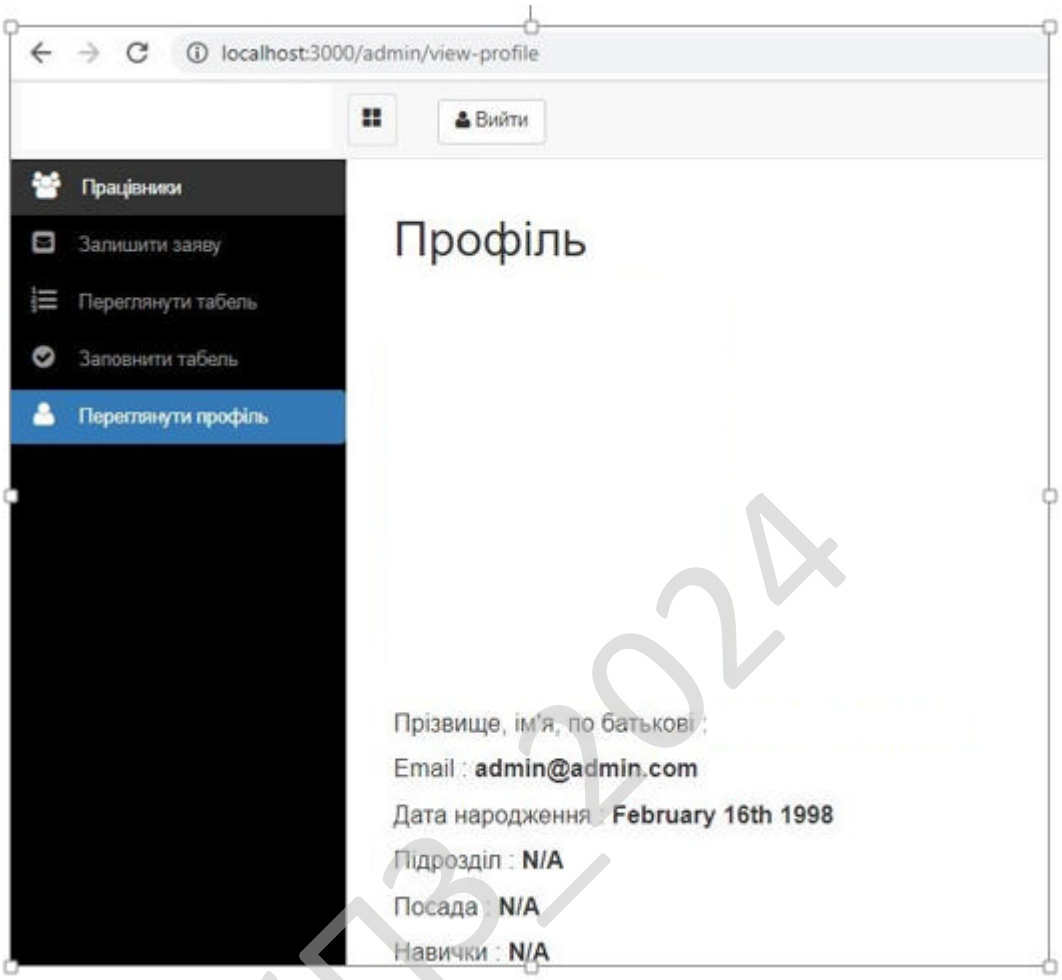


Рисунок 5.2 – Реалізація функціональності для адміністратора

Адміністратор має повний доступ до системи, що включає реєстрацію співробітників, прийняття рішення щодо привілеїв для інших співробітників, перегляд і зміну записів про робочий час, перегляд і зміну заробітної плати співробітників, видалення записів або профілів співробітників, призначення та перепризначення проектів кожному співробітнику, схвалення та відхилення заяви працівників про відсутність.

На рисунку 5.3 зображено форму реєстрації нового співробітника в системі. Після заповнення відповідних даних вся інформація буде зберігатися у відповідних документах розробленої бази даних MongoDB.

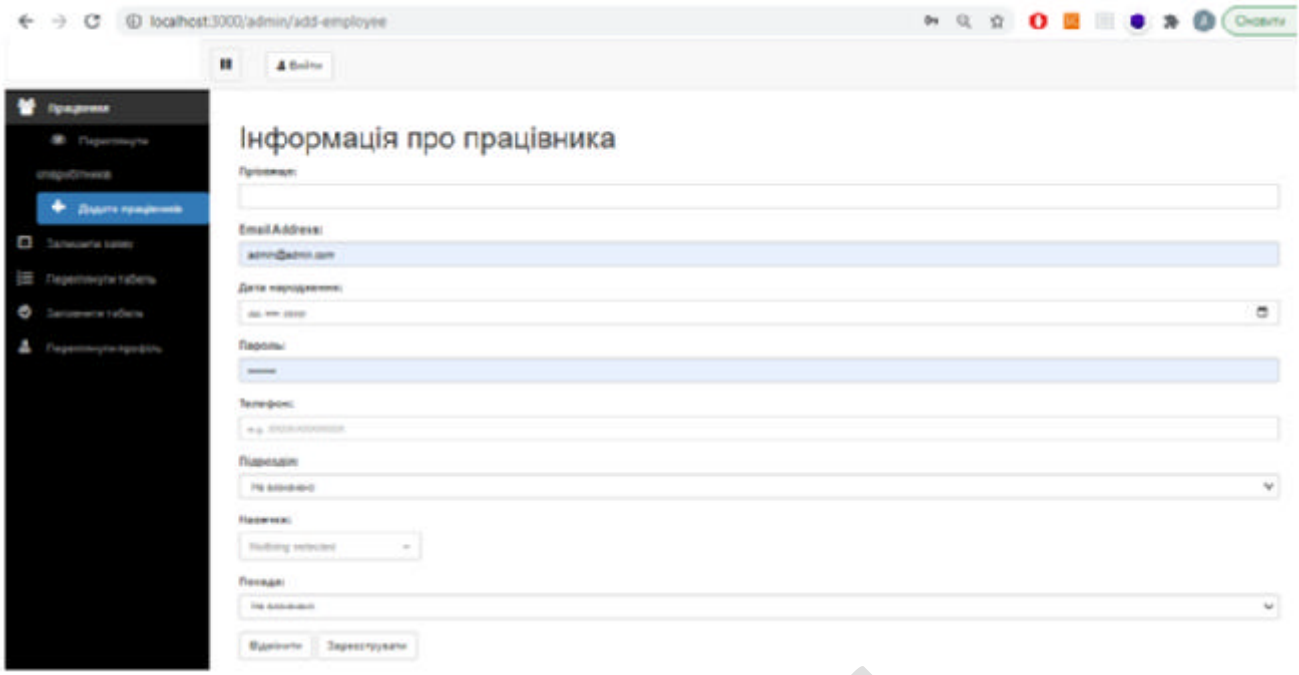


Рисунок 5.3 – Форма реєстрації нового працівника

Обов'язковою умовою успішного додавання співробітника є відмітка відповідних практичних навичок, які будуть використані на етапі реалізації конкретного проекту (рисунок 5.4).

Також адміністратор може бачити інформацію про всіх співробітників, які на даний момент зареєстровані в системі, їх контактну інформацію, посаду, підрозділ, проекти, до яких вони закріплені.

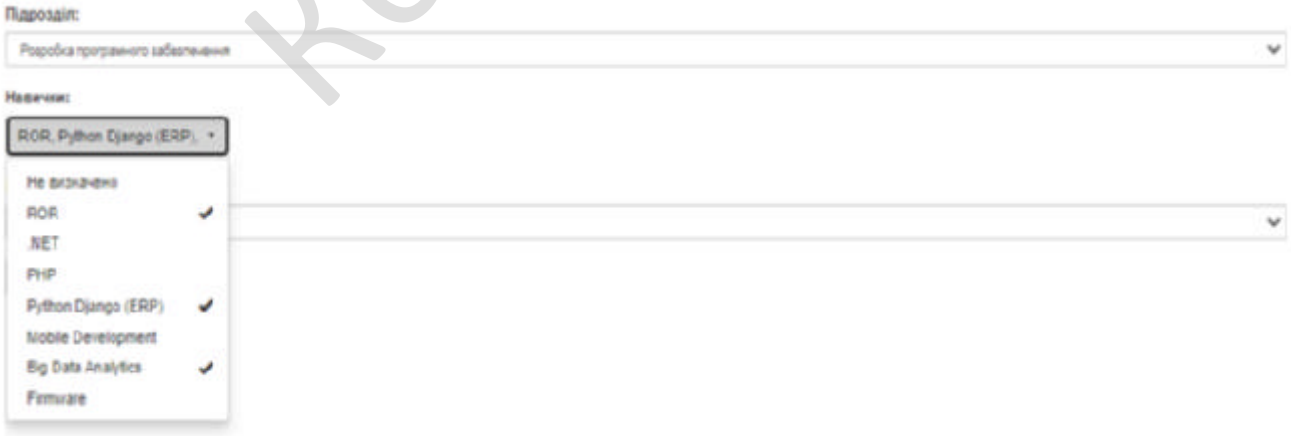


Рисунок 5.4 – Форма відзначення практичних навичок

На рисунку 5.5 представлено сторінку з відображенням інформації про працівників.

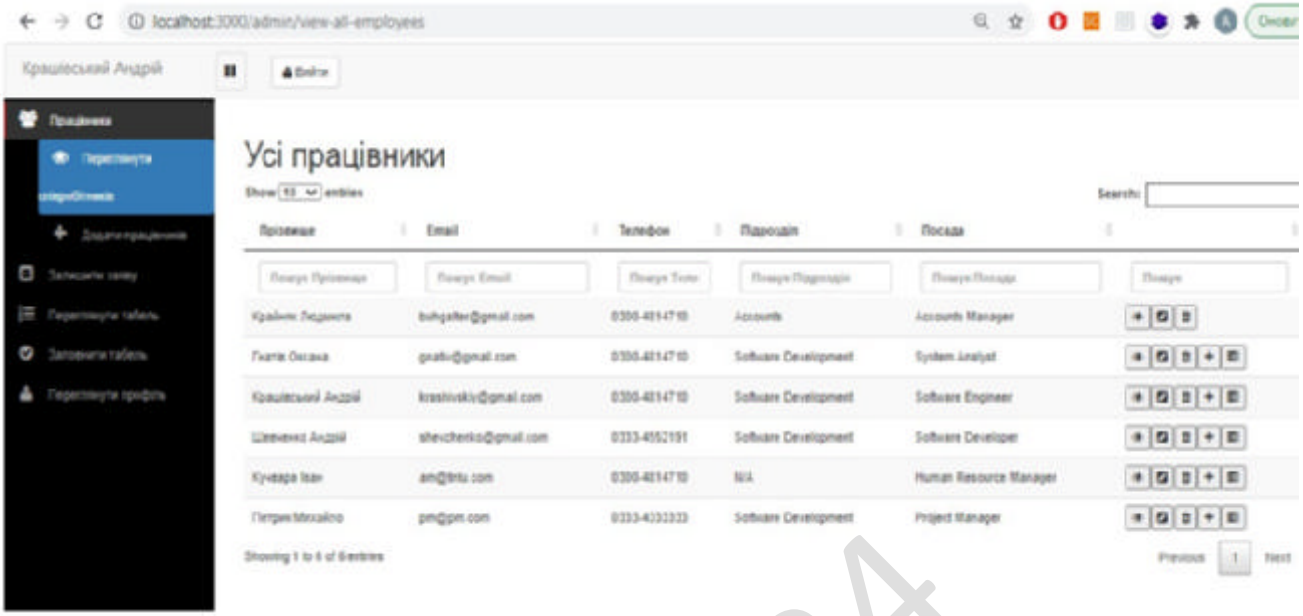


Рисунок 5.5 – Сторінка відображення інформації про працівників, які зареєстровані в системі

Також в ІС реалізована можливість перегляду профілю кожного працівника (рисунок 5.6).

