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
“____” _____ 2025 р.

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
## за першим (бакалаврським) рівнем вищої освіти

на тему:

**Програмне забезпечення інформаційної системи побудованої на основі об’єктно-реляційної СУБД PostgreSQL**



Виконав здобувач вищої освіти
IV курсу, групи К1-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____Пуріхова М.В.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
Босько В.В
«____» _____ 2025 р.
Рецензент _____

м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань. 12 *"Інформаційні технології"*
Спеціальність *123 "Комп'ютерна інженерія"*
Освітньо-професійна (освітньо-наукова) програма *" Комп'ютерна інженерія "*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
"____" _____2025 року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Пуріховій Марії Василівні*
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення інформаційної системи побудованої на основі об'єктно-реляційної СУБД PostgreSQL*

2. Керівник роботи *Босько Віктор Васильвич, канд. техн. наук, доцент*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від 17.01.2025 року № 46-02

3. Строк подання роботи до захисту *20.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи. *Дослідження переваг та недоліків застосування ОРСУБД з розробкою системи безпеки ІС на основі СУБД PostgreSQL*

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. Висновки.*

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

| | |
|---|---|
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |

7. Дата видачі завдання « » 2025р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем керування | 10.03.2025 | |
| 2. | Постановка задачі, оформлення ТЗ | 15.03.2025 | |
| 3. | Розробка моделі компонента | 20.03.2025 | |
| 4. | Розробка структур даних | 25.03.2025 | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.03.2025 | |
| 6. | Програмування алгоритмів | 10.04.2025 | |
| 7. | Оформлення ПЗ | 17.05.2025 | |
| 8. | Попередній захист роботи | 20.05.2025 | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |



Дата видачі завдання
«__»_____2025р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання

«__»_____2025р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

# АНОТАЦІЯ

**Пуріхова М.В. Програмне забезпечення інформаційної системи побудованої на основі об'єктно-реляційної СУБД PostgreSQL.**

**123 Комп'ютерна Інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.**

В даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для реалізації застосувань ОРСУБД в розробці ПЗ.

**Метою роботи** є дослідження об'єктно-реляційної моделі даних та її особливостей при побудові ІС.

Об'єкт дослідження - об'єктно-реляційна модель даних, а предмет - реалізація ІС основі цієї моделі.

Дослідження об'єктно-реляційної моделі та власне побудова бази даних проводилися в інструментальній системі управління базами даних PostgreSQL.

Практичним результатом даної роботи є готова функціонуюча база даних ІС з впровадженням системи безпеки.

База даних заповнена реальними даними, протестована та готова для роботи.

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 PostgreSQL.

**Ключові слова:** кібербезпека, веб-сайт, NodeJS, бази даних, ООП, PostgreSQL.

# ABSTRACT

**Purikhova M.V. The software of the cyber security system of the information system built on the basis of the PostgreSQL object-relational DBMS. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.**

In this qualifying bachelor's thesis, software was developed, which is intended for the implementation of DBMS applications in software development.

The purpose of the work is to investigate the object-relational data model and its features in the construction of IS.

The object of research is an object-relational data model, and the subject is the implementation of IC based on this model.

The study of the object-relational model and the actual construction of the database were carried out in the PostgreSQL database management tool.

The practical result of this work is a ready functioning IC database with the implementation of a security system.

The database is filled with real data, tested and ready to work.

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 PostgreSQL database.

**Keywords:** cyber security, website, NodeJS, databases, OOP, PostgreSQL.

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

JSON - JavaScript Object Notation

MIME - Multipurpose Internet Mail Extensions

REST - Representational State Transfer

MVC - Model-View-Controller

CSS - Cascading Style Sheets

HTML - Hypertext Markup Language

ОС - операційна система

XML - Extensible Markup Language

CMS - Content Management System

ORM - Object-relational mapping

Sass - Syntactically Awesome Stylesheets

DOM - Document Object Model

EJS - Embedded JavaScript templating

ОРБД – Об’єктно-реляційна база даних

# ВСТУП

**Актуальність теми**. У сучасному інформаційному суспільстві, де обсяги даних стрімко зростають, ефективне управління та обробка інформації набувають особливої важливості. Традиційні реляційні бази даних, які довгий час залишалися основним засобом зберігання та організації інформації, мають певні обмеження в умовах роботи з великими масивами даних і високими навантаженнями.