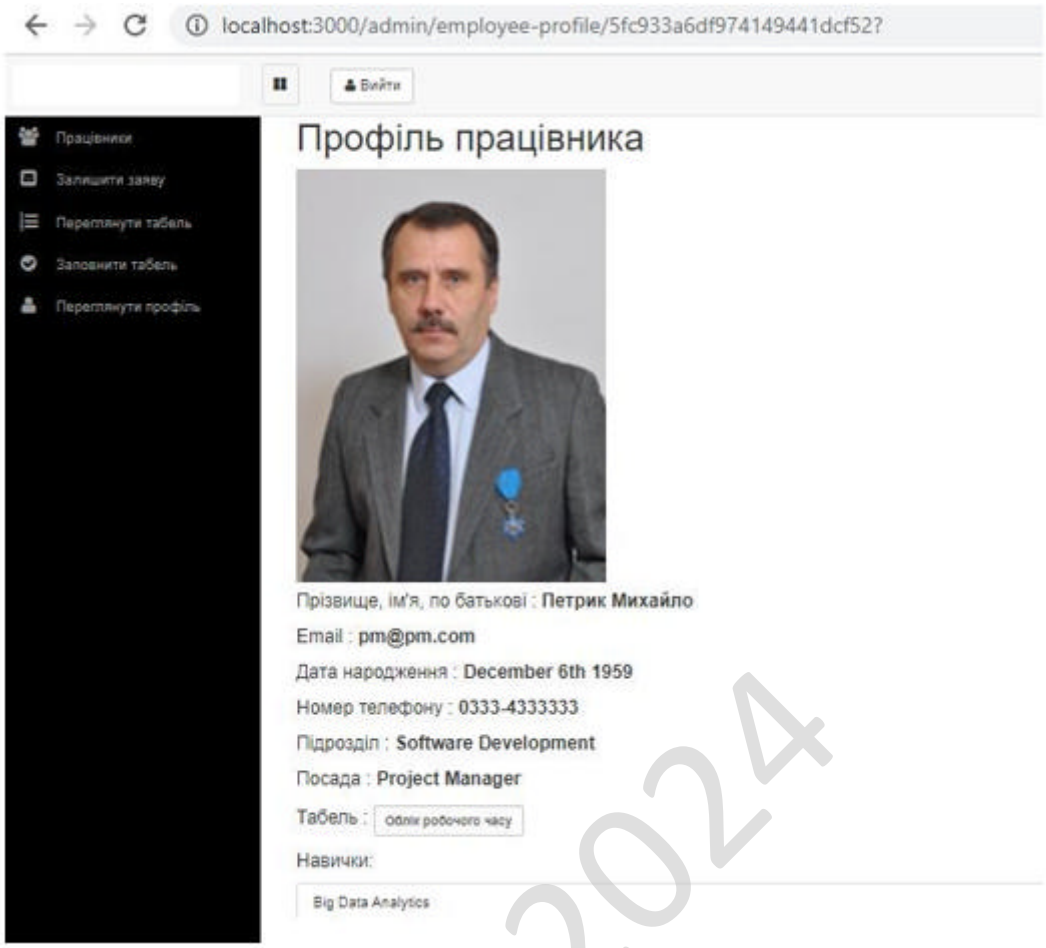


Рисунок 5.6 – Сторінка відображення інформації про працівника

Також в системі реалізовано функціонал, пов’язаний з підрахунком відпрацьованого часу, а також наданням щорічних відпусток, реєстр відповідних заяв. На рисунку 5.7 зображено форму подання працівником заяви про надання відпустки.

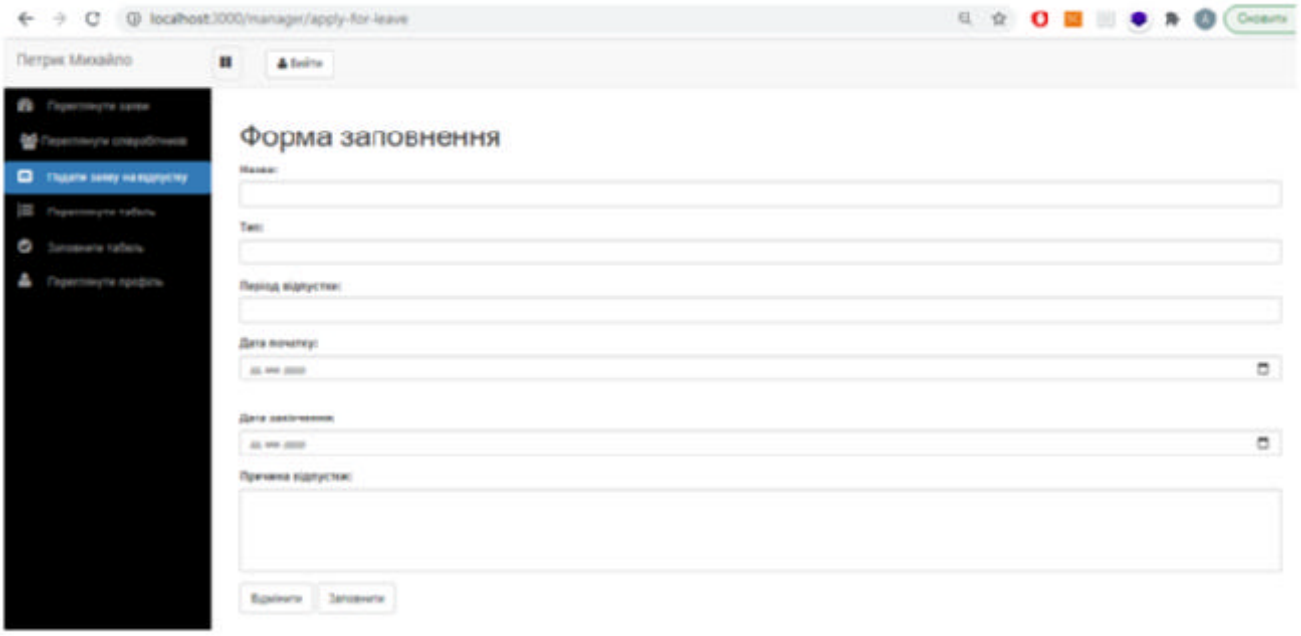


Рисунок 5.7 – Форма подачі заяви на відпустку

Керівник проекту в ІС може призначати працівників на відповідні проекти, а також здійснювати оцінку їх діяльності, формуючи відповідну систему якісної оцінки кожного працівника.

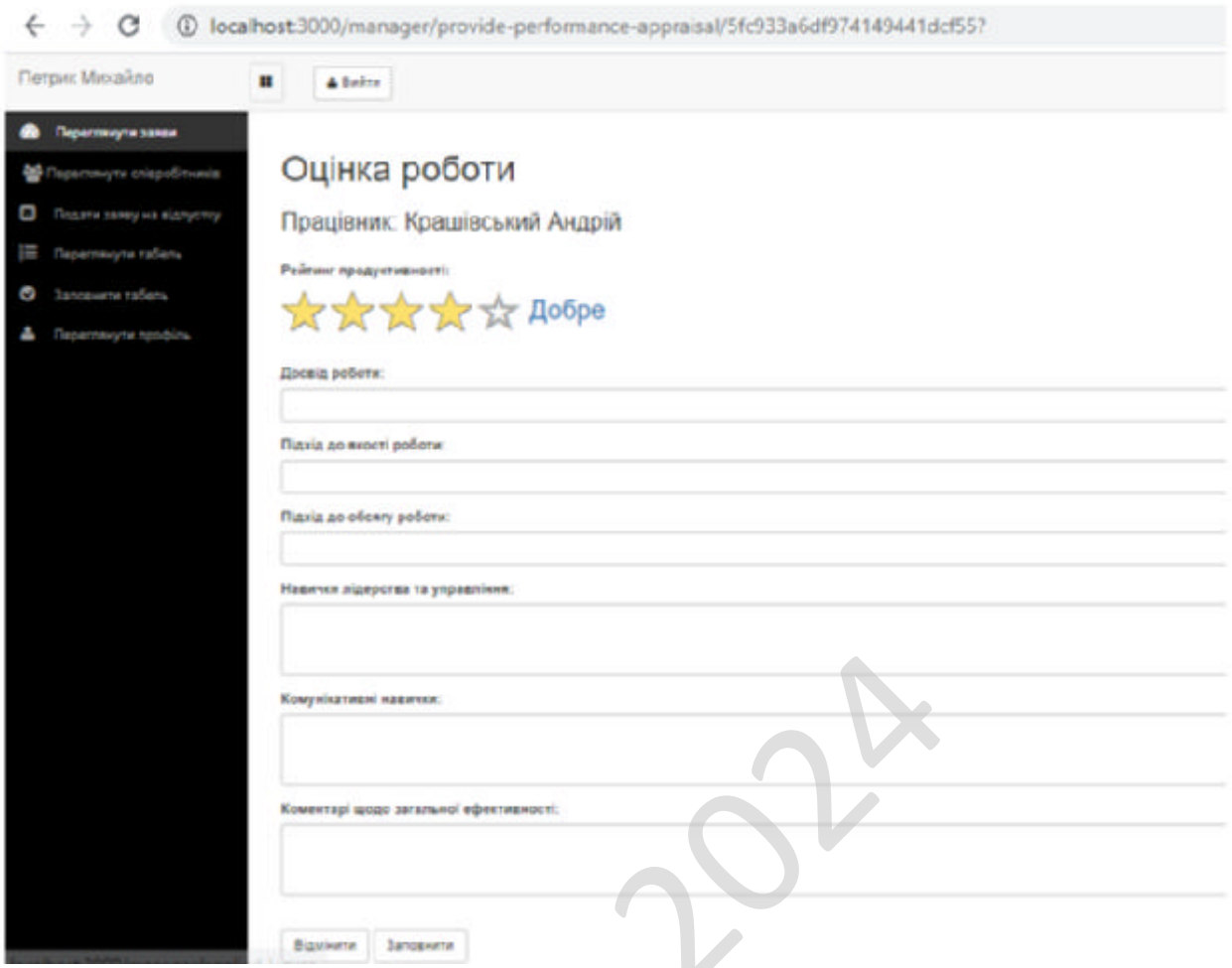


Рисунок 5.8 – Форма оцінки результативності роботи працівників

Особлива увага в системі приділяється питанням обліку відпрацьованого часу і заробітної плати. Кожен співробітник зможе заповнити свій графік, переглянути свою історію, побачити поточну зарплату, переглянути поточний профіль співробітника (включаючи освітню та робочу історію), переглянути всі проекти в організації, побачити інших співробітників, які спільно з ними працюють. , подати заяву на відпустку та переглянути статус відповідної заяви на відпустку.

На рисунку 5.9 зображено сторінку з даними про робочий час працівника за день.

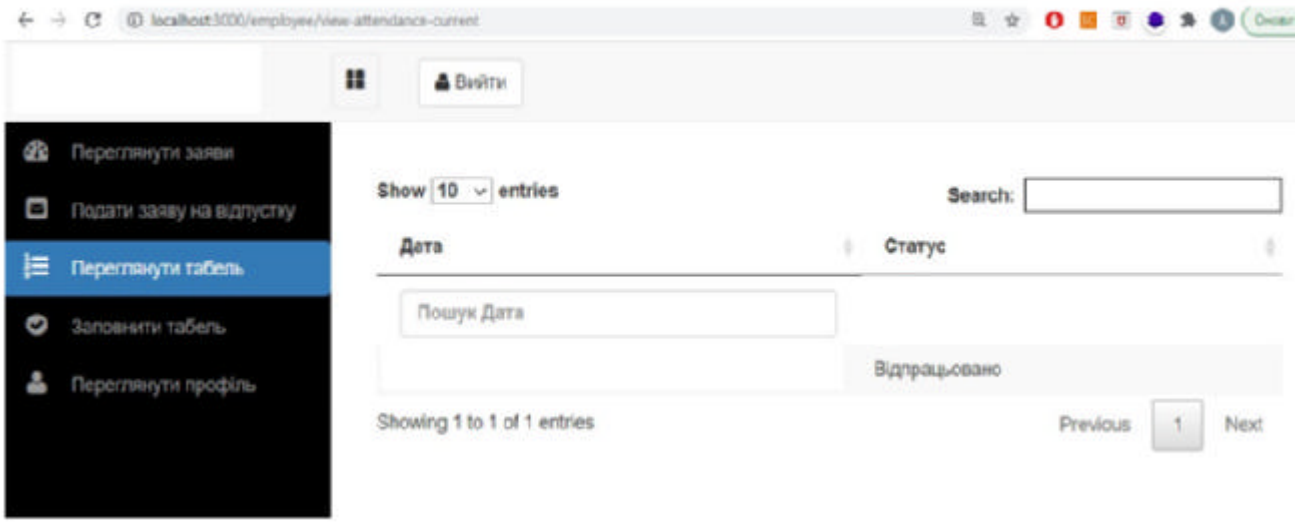


Рисунок 5.9 – Сторінка перегляду відпрацьованого часу працівником

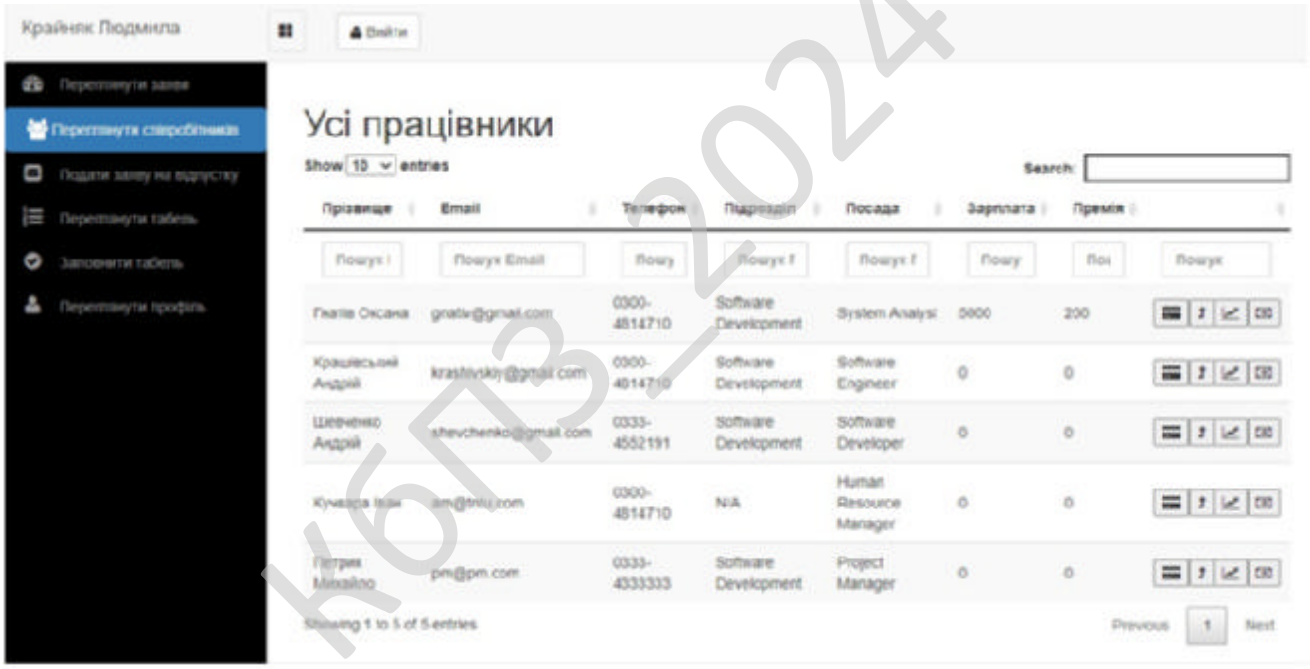


Рисунок 5.10 – Сторінка обліку оплати праці співробітників

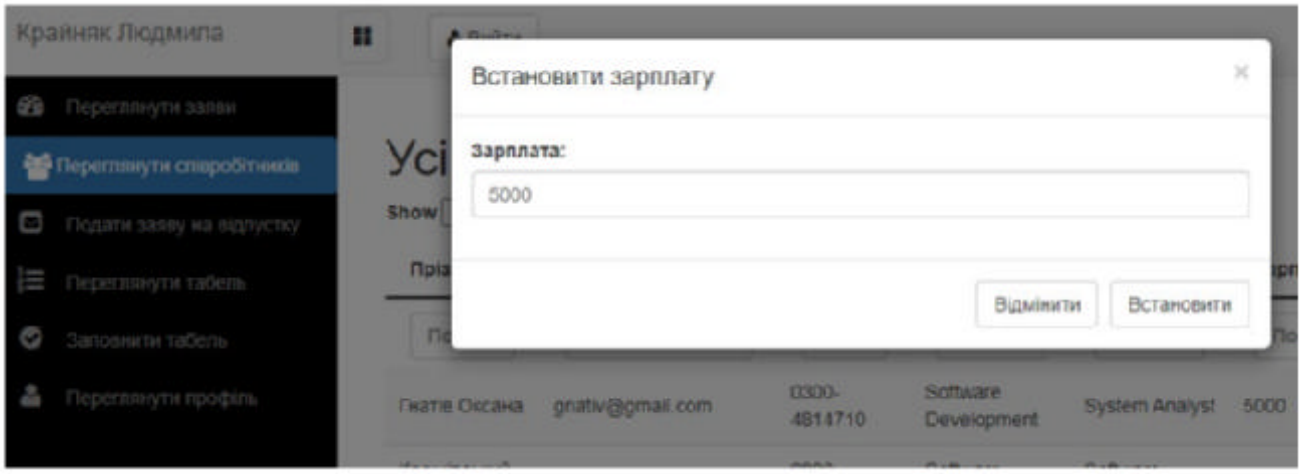


Рисунок 5.11 – Сторінка встановлення зарплати для окремого працівника

Система передбачає можливість створення бонусів для працівників за рахунок відповідних премій. Відповідно до пропозицій, поданих керівниками проекту, ці премії мають диференційований характер. На рисунку 5.12 наведено форму визначення премії працівнику.

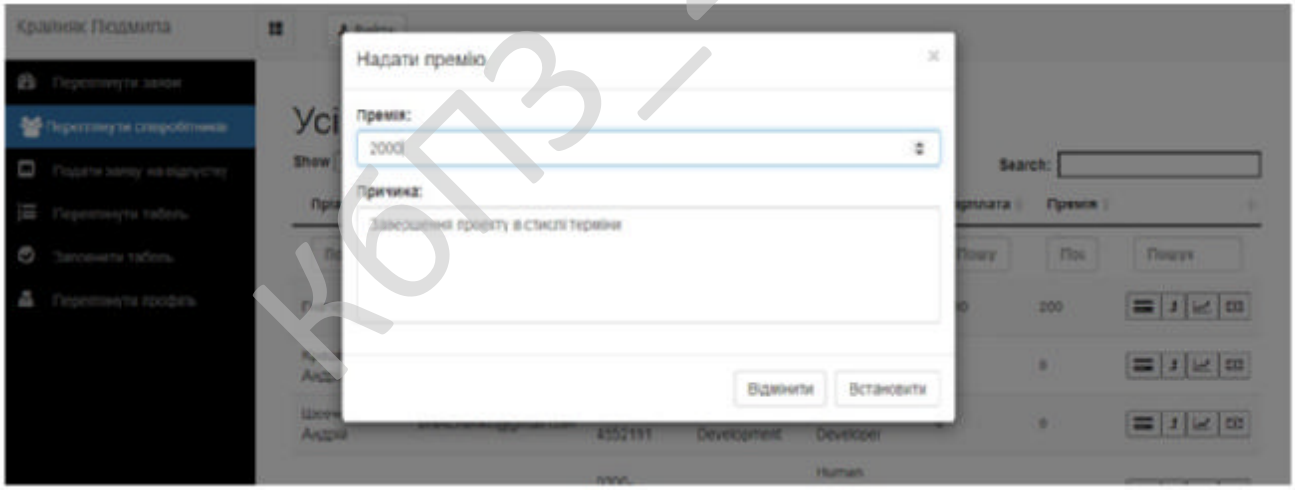


Рисунок 5.12 – Сторінка встановлення премії для окремого працівника

Виходячи з вищесказаного, можна підсумувати напрацьоване інформаційна система дозволяє:

- впровадження окремих облікових записів для всіх адміністраторів і співробітників;

- різні види даних і чіткий доступ до даних для учасників на основі їхніх привілеїв;

- реєстрація співробітників/адміністраторів;

- управління робочим часом усіх працівників та їх заробітком.