Дані є ключовим ресурсом у різних сферах діяльності – від бізнесу до науки. Вибір технології зберігання та управління даними значною мірою визначає якість та ефективність їх обробки. Серед різноманіття систем управління базами даних (СУБД) PostgreSQL виступає як потужне рішення для реалізації сучасних інформаційних систем.

PostgreSQL – це високопродуктивна, відкрита СУБД, що забезпечує розширені можливості управління транзакціями та зберіганням даних. Її висока стандартизація та підтримка SQL роблять її ідеальним вибором для створення переносимих і масштабованих інформаційних систем.

Застосування об'єктно-реляційних СУБД розширює традиційні підходи до зберігання та обробки інформації. Вони забезпечують гнучкість схем, високу швидкодію при роботі з великими обсягами даних і можливість горизонтального масштабування, що дозволяє адаптуватися до зростаючих вимог сучасних інформаційних систем.

**Цілі дослідження.** Дослідження спрямоване на аналіз використання PostgreSQL в інформаційних системах та оцінку її впливу на ефективність обробки даних і надійність інформаційних процесів.

Випускна кваліфікаційна робота фокусується на вивченні об'єктно-реляційних систем управління базами даних (ОРСУБД) та їхнього застосування в сучасних інформаційних системах.

Зокрема, досліджуються:

- переваги та обмеження ОРСУБД;
- вплив на швидкодію та масштабованість систем;
- можливості їх використання в конкретних галузях.

Окрему увагу приділено практичним аспектам впровадження PostgreSQL у реальних проектах, а також порівнянню з традиційними реляційними базами даних.

Дослідження сприятиме розширенню знань у сфері баз даних, а також допоможе визначити найбільш ефективні стратегії використання ОРСУБД для розробки сучасних інформаційних систем.

**Мета роботи** – оцінити потенціал PostgreSQL та сформувати рекомендації щодо її впровадження в конкретні інформаційні системи.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Об'єктно-реляційна система управління базами даних (ОРСУБД) – це реляційна СУБД, яка підтримує технології об'єктно-орієнтованого підходу, такі як об'єкти, класи та спадкування, що реалізовані в структурі баз даних і мові запитів.

PostgreSQL є однією з найрозвиненіших відкритих СУБД у світі та вільно розповсюджується, що робить її реальною альтернативою комерційним базам даних.

Історія розвитку PostgreSQL. Спочатку PostgreSQL був некомерційною СУБД під назвою Postgres, яка розроблялася в Каліфорнійському університеті в Берклі. Її розробку у 1986 році очолив Майкл Стоунбрейкер, який раніше працював над Ingres – ще одним проєктом Берклі, що згодом був придбаний компанією Computer Associates.

Назва "Postgres" походить від "Post Ingres", оскільки в її основі використовувалися напрацювання попереднього проєкту. Стоунбрейкер і його студенти працювали над Postgres до 1994 року, додавши до нього процедури, правила, користувацькі типи даних та інші розширення.

У 1995 році відбулася важлива зміна:

- Стоунбрейкер застосував отриманий досвід для створення комерційної СУБД Illustra;
- його студенти розробили оновлену версію Postgres – Postgres95, у якій мову запитів POSTQUEL (спадщину Ingres) замінили на SQL;
-розробка вийшла за межі університету та перейшла до команди ентузіастів.

Саме з цього моменту система отримала свою сучасну назву – PostgreSQL.

**Переваги PostgreSQL:**

- висока надійність і стійкість – має репутацію стабільної та надійної

СУБД, що важливо для систем із високими вимогами до доступності та цілісності даних;

- підтримка ACID – гарантує атомарність, узгодженість, ізольованість і надійність транзакцій, що забезпечує точну та безпечну обробку даних;

- розширені можливості – підтримує складні типи даних, спадковість, збережені процедури, тригери, що полегшує реалізацію складної бізнес-логіки;

- масштабованість – PostgreSQL здатна ефективно обробляти великі обсяги даних і велику кількість одночасних з'єднань, що робить її придатною для масштабних проєктів;

- безпека – пропонує гнучку систему ролей і дозволів, шифрування даних, а також безпечну передачу інформації.

**Недоліки PostgreSQL:**

- вартість масштабування – хоча PostgreSQL підтримує горизонтальне масштабування, це може бути складнішим і дорожчим у порівнянні з деякими комерційними СУБД (наприклад, Oracle) або розподіленими базами даних;

- складність управління – для налаштування, оптимізації та моніторингу PostgreSQL потрібен кваліфікований персонал із глибокими знаннями в області баз даних;

- ресурсоємність у сценаріях із високими навантаженнями PostgreSQL може споживати значні ресурси, що може призвести до зростання витрат на обладнання;

- обмеження підтримки – хоча спільнота PostgreSQL дуже активна, для нішевих застосувань може бути менше готових рішень у порівнянні з комерційними СУБД.