- ведення обліку навчального та трудового стажу працівника;

- управління поточними розподіленими проектами співробітників всередині організації.

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для системи аналізу застосування NOSQL баз даних в сучасних ІС.

Метою розробки є дослідження та програмна реалізація системи аналізу застосування NoSQL баз даних в сучасних ІС, оцінці можливостей, переваг та обмежень цих технологій, а також виявленні їхнього впливу на якість та швидкість обробки даних в різноманітних сценаріях використання

Об'єктом дослідження є дослідження та програмна реалізація системи аналізу застосування NoSQL баз даних в сучасних ІС, оцінці можливостей, переваг та обмежень цих технологій, а також виявленні їхнього впливу на якість та швидкість обробки даних в різноманітних сценаріях використання.

Предметом дослідження є самі NoSQL бази даних та їхнє застосування в інформаційних системах. Методи та програмні засоби реалізації інформаційної системи.

Методи дослідження базуються на методах зберігання даних, методах математичної статистики, методах теорії масового обслуговування, методах розробки програмного забезпечення, теорії алгоритмів та об'єктно-орієнтованого проектування.

Наукова новизна отриманих результатів.

У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– удосконалено метод вибору NoSQL БД для роботи з сучасними ІС;

– розроблено інформаційну систему на основі NOSQL БД, яка має більш широкі можливості та швидкість доступу до даних на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати досліджень і програмного впровадження NoSQL баз даних для підвищення ефективності роботи систем обробки даних можуть досить широку аудиторію (рисунок 7.1).

ІТ-компанії та стартапи – для оптимізації роботи своїх систем обробки великих даних, особливо в тих випадках, де важливі швидкість обробки та гнучкість масштабування (наприклад, для веб-додатків, аналітики або хмарних сервісів).
Аналітичні центри та науково-дослідні інститути – для обробки великих обсягів інформації з високою швидкістю, що важливо при роботі з великими даними (Big Data) в наукових або економічних дослідженнях.
Фінансові установи – банки та інші організації, які працюють з великими обсягами даних і потребують високошвидкісного доступу для обслуговування клієнтів та підтримки роботи аналітичних систем.
Медичні заклади та біотехнологічні компанії – для зберігання і швидкого аналізу великих обсягів даних, наприклад, з результатів генетичних досліджень або при обробці медичних записів.
Державні організації – особливо ті, що займаються безпекою, обороною та обробкою даних громадян, де потрібне високошвидкісне управління і доступ до великих обсягів інформації.
Освітні та навчальні заклади – окрема факультети з IT та комп'ютерних наук, де результати можуть стати базою для курсів з баз даних, обробки даних та високопродуктивних обчислень.

Рисунок 7.1 – Цільова аудиторія проєкту

Таким чином, впровадження NoSQL технологій дійсно цікавить значне коло стейкхолдерів і має широку цільову аудиторію.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Оцінка привабливості проєкту програмної реалізації NoSQL баз даних для підвищення ефективності роботи систем обробки даних за допомогою методу експертних оцінок може включати кілька ключових кроків. Розпочинаємо з визначення критеріїв оцінки (рисунок 7.2).

Рисунок 7.2 – Критерії експертної оцінки

Залучаємо експертів з різних сфер, які знайомі з базами даних та обробкою даних: інженери баз даних, IT-архітектори, фінансові аналітики, експерти з безпеки даних, керівники проєктів.

Для кожного критерію експертна група визначає вагу за важливістю (на приклад, на шкалі від 0 до 1, де сума всіх ваг дорівнює 1): продуктивність і масштабованість – 0.3, інтеграція та сумісність – 0.2, зручність впровадження та підтримки – 0.2, економічна ефективність – 0.15, безпека даних – 0.15.

Кожен експерт оцінює проєкт за кожним критерієм за бальною шкалою (наприклад, від 1 до 5, де 5 – найвищий рівень задоволення критерію).

Таблиця 7.1 – Зведені результати експертних оцінок

Критерій	Вага	Експерт 1	Експерт 2	Експерт 3	Середній бал
Продуктивність і масштабованість	0.3	5	4	5	4.67
Інтеграція та сумісність	0.2	4	3	4	3.67
Зручність впровадження	0.2	3	4	3	3.33
Економічна ефективність	0.15	4	5	4	4.33
Безпека даних	0.15	4	4	4	4.00

Підсумкову оцінку привабливості проєкту можна отримати, помноживши середній бал кожного критерію на його вагу і підсумувавши результати: Загальний бал = (4.67×0.3) + (3.67×0.2) + (3.33×0.2) + (4.33×0.15) + (4.00×0.15)

На основі отриманого підсумкового балу проєкт можна оцінити як перспективний або ж такий, що потребує доопрацювання для досягнення необхідної привабливості.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації NoSQL баз даних, яка підвищує ефективність роботи систем обробки даних, можна використовувати різні методи залежно від специфіки проєкту, його масштабів і вимог замовника.

Пропонуємо як основний розглядами метод експертних оцінок. Цей метод використовує оцінки вартості, надані експертами з баз даних або спеціалістами з розробки NoSQL рішень. Це якісний метод, що дозволяє отримати приблизні оцінки на основі досвіду професіоналів.

Процес застосування це першочергово залучення групи експертів з аналогічним досвідом роботи, оцінка вартості на основі дискусій або окремих опитувань, агрегація оцінок і отримання середнього значення.

Переваги, які має метод, це те що він підходить для початкової оцінки, коли конкретні вимоги або параметри ще не визначені.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Економічна ефективність від впровадження NoSQL баз даних для підвищення ефективності роботи систем обробки даних може проявлятися в аспектах згрупованих на рисунку 7.3.

Рисунок 7.3 – Економічна ефективність від реалізації проєкту для клієнта

Економічна ефективність від впровадження NoSQL баз даних може виражатися у скороченні витрат на апаратне забезпечення та ліцензії, зниженні адміністративних витрат, підвищенні продуктивності та збільшенні доходів від покращення користувацького досвіду.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Ось алгоритм просування проєкту програмної реалізації NoSQL баз даних для підвищення ефективності роботи систем обробки даних. Він охоплює етапи від визначення цільової аудиторії до активних маркетингових дій (рисунок 7.4).

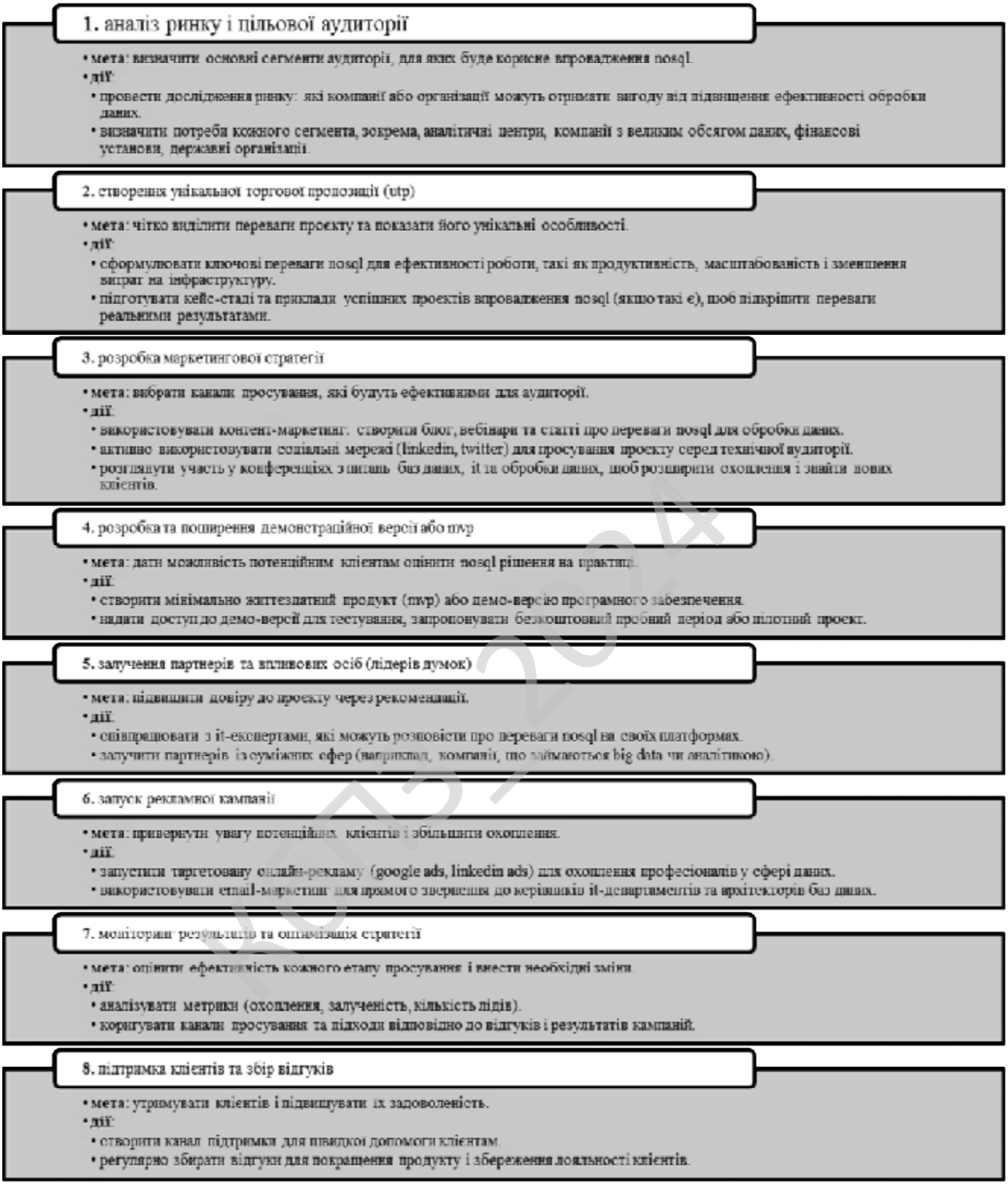


Рисунок 7.4 – Алгоритм просування проєкту

Цей алгоритм допоможе ефективно просувати проєкт, залучати потенційних клієнтів і сприяти росту зацікавленості в NoSQL рішеннях для обробки даних.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації проєкту програмної реалізації NoSQL баз даних можна застосувати декілька стратегій, спрямованих на підвищення ефективності продажу, збільшення охоплення аудиторії та оптимізацію ресурсів:

- сегментація каналів збуту за типом клієнтів;
- залучення партнерських каналів для поширення продукту;
- інтеграція з популярними хмарними платформами;
- автоматизація маркетингових та збутових процесів;
- розробка демо-версій та пробних періодів для нових клієнтів;
- побудова каналу збуту через соціальні мережі та професійні спільноти;
- оптимізація цінової стратегії та впровадження підписок;
- зворотний зв’язок та підтримка клієнтів після продажу.

Застосування цих стратегій сприятиме успішній реалізації та оптимізації збутових каналів для проєкту програмної реалізації NoSQL баз даних, забезпечуючи водночас високу ефективність і задоволення потреб клієнтів.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключові фактори успіху проєкту програмної реалізації NoSQL баз даних для підвищення ефективності роботи систем обробки даних охоплюють технологічні, організаційні та бізнес-аспекти (рисунок 7.5).

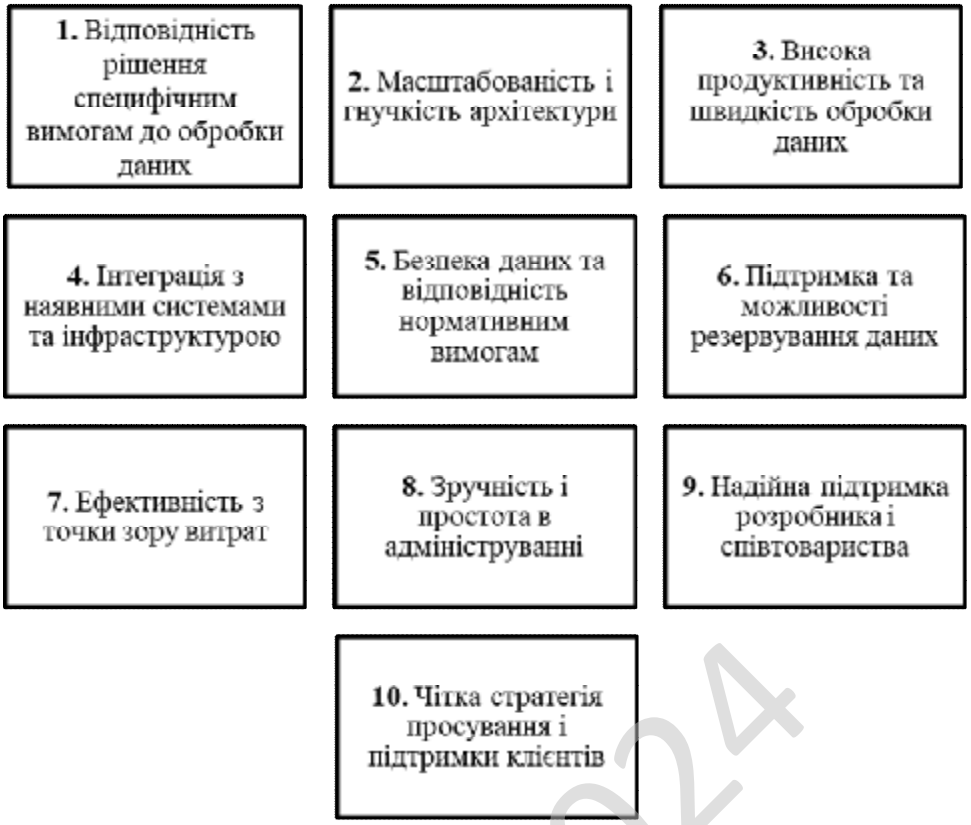


Рисунок 7.5 – Ключові фактори успіху проєкту

Ці ключові фактори є основними для досягнення успіху в реалізації NoSQL баз даних, забезпечуючи проєкту конкурентні переваги та високий рівень задоволення клієнтів.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Аналіз умов праці програміста

Інтернет відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують Інтернет для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин. Багато людей користуються Інтернетом у професійних цілях, оскільки завдяки Інтернету з'явилося багато нових професій. Тому для веб-розробника так важливо розробити зручний інтерфейс для зручного сприйняття інформації, та необхідний функціонал, який буде відповідати необхідним вимогам та навантаженням. Все це вимагає багато часу та великого навантаження з боку розробників.

Тому так важливо слідкувати за умовами праці, в яких відбувається робочий процес. Оскільки захворювання можуть бути спричинені надмірним фізичним або розумовим навантаженням, через велику нервово-емоційну напругу, або через виробниче середовище. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

При роботі за комп'ютером, розробник має велике зорове навантаження, тому йому необхідне належне освітлення приміщення. Якщо в приміщенні недостатньо природного освітлення, потрібно використовувати спеціальні світильники. Також оскільки розробник значний час працює з електричними приборами є можливість, бути ураженим електричним струмом, тому потрібно дотримуватись всіх необхідних норм. Серед основних чиників, які впливають на розробників під час трудової діяльності можна виділити[46]:

1. Рівень освітлення в приміщенні;
2. Температура, вологість в приміщенні;
3. Рівень шуму на робочому місці;

4 .Напруга в електричному ланцюзі, електричні показники.

Розберемо кожний з чиників окремо, проаналізуємо які повинні бути стандарти кожного з чиників, відповідно до правил з охорони праці.

Рівень штучного освітлення

Головним документом для встановлення норм необхідних показників освітлення є ДБН В.2.5-28:2018 «Природне та штучне освітлення» [47].

Сьогодні найбільш розповсюдженішими є світлодіодні ламп. В середньому світловіддача від таких ламп знаходиться на рівні 80-120 Лм/Вт [48]. Джерелом живлення прийнято вважати електричну мережу у 220В. А освітленіть робочого приміщення повина бути Е =300-500 Лк, оскільки робота програміста відноситься до робіт середньої точності з присвоєнням розряду зорових робіт IV[49].

Рівень сітла повинен бути достатнім, щоб працівник міг працювати без навантаження на зір. Це залежить від системи освітлення, кількості світильниув, їх типу та розміщення у приміщенні. Допустиме значення освітленості робочої поверхні приймається Е = 400 лк [50].

Для покращення освітлення комп'ютерній лабораторії будуть використовуватися світлодіодні лампи, а саме FL-LED T8-900 світловий потік яких F=1500лм.

Мікроклімат робочої зони: температура, відносна вологості, швидкість руху повітря

Головним документом для встановлення норм мікроклімату робочої зони є ДСН 3.3. 6.042 -99 «Державні санітарні норми мікроклімату виробничих приміщень» [45].

Праця програміста за важкістю відноситься до легкої фізичної роботи категорії Іа [49]. Де вказано, що в приміщенні, я кому знаходиться компютерне обладнення, повнині бути встановлені певні норми, оскільки офісна техніка є джерелом тепловиділень, що може спричинити підвищення температури. Серед

потимальних параметрів для роботи встановлена температура 23 – 25^{0}С і вологість на рівні 40 – 60% в залежності ві періоду року.

Для підтримки комфортної температури можна використовувати як організаційні методи, наприклад розпорядок дня, так і технічне обладнення, наприклад кондиціонери, вентіляцію. Як правило в холодний період часу використовуються додаткове опалення для підтримання комфортної температури, а в літку встановлюються кондеціонери.

Рівень шуму на робочому місці

Як правило при використанні великої кількості компютерів в одному приміщенні, через гудіння, рівень шуму має значення більше норми. Допустима норма становить менше 50 дБ [48].

Гучний шум негативно впливає на умови праці та організм людини. Якщо шум триває тривалий час цу може спричинии головні болі, біль у вухах, підвищення стомлюваності, зниження концентрації та уваги. Такі симптоми можуть викликати стресові ситуації у людини. Все це шкодить продуктивності працініка та його стану здоровья.

Щоб встановити необхідний рівень шуму, використовують додаткову звукоізоляцію. Для цього найчатіше використовуються мати та плити із скляного та мінерального волокна, м'які плити з деревних стружок, картон, гуму, утеплений лінолеум, а також заміна вікон на звукоізолюючі.

8.2 Заходи профілактики при роботі з комп’ютерною технікою

Санітарно-гігієнічні норми є важливим критерієм при роботі в приміщенні. Від них залежить здоровє працівників, їх рівнень працездатності, втомлюваність. Щоб всього цього уникнути потрібно стежити за нормами на робочому місці.

Якщо говорити про електробезпеку в приміщенні, то в приміщенні необхідно устаткування розподільних щитів спеціальними розетками з

заземлюючими контактами, повині бути заземлені всі прилади і пристрої, час від часу повинна проводитися перевірка всіх приладів, щорічна здача іспитів з охорони праці.

Для оптимальних показників мікроклімату та освітленості потрібні використовувати дефлектор, для організації вентиляції, та повітрообміну. Та перевірку освітленості в приміщенні згідно відділом охорони праці, щоб відповідати нормам для зорової роботи .

Ще однією проблемою з якою часто зустрічаються програмісти є мала рухливість та повний сидячий робочий день. Тому рекомендується час від часу робити невеликі перерви, під час обіднього перериву вживати їжу не на робочому місці. У окремих випадках, коли при дотриманні всіх санітарних норм, працівник все одно себе погано почуває, дозволяється індивідуальний підхід для обмеження роботи з обчислювальними пристроями. Тривалість роботи за комп'ютером не повинна безперервно тривати більше 4 годин.

Для зменшення зорового та нервово-емоційного навантаження, та поліпшення мозкової діяльності рекомендується робити перерви для психологічного та фізичного розвантаження.

В приміщеннях також подині бути протипожежне обладнання, та інструкція у разі надзвичайних ситуаціях. Повина бути особа, яка відповідає за пожежну безпеку, перевіряє обладнання, та системи протипожежного захисту а також щорічне проведення інструктажів серед працівників.

Автоматична пожежна сигналізація повинна відповідати вимогам ДБН ДБН В.2.5-56:2014, яке вимагає використання вогнестійких кабелів та автоматичну роботу системи оповіщення та евакуації людей у випадку надзвичайної ситуації [43].

При перевірці, приміщення повинно відповідати всім нормам пожежної безпеки. Це виконується за допомогою перевірки пожежної охорони та техніки, проведенню інструктажу і своєчасне інформування пожежної охорони про несправність пожежної техніки, впровадження систем протипожежного

захисту. Організаційні та технічні заходи, спрямовані на попередження виникнення пожежі, обмеження поширення вогню та успішної евакуації людей.

8.3 Розрахнок занулення глухозаземленої нейтралі

Занулення, як основний засіб захисту, застосовується в електроустановках до 1 кВ з глухозаземленою нейтраллю трансформатора або генератора [54]. Початкові дані для розрахунку занулення глухозаземленої нейтралі трансформатора виробничого приміщення:

Загальна потужність: P = 5 кВт.