PostgreSQL є потужною, надійною та функціональною об'єктно-реляційною СУБД, яка успішно конкурує з комерційними рішеннями. Її відкритий код, висока продуктивність та підтримка SQL-стандартів роблять її чудовим вибором для розробки критично важливих систем. Однак для ефективного використання PostgreSQL необхідні висока кваліфікація

адміністраторів і правильне налаштування для роботи з великими обсягами даних.

## 1.1 Призначення системи

У сучасному цифровому світі, де щодня генеруються величезні обсяги даних у режимі реального часу, питання зберігання, управління та аналізу інформації набуває все більшої актуальності. Швидке зростання кількості даних створює не лише виклики, а й відкриває нові можливості для розвитку інформаційних систем.

Інформаційні системи (ІС) відіграють ключову роль у сучасному суспільстві, визначаючи ефективність і конкурентоспроможність підприємств та організацій.

Основне призначення ІС – це обробка, зберігання та передача інформації для підтримки прийняття рішень та оптимізації організаційних процесів. Завдяки цьому інформаційні системи забезпечують ефективне управління ресурсами та сприяють удосконаленню бізнес-процесів у різних галузях.

**Роль інформаційних систем у сучасному бізнесі**

**Автоматизація бізнес-процесів**

Однією з ключових функцій інформаційних систем (ІС) є автоматизація бізнес-процесів. Вони допомагають оптимізувати внутрішні та зовнішні операції підприємства, зменшують час виконання завдань та підвищують точність операцій. Завдяки ІС можна автоматизувати фінансовий облік, управління запасами, обробку замовлень та багато інших аспектів діяльності.

**Підтримка прийняття рішень**

ІС забезпечують аналіз даних, розробку звітів та прогнозування, що сприяє ефективному управлінню ресурсами та стратегічному плануванню. Вони надають актуальну інформацію, необхідну для прийняття тактичних і стратегічних рішень.

**Забезпечення доступу до інформації**

Інформаційні системи забезпечують швидкий та структурований доступ до інформації для різних користувачів. Це включає:

- зручні інтерфейси;
- ефективний обмін даними;
- гарантовану конфіденційність і безпеку інформації;
- Впровадження електронного бізнесу.

ІС відіграють ключову роль у розвитку електронного бізнесу, сприяючи інтеграції компаній із клієнтами, партнерами та постачальниками через Інтернет. Вони дозволяють:

- створювати та підтримувати електронні торгові майданчики;
- автоматизувати онлайн-транзакції;
- швидко реагувати на зміни ринку;
- Забезпечення безпеки інформації.

ІС містять засоби захисту конфіденційності, цілісності та доступності даних. Вони допомагають запобігати кіберзагрозам, виявляти потенційні загрози та гарантувати захищене зберігання та передачу інформації.

**Управління взаємодією з клієнтами**

Сучасні ІС включають CRM-системи, які допомагають зберігати, аналізувати та використовувати дані про клієнтів. Це дає змогу:

- підвищити рівень обслуговування;
- автоматизувати процеси комунікації;
- покращити персоналізацію сервісу.

**База даних як основа інформаційних систем**

Бази даних є ключовим компонентом сучасних інформаційних систем. У світі стрімкого зростання обсягів даних, що генеруються у різних сферах (медицина, фінанси, наука, соціальні мережі тощо), виникає необхідність у гнучких і потужних інструментах для їх обробки та зберігання.

Сучасні бази даних повинні підтримувати різноманітні типи інформації, включаючи:

- текст;
- графи;
- геодані;
- зображення, відео та інші мультимедійні формати.

**Метою цієї роботи** є розробка інформаційної системи для автоматизації HR-процесів із використанням сучасних технологій та об'єктно-реляційних систем управління базами даних (ОРСУБД).

Особливу увагу приділено узгодженню даних на етапі проектування інформаційної системи, що забезпечить її ефективність, надійність і масштабованість.

Традиційні реляційні бази даних мають обмеження у роботі з різнорідними типами інформації. ОРСУБД, такі як PostgreSQL, надають нові можливості для обробки та управління великими масивами складних даних, що робить їх перспективним вибором для сучасних інформаційних систем.