Кількість електродвигунів: m = 10

Потужність освітлювальних приладів: Po = 3 кВт.

Довжина магістрального кабеля: LM =70 м.

Довжина розгалудження: l=22м.

Лінійна напруга U=380 В.

Фазна напруга Uф=220 В.

Визначаємо силу номінального струму електроустановки:

$$Iном = Imax = (P * 1000) / (\sqrt{3} * Uл * cosф) \quad (8.1)$$

$$Iном = Imax = (5 * 1000)/(\sqrt{3} * 380 * 0,85) = 8,9 А$$

де: P − номінальна сумарна потужність електроприладів, кВт;

Uл − лінійна напруга, В;

cosф − коефіцієнт потужності, приймається в залежності від типу електрообладнання в межах 0,8..0,87.

Визначаємо силу пускового струму електродвигуна:

$$Iпус = 5 * Iном \quad (8.2)$$

$$Іпус = 5 * 8,9 = 44,5 А$$

Визначаємо номінальну силу струму апарата захисту:

$$Ін = Іпус/b \quad (8.3)$$

$$Ін = 44,5 /2,5 = 17,8 А$$

b − коефіцієнт пуску електродвигуна − для легких умов пуску − 2,5..3.

Вибираємо запобіжник ПН 2-100 з плавкою вставкою Іном = 50 А. Визначаємо найменше допустиме по умовам спрацьовування захисту значення сили струмукороткого замикання:

$$Іктіп = Ін * К \quad (8.4)$$

$$Іктіп = 50 * 3 = 150 А$$

Ін−номінальний струм апарата захисту;

К − коефіцієнт надійності;

Знаходимо переріз провода або кабеля розгалуження з умови допустимого нагрівання:

$$Ідоп = Івст/а \quad (8.5)$$

$$Ідоп = 50 / 3 = 16,6 А$$

Вибираємо площу перерізу 10 мм^2 (Sф) при числі проводів і = 4 розташований у повітрі. Визначаємо максимальний робочий струм:

$$Iроб = Ко(Кз*(Р*1000)/(\sqrt{3}*Uл*cosф)*m+Кз*(Ро*1000)/(\sqrt{3}*Uл*cosф)) \quad (8.6)$$

$$Iроб = 0,75*(0,85*((5*1000)/(1,73*380*0,85)*10+(3*1000)/(1,73*380*0,85))=60,4\ А$$

Ко − коефіцієнт одночасності роботи групи електроприймачів;

Ко = 0.7...0.8; Кз = 0.8...0.9;

Кз – коефіцієнт завантажених електродвигунів;

Ро =3кВт − потужність освітлювальної мережі;

Визначається струм короткочасного перевантаження магістрального кабеля:

$$Iпер = Ко(Кз*(Р*1000)/(\sqrt{3}*Uл*cosф)*n+Кз*(Ро*1000)/(\sqrt{3}*Uл*cosф))+Iпус \quad (8.7)$$

$$Iпер = 0,75*(0,85\ *(5*1000)/(1,73*380*0,85)*9+0,85*(30*1000)/(1,73*380\ *\ 0,85)) + 44,5 = 130,0643\ А$$

Струм спрацювання електромагнітного розчеплювача додатково перевіряємо по максимальному струму перевантаження лінії:

$$Iспр >= 1,25\ *\ Iпер \quad (8.7)$$

$$Iспр >= 1,25\ *\ 130,0643 = 162,5803\ А$$

Приймаємо Iспр = 162,5803 А. Вимикач : А3714Б.

Вибираємо площу перерізу Sф магістрального кабеля (провідника) по Iдоп. Sф = 70mm^2 , − кабель АВРГ прокладений в землі, і=3 (число проводів). Вибрану площу перерізу перевіряємо для автоматів з електромагнітним розчеплювачем:

$$Iдоп >= Iспр/4,5 \quad (8.8)$$

$$Iдоп >= 162/4,5 = 36 А$$

Проводимо узгодження з номінальним струмом автомата:

$$Iдоп = Iспр/3 \quad (8.9)$$

$$Iдоп = 162/3 = 54 А$$

Значення 36 і 54 А. менше ніж Imax= 60,4 А., значить площа перерізу кабеля вибрана вірно. Визначаємо потужність трансформатора:

$$Nтр = ((Кп * Рном)/cosф) \quad (8.10)$$

$$Nтр = (0,7 * 53)/0,8 = 46,375 кВ*А$$

Рном − сумарна потужність електроприймачів, кВт;

Cosф − середній коефіцієнт потужності електроприймачів (0,8);

Кп − коефіцієнт попиту (0,7);

Одержане значення потужності трансформатора округляємо до ближчого стандартного значення. Визначаємо опір трансформатора Z_T. Вибираємо трансформатор на 40 кВА (Z_T =0,562 Ом). визначаємо орієнтовно площу перерізу провідника. Для магістрального кабеля:

$$Sn_1 >= 0,5 * Sф \quad (8.11)$$

$$Sn_1 >= 0,5 * Sф = 0,5*70=35 мм^2$$

Визначаємо для розгалуження:

$$Sn_2 >= 0{,}5*10 = 5 \text{ мм}^2$$

Округляємо ці значення до найближчих більших 35 мм^2 (Sn1), і 6 мм^2 (Sn2). Визначаємо активний і індуктивний опір фазного і нульового захисного провідників на дільницях 1 і 2:

$$Rф = p * (Lм/Sф_1) + p * (L/Sф_2) \qquad (8.12)$$

$$Rф = 0{,}028 * (70/70) + 0{,}028 * (22/10) = 0{,}0896 \text{ Ом}$$

$$Rn = p * (Lm/Sn_1) + p * (L/Sn_2) \qquad (8.13)$$

$$Rn = 0{,}028 * (70/35) + 0{,}028 * (22/6) = 0{,}1586 \text{ Ом}$$

Для окремо проложених нульових провідників иого приимають рівним 0,6 Ом/км. При прокладці кабелем, або в сталєвих трубах індуктивним опором нехтують.

Знаходимо дійсне значення (модуль) струма однофазного короткого замикання:

$$Iкр = Uф/((Zт/3) + \sqrt{((Rф + Rn)^2 + (Xф + Xn + X'n^2))} \qquad (8.14)$$

$$Iкр = 220/((0{,}562/3) + \sqrt{(0{,}0896 + 0{,}1586)^2}) = 418{,}18 \text{ А}$$

Визначення максимальноі напруги на корпусі обладнання відносно землі при замиканні фази на корпус.

$$Uктах = Iкр * Zн \quad (8.15)$$

$$Uкmax = 418,18 * 0,1586 = 66,32 В$$

$Z_Н$ − повний опір нульового провідника.

Умова не виконується. Необхіно збільшити перерізи Sn 1 та Sn 2 до Sф1 та Sф2 і зробити перерахунок:

$$Rn = 0,028 * (70/70) + 0,028 * (22/10) = 0,0896 Ом,$$

$$Iкр = 220/((0,562/3) + \sqrt{(0,0896 + 0,0896)^2}) = 600,21 А,$$

$$Uкmax = 600,21 * 0,0896 = 53,77 В.$$

Умова не виконується, необхідно або замінити запобіжник з плавкою вставкою на автоматичний вимикач із струмовим реле, що дає можливість зменшити час замикання на копус і підвищити допустиму напругу на корпусі або застосувати повторне заземлення нульового захисного провідника. Повторне заземлення нульового захисного провідника:

$$Rn=(Uдоп * Ro)/((Iкр * Zн)−Uдоп) \quad (8.15)$$

$$Rn= 36 * 4/((600,21 * 0,0896) - 36) = 8,09 Ом.$$

8.4 Висновки

Існує багато факторів, які можуть вплинути на роботу розробника, через некомфортні або навіть небезпечні умови праці, які знаходяться в приміщенні. Серед основних причин виділяється: недостатній рівень світла, гучний шум, високий рівень навантаження, умови мікроклімату. Під час дослідження теми,

були переглянуті можливі шкідливі та небезпечні ситуації, які можуть виникнути на робочому місці та спообби їх уникнення та ліквідації.

В результаті можна зробити висновки, які умови повині бути для продуктивної роботи працівника, як повино бути організоване приміщення і його робоче місце. За допомогою проведених обчислень, можна становити необхідні для комфортної роботи умови.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для порівняння та виявлення кращих технологій відповідно до поставлених сучасних вимог у галуз веб-розробки, а саме застосування різних типів БД при розробці додатків.

У магістерській роботі представлені дані з яких можна зробити висновки, які технології сьогодні використовуються, у яких сферах, та для його функціоналу призначені бази даних різних типів. Які можливості та для яких проектів краще використовувати класичні реляційні а коли ООБД.

Для отримання результатів були проведені наступні дослідження:

– Було створено односторінковий додаток засобами NodeJS.

– Було розробленно та протестовано функціонал REST API та веб-сокети за допомогою AngularJS.

– Було створенно сервер засобами NODE.JS, з підтримкою бази даних MongoDB.

– Було створено шаблони, які можна надалі використовувати при створенні нових додатків.

Розроблені під час виконання магістерської роботи додатки дозволяють чітко зрозуміти для яких проектів краще підходять реляційні а коли NOSQL бази даних, які їх переваги та недоліки.

Розроблені додатки підтримують функціонал, який використовується на більшості сайтів в Інтернеті, серед основних можливостей це додавання медіа файлів, можливість надсилання e-mail повідомлень, використання сокетів для спілкування в чаті, авторизація, динамічний інтерфейс. За допомогою фреймворків була продемонстрована технологія односторінкових додатків, яка надала можливість розробити швидкий та зручний для сприйняття інтерфейс.

При розробці використовувалися мови JavaScript, та TypeScript що дало можливість порівняти на скільки сильно відрізняються фреймворки Angular та AngularJS та їх основні концепції. З цього можна зробити висновки скільки займає час розробки, надійність та можливість подальшого підтримання продукту.

В результаті розроблені додатки відповідають поставленим вимогам технічного завдання, та мають можливість на подальше вдосконалення. Створені додатки можуть бути впровадженні у виробництво.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бахрушин В. Є. Методи аналізу даних: Навч. посібник / В. Є. Бахрушин. – Запоріжжя: КПУ, 2011. – 268 с

2. http://www.nbuv.gov.ua/eb/ep.html - Національна бібліотека України імені В.І.Вернадського

4. Node.js v14.0.0 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/api/

7. John Resig, Bear Bibeault, Josip Maras // “Secrets of theJavaScript Ninja”.

8. Express [Електронний ресурс] – Режим доступу до ресурсу: http://expressjs.com/

9. Фреймворк AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AngularJS

10. AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://angularjs.org/

11. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/

12. React.js [Електронний ресурс] – Режим доступу до ресурсу: https://https://uk.reactjs.org/

13. AngularJS MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

14. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

15. Angular 2 [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

16. Односторінковий застосунок [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Односторінковий_застосунок

18. Build Node.js Apps [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial

19. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

20. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/

23.Мова розмітки гіпертексту [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTML

24. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/HTML

25. Каскадні таблиці стилів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CSS

26. Стек MEAN [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MEAN_(веброзробка)

27. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MongoDB

28. Веб-технології для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web

29. CORS, XSS [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/maleta/cors-xss-and-csrf-with-examples-in-10-minutes-35k3

30. AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX

31. Fetch API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API

32. AngularJS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/angular/default.asp

33. jQuery [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JQuery

34. Основи Web-технологій [Електронний ресурс] – Режим доступу до ресурсу:https://pidruchniki.com/1243020547796/informatika/web-tehnologiyi_pidpriyemstvah

35. npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/

36. Install MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mongodb.com/manual/administration/install-on-linux/

38. Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Linux

39. npm-audit [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/cli/audit

40. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

41. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

42. MongoDB Compass [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass

43. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/

44. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

45. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA

46. Охорона праці [Електронний ресурс]: реферат – Режим доступу до ресурсу: https://revolution.allbest.ru/life/00468031_0.html

47. Природне і штучне освітлення ДБН В.2.5-28:2018: державні будівельні норми України [Електронний ресурс] / Ю. Громадський, С. Облакевич, М.Громадський, Г. Фаренюк, Є. Фаренюк, О. Підгорний, О. Сергейчук, Є.Рейцен, В. Єгорченков, Л. Коваль, Д. Радомцев, В. Злоба, Н. Кучеренко, Г.Кожушко, О. Гончар, О. Козенко, Б. Шабашкевіч, Ю. Добровольський, В.Акіменко, С. Гозак, А. Яригін, В. Назаренко, В. Мартиросова, В. Сорокін,

Є.Пугачов - Київ 2018 - Режим доступу до ресурсу: https://ledeffect.com.ua/images/__branding/dbn2018.pdf

48. Розрахунок світлодіодного освітлення кімнати [Електронний ресурс] – Режим доступу до ресурсу: https://luxled.biz.ua/rozrahynok-svitlodoidnogo-osvitlennja-kimnatu-v-kvarturi-abo-bydunky

49. Охорона праці, охорона праці та безпека в надзвичайних ситуаціях : метод. вказ. до викон. розділів у дипломних роботах / [укл. В.М. Челябієва, О.Л. Гуменюк] - Чернігів ЧДТУ 2013 - [Електронний ресурс] – Режим доступу до ресурсу: http://ir.stu.cn.ua/bitstream/handle/123456789/12461/Охорона праці та безпека. в надзв. ситуац;метод.вказ..pdf?sequence=1&isAllowed=y

50. Освітленість робочих місць [Електронний ресурс] – Режим доступу до ресурсу:https://ua-referat.com/Освітленість_робочих_місць_сучасні_підходи_до_вимірів_і_оцінки

51. Температурний режим праці [Електронний ресурс] – Режим доступу до ресурсу: http://poltava.medprof.org.ua/poltava/zakhist-trudovikh-ta-socialno-ekonomichnikh-prav-pracivnikiv-galuzi/pravova-dopomoga/temperaturnii-rezhim-praci-jakim-vin-maje-buti/

52. Шум. Методи захисту від його дії : метод. вказ. до лабораторної роботи / [укл. В. І. Шмирко, С. М. Журавель] — Запоріжжя: ЗНТУ, 2014. - 14 с. [Електронний ресурс] – Режим доступу до ресурсу: https://zp.edu.ua/sites/default/files/konf/oop_shum-2014.pdf

53. Системи протипожежного захисту ДБН В.2.5-56:2014 / Б. Платкевич, В. Носач, В. Федюк, В. Мусійчук, В. Євстіфєєв, Г. Дубінський, В. Сокол, А.Бушиленко, В. Дунюшкін, Р. Уханський, С. Пономарьов, В. Приймаченко, А.Приймаченко, С. Пітайчук, Н. Морозова, І. Колосов, О. Лагода, П. Мізін, В.Савченко, М. Федорович, П. Шаповалов, Л. Фесенко — Київ 2015 — [Електронний ресурс] – Режим доступу до ресурсу: http://kbu.org.ua/assets/app/documents/dbn2/98.1. ДБН В.2.5-56~2014. Системи протипожежного захисту.pdf

54. Охорона праці. Ч. 2. Занулення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ IBM−сумісного типу / [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака, А. Е. Солових, С. Е. Катеринич] ; Центральноукраїн. нац. техн. ун-т. - 2−ге вид., перероб. та доп. - Кропивницький : ЦНТУ, 2019. - 27 с.[Електронний ресурс] – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/8769

55. The Top JavaScript Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-122.24.0011.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Погорілий Д.В			Дослідження та програмна реалізація NoSQL баз даних для підвищення ефективності роботи систем обробки даних	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Марченко К.М				Б	
Н. Контр.		Коваленко А.С.				ЦНТУ КІ-23М	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку та програмну реалізацію додатку з використанням NoSQL баз даних.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 18-13 від 07.08.2024 року).

3 Мета та призначення розробки

Метою розробки є дослідження та програмна реалізація додатку з використанням NoSQL баз даних. В результаті роботи розглянуто методи управління реплікаціями в NoSQL системах, розроблено програмну реалізацію системи автоматизації HR-процесів, яка представляє собою веб-додаток з використанням Node.js та MongoDB, Node Express, який є веб-додатком для Node.js, а інтерфейс створений за допомогою HTML, Bootstrap, CSS та JS

4 Джерела розробки

Джерелом цієї магістерської роботи є відносна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– Розробку додатку;

– систему підключення та тестування баз даних ;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище NodeJS та СУБД MongoDB

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

Виконати маркетингове то економічне обґрунтування проєкту

8 Вимоги щодо охорони праці

В частині охорони праці та техніки безпеки в магістерській роботі повинен бути розглянутий аналіз умов праці програміста та розрахунок захисного заземлення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.

– Блок-схема алгоритму роботи програми – 2 аркуша.

– Показники економічної ефективності – 1 аркуш.

– Пояснювальна записка – 102 аркуша.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської дипломної роботи. Постановка задачі на виконання бакалаврської дипломної роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень бакалаврської дипломної роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

11.1 Подання магістерської дипломної роботи на попередній захист 07.12.2024 р.

1.2 Подання магістерської роботи на захист 12.12.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Марченко К.М.

Дослідження та програмна реалізація NoSQL баз даних для підвищення ефективності роботи систем обробки даних

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 17

Літера: РП

Кропивницький – 2024

```
var express = require('express'); var router = express.Router();
var passport = require('passport'); var User = require('../models/user');
var Project = require('../models/project'); var csrf = require('csurf');
var csrfProtection = csrf();
var config_passport = require('../config/passport.js'); var moment =
require('moment');
var Leave = require('../models/leave');
var Attendance = require('../models/attendance');



router.use('/', isLoggedIn, function isAuthenticated(req, res, next)
{
next();
});
/**
*     Description:
*     Displays home page to the admin
*
*
*     Known Bugs: None
*/
router.get('/', function viewHome(req, res, next) {
res.render('Admin/adminHome', {
title: 'Admin Home', csrfToken: req.csrfToken(),
userName: req.session.user.name
});
});
/**
*     Description:
*     First it gets attributes of the logged in admin from the User Schema.
*     Attributes are get with the help of id of logged in admin stored in
session.
*/
router.get('/view-profile', function viewProfile(req, res, next) {
User.findById(req.session.user._id, function getUser(err, user)
{
if (err) {
console.log(err);
```

```
}
res.render('Admin/viewProfile', { title: 'Profile',
csrfToken: req.csrfToken(), employee: user,
moment: moment,
userName: req.session.user.name
});
});

});
/**
*     Description:
*     Sorts the list of employees in User Schema.

*     Known Bugs: None
*/

router.get('/view-all-employees', function viewAllEmployees(req, res, next) {
var userChunks = []; var chunkSize = 3;
//find is asynchronous function
User.find({$or: [{type: 'employee'}, {type: 'project_manager'},
{type: 'accounts_manager'}]}).sort({_id: -1}).exec(function getUsers(err, docs)
{
for (var i = 0; i < docs.length; i++) { userChunks.push(docs[i]);
}
res.render('Admin/viewAllEmployee', { title: 'All Employees', csrfToken:
req.csrfToken(), users: userChunks,
userName: req.session.user.name
});
});



});
/**
*     Description:
*     Displays add employee form to the admin.
*/
```

```
router.get('/add-employee', function addEmployee(req, res, next) { var messages
= req.flash('error');

var newUser = new User(); res.render('Admin/addEmployee', {









});









});


title: 'Add Employee', csrfToken: req.csrfToken(), user: config_passport.User,
messages: messages,

hasErrors: messages.length > 0, userName: req.session.user.name




/**

*     Description:

*     First it gets the id of the given employee from the parameters.

*/

router.get('/all-employee-projects/:id', function getAllEmployeePojects(req,
res, next) {

var employeeId = req.params.id; var projectChunks = [];


//find is asynchronous function Project.find({employeeID:
employeeId}).sort({_id: -

1}).exec(function findProjectOfEmployee(err, docs) { var hasProject = 0;
```

```
if (docs.length > 0) { hasProject = 1;
}
for (var i = 0; i < docs.length; i++) { projectChunks.push(docs[i]);
}
User.findById(employeeId, function getUser(err, user) { if (err) {
console.log(err);
}
res.render('Admin/employeeAllProjects', { title: 'List Of Employee Projects',
hasProject: hasProject,
projects: projectChunks, csrfToken: req.csrfToken(), user: user,
userName: req.session.user.name
});
/**
*/
});
});
});
router.get('/leave-applications', function getLeaveApplications(req, res, next)
{
var leaveChunks = [];
var employeeChunks = []; var temp;
//find is asynchronous function
Leave.find({}).sort({_id: -1}).exec(function findAllLeaves(err, docs) {
var hasLeave = 0;
if (docs.length > 0) { hasLeave = 1;
```

```
}
for (var i = 0; i < docs.length; i++) { leaveChunks.push(docs[i])
}
for (var i = 0; i < leaveChunks.length; i++) {
User.findById(leaveChunks[i].applicantID, function getUser(err, user) {
if (err) {
console.log(err);
}
employeeChunks.push(user);
})
}



seconds


// call the rest of the code and have it execute after 3
setTimeout(render_view, 900); function render_view() {
res.render('Admin/allApplications', { title: 'List Of Leave Applications',
csrfToken: req.csrfToken(), hasLeave: hasLeave,
leaves: leaveChunks,
employees: employeeChunks, moment: moment, userName:


req.session.user.name
});









});


}
});



/**
*     Description:
```

```
*/

router.get('/respond-application/:leave_id/:employee_id', function
respondApplication(req, res, next) {

var leaveID = req.params.leave_id;

var employeeID = req.params.employee_id; Leave.findById(leaveID, function
getLeave(err, leave) {

if (err) {

console.log(err);

}


User.findById(employeeID, function getUser(err, user) { if (err) {

console.log(err);

}

res.render('Admin/applicationResponse', { title: 'Respond Leave Application',
csrfToken: req.csrfToken(),

leave: leave, employee: user,

moment: moment, userName: req.session.user.name

});




})




});




});


/**

*     Description:

*/

router.get('/employee-profile/:id', function getEmployeeProfile(req, res, next)
{

var employeeId = req.params.id; User.findById(employeeId, function getUser(err,
user) {

if (err) {

console.log(err);

}
```

```
res.render('Admin/employeeProfile', { title: 'Employee Profile', employee: user,
csrfToken: req.csrfToken(), moment: moment,
userName: req.session.user.name
});


});
});


/**
*     Description:
*/
router.get('/edit-employee/:id', function editEmployee(req, res, next) {
var employeeId = req.params.id; User.findById(employeeId, function getUser(err,
user) {
if (err) {


res.redirect('/admin/');
}
res.render('Admin/editEmployee', { title: 'Edit Employee', csrfToken:
req.csrfToken(), employee: user,
moment: moment, message: '',
userName: req.session.user.name
});




});
});


/**
*     Description:


*
*     Known Bugs: None
*/
router.get('/edit-employee-project/:id', function editEmployeeProject(req, res,
next) {
var projectId = req.params.id;
```

```
Project.findById(projectId, function getProject(err, project) { if (err) {

console.log(err);

}

res.render('Admin/editProject', { title: 'Edit Employee', csrfToken:
req.csrfToken(), project: project,

moment: moment, message: '',

userName: req.session.user.name

});




});


});

/**

*     Description:

*     Gets the id of the employee from parameters.

*     Displays the add employee project form to the admin.

*

*     Known Bugs: None

*/

router.get('/add-employee-project/:id', function addEmployeeProject(req, res,
next) {


var employeeId = req.params.id; User.findById(employeeId, function getUser(err,
user) {

if (err) {

res.redirect('/admin/');

}

res.render('Admin/addProject', { title: 'Add Employee Project', csrfToken:
req.csrfToken(), employee: user,

moment: moment, message: '',

userName: req.session.user.name

});

});


});
```

```
/**
*     Description:
*     First finds project in the Project Schema with the help of id from the
parameters.
*     Gets the Employee of the project.
*
*     Known Bugs: None
*/
router.get('/employee-project-info/:id', function viewEmployeeProjectInfo(req,
res, next) {
var projectId = req.params.id;
Project.findById(projectId, function getProject(err, project) { if (err) {
console.log(err);



user) {


}
User.findById(project.employeeID, function getUser(err,
if (err) {
console.log(err);


}
res.render('Admin/projectInfo', {
title: 'Employee Project Information', project: project,
employee: user, moment: moment, message: '',
userName: req.session.user.name, csrfToken: req.csrfToken()
});
})
});


});


/**
*     Description:
*     Redirects admin to the employee profile page.
*     Known Bugs: None
```

```
*/
router.get('/redirect-employee-profile', function viewEmployeeProfile(req, res,
next) {
var employeeId = req.user.id;
User.findById(employeeId, function getUser(err, user) { if (err) {
console.log(err);
}
res.redirect('/admin/employee-profile/' + employeeId);
});

});
/**
*     Description:
*     Displays the admin its own attendance sheet
*     Known Bugs: None
*/
router.post('/view-attendance', function viewAttendance(req, res, next) {
var attendanceChunks = []; Attendance.find({
employeeID: req.session.user._id, month: req.body.month,
year: req.body.year
}).sort({_id: -1}).exec(function viewAttendanceSheet(err, docs)
{
var found = 0;
if (docs.length > 0) { found = 1;
}
for (var i = 0; i < docs.length; i++) { attendanceChunks.push(docs[i]);
}
res.render('Admin/viewAttendanceSheet', { title: 'Attendance Sheet',
month: req.body.month, csrfToken: req.csrfToken(), found: found,
attendance: attendanceChunks, userName: req.session.user.name, moment: moment
});
});

});
/**
*     Description:
*     Known Bugs: None
```

```
*/

router.get('/view-attendance-current', function
viewCurrentlyMarkedAttendance(req, res, next) {

var attendanceChunks = [];

Attendance.find({

employeeID: req.session.user._id, month: new Date().getMonth() + 1, year: new
Date().getFullYear()

}).sort({_id: -1}).exec(function getAttendanceSheet(err, docs) { var found = 0;

if (docs.length > 0) { found = 1;

}

for (var i = 0; i < docs.length; i++) { attendanceChunks.push(docs[i]);

}

res.render('Admin/viewAttendanceSheet', { title: 'Attendance Sheet',

month: new Date().getMonth() + 1, csrfToken: req.csrfToken(), found: found,

attendance: attendanceChunks, moment: moment,

userName: req.session.user.name

});

});


});

/**

*     Description:

*     Displays the attendance sheet of the given employee to the admin.

*/

router.get('/view-employee-attendance/:id', function viewEmployeeAttendance(req,
res, next) {

var attendanceChunks = [];

Attendance.find({employeeID: req.params.id}).sort({_id: - 1}).exec(function
getAttendanceSheet(err, docs) {

var found = 0;

if (docs.length > 0) { found = 1;

}

for (var i = 0; i < docs.length; i++) { attendanceChunks.push(docs[i]);

}


User.findById(req.params.id, function getUser(err, user) {

res.render('Admin/employeeAttendanceSheet', { title: 'Employee Attendance
Sheet', month: req.body.month,
```

```
csrfToken: req.csrfToken(), found: found,
attendance: attendanceChunks, moment: moment,
userName: req.session.user.name
,
'employee_name': user.name
})
});
});



});
/**
*     Description:

*/
router.post('/add-employee', passport.authenticate('local.add- employee', {
successRedirect: '/admin/redirect-employee-profile', failureRedirect:
'/admin/add-employee', failureFlash: true,
}));
/**
*     Description:

*/
router.post('/respond-application', function respondApplication(req, res) {
Leave.findById(req.body.leave_id, function getLeave(err, leave)
{
leave.adminResponse = req.body.status; leave.save(function saveLeave(err) {
if (err) {
console.log(err);
}
res.redirect('/admin/leave-applications');
})
})


});
router.post('/edit-employee/:id', function editEmployee(req, res) { var
employeeId = req.params.id;
```

```
var newUser = new User(); newUser.email = req.body.email;

if (req.body.designation == "Accounts Manager") { newUser.type =
"accounts_manager";

}

else if (req.body.designation == "Project Manager") { newUser.type =
"project_manager";

}

else {

newUser.type = "employee";

}

newUser.name = req.body.name,

newUser.dateOfBirth = new Date(req.body.DOB), newUser.contactNumber =
req.body.number, newUser.department = req.body.department;

newUser.Skills = req.body['skills[]']; newUser.designation =
req.body.designation;


User.findById(employeeId, function getUser(err, user) { if (err) {

res.redirect('/admin/');

}

if (user.email != req.body.email) {

User.findOne({'email': req.body.email}, function getUser(err, user) {

if (err) {

res.redirect('/admin/');

}

if (user) {

res.render('Admin/editEmployee', { title: 'Edit Employee', csrfToken:
req.csrfToken(), employee: newUser,

moment: moment,

message: 'Email is already in use', userName: req.session.user.name

});

}

});

}

user.email = req.body.email;

if (req.body.designation == "Accounts Manager") { user.type =
"accounts_manager";

}
```

```
else if (req.body.designation == "Project Manager") { user.type =
"project_manager";

}

else {

user.type = "employee";

}

user.name = req.body.name,

user.dateOfBirth = new Date(req.body.DOB), user.contactNumber = req.body.number,


user.department = req.body.department; user.Skills = req.body['skills[]'];
user.designation = req.body.designation;


user.save(function saveUser(err) { if (err) {

console.log(error);

}

res.redirect('/admin/employee-profile/' + employeeId);


});

});


});

router.post('/add-employee-project/:id', function addEmployeeProject(req, res) {

var newProject = new Project(); newProject.employeeID = req.params.id;
newProject.title = req.body.title; newProject.type = req.body.type;

newProject.startDate = new Date(req.body.start_date), newProject.endDate = new
Date(req.body.end_date), newProject.description = req.body.description,
newProject.status = req.body.status;

newProject.save(function saveProject(err) { if (err) {

console.log(err);

}

res.redirect('/admin/employee-project-info/' + newProject._id);

});

});

router.post('/edit-employee-project/:id', function editEmployeeProject(req, res)
{

var projectId = req.params.id; var newProject = new Project();

Project.findById(projectId, function (err, project) { if (err) {

console.log(err);

}
```

```
project.title = req.body.title; project.type = req.body.type;
project.startDate = new Date(req.body.start_date), project.endDate = new
Date(req.body.end_date), project.description = req.body.description,
project.status = req.body.status;
project.save(function saveProject(err) { if (err) {
console.log(err);




projectId);
});


}
res.redirect('/admin/employee-project-info/' +


});
});



router.post('/delete-employee/:id', function deleteEmployee(req, res) {
var id = req.params.id;
User.findByIdAndRemove({_id: id}, function deleteUser(err) { if (err) {
console.log('unable to delete employee');
}
else {
res.redirect('/admin/view-all-employees');




});


}
});

router.post('/mark-attendance', function markAttendance(req, res, next) {

Attendance.find({
```

```
employeeID: req.session.user._id, date: new Date().getDate(), month: new
Date().getMonth() + 1, year: new Date().getFullYear()

}, function getAttendance(err, docs) { var found = 0;

if (docs.length > 0) { found = 1;

}

else {

var newAttendance = new Attendance(); newAttendance.employeeID =
req.session.user._id; newAttendance.year = new Date().getFullYear();
newAttendance.month = new Date().getMonth() + 1; newAttendance.date = new
Date().getDate(); newAttendance.present = 1; newAttendance.save(function
saveAttendance(err) {

if (err) {

console.log(err);

}

});

}

res.redirect('/admin/view-attendance-current');

});


});

module.exports = router;

function isLoggedIn(req, res, next) { if (req.isAuthenticated()) {

return next();

}

res.redirect('/');

}



function notLoggedIn(req, res, next) { if (!req.isAuthenticated()) {

return next();

}

res.redirect('/');

}
```