## 1.2 Область застосування

Застосування об'єктно-реляційних баз даних у сучасних інформаційних системах. Об'єктно-реляційні бази даних (ОРБД) знайшли широке застосування в різних галузях, таких як фінанси, медицина, телекомунікації, аналітика даних, Інтернет речей (IoT), веб-розробка тощо. Дослідження їхнього використання в інформаційних системах допомагає розробникам та архітекторам краще зрозуміти, як ці технології можуть бути ефективно застосовані для різних завдань.

**Головні напрямки застосування ОРБД:**

- веб-застосунки та соціальні мережі. Великі веб-сайти та соціальні платформи потребують швидкого доступу до даних та масштабованості. ОРБД

забезпечують гнучкість схем, що дозволяє швидко оновлювати структуру даних, а також ефективно обробляти великі обсяги інформації;

- аналітика та обробка великих даних. ОРБД використовуються для аналітичних обчислень і роботи з Big Data. Завдяки можливостям горизонтального масштабування, вони дозволяють швидко аналізувати великі обсяги інформації та отримувати цінні інсайти;

- Інтернет речей (IoT). Об'єктно-реляційні бази даних ефективно зберігають та обробляють дані, отримані з сенсорів та IoT-пристроїв. Завдяки гнучким схемам вони дозволяють інтегрувати різні типи даних, а масштабованість допомагає обробляти мільйони одночасних записів;

- електронна комерція. ОРБД широко використовуються в e-commerce, де важливо обробляти великі обсяги даних про товари, клієнтів і транзакції. Вони забезпечують високу продуктивність, безпеку та можливість масштабування відповідно до потреб бізнесу;

- геопросторові дані. У сферах картографії, геонавігації, агротехнологій та інших геоорієнтованих галузях ОРБД дозволяють зберігати та ефективно обробляти геопросторові дані, підтримуючи складні географічні запити;

- логістика та управління ланцюгом постачання. Важливою задачею логістики є відстеження товарів, управління запасами та аналіз потоків поставок. ОРБД дозволяють швидко виконувати складні запити, аналізувати маршрути та оптимізувати процеси поставок.

Дослідження та програмна реалізація системи аналізу застосування об'єктно-реляційних баз даних у сучасних інформаційних системах є актуальною науковою та технічною задачею. Це питання потребує детального вивчення та вирішення у межах даної кваліфікаційної роботи.

**Основні можливості ОРСУБД PostgreSQL**

PostgreSQL є однією з найпотужніших об'єктно-реляційних систем управління базами даних (ОРСУБД), що підтримує розширену функціональність для ефективного управління та обробки даних.

Функції в PostgreSQL виконуються на сервері, а не на клієнті БД, що дозволяє підвищити продуктивність та зменшити навантаження на клієнтську частину.

Функції можна писати на різних мовах програмування:

- PL/pgSQL – процедурна мова, схожа на PL/SQL у Oracle;
- скриптові мови – PL/Lua, PL/Perl, PL/Python, PL/Ruby, PL/Tcl, PL/Scheme;
- класичні мови – C, C++, Java (через модуль PL/Java);
- статистична мова R – (через модуль PL/R).

Функції можуть повертати набір записів, що дозволяє їх використовувати як результати SQL-запитів. Можливе виконання функцій з правами їхнього творця або поточного користувача. Відмінність від збережених процедур – функції виконуються в межах транзакції, тоді як збережені процедури можуть запускати власні транзакції.

Тригери – це функції, які виконуються автоматично при певних DML-операціях (INSERT, UPDATE, DELETE).

використовуються для перевірки даних, автоматичних оновлень та контролю цілісності. Можуть бути написані на різних мовах програмування. Множинні тригери виконуються в алфавітному порядку.

**Індекси. PostgreSQL підтримує кілька типів індексів:**

- B-дерево – основний тип для швидкого пошуку;
- хеш-індекси – для рівних порівнянь (=);
- R-дерево для геопросторових даних;
- GiST, GIN – для складних пошукових запитів;
- можливість використання декількох індексів одночасно для складних запитів;
- функціональні індекси – базуються на значеннях функцій від полів. Часткові індекси –зменшують обсяг індексних структур.

**Підтримка Multiversion Concurrency Control (MVCC)**

- механізм MVCC дозволяє кільком користувачам одночасно змінювати базу даних без блокування читання;
- гарантує відповідність ACID-транзакціям;
- зменшує конфлікти між запитами та підвищує продуктивність.

PostgreSQL – це потужна об'єктно-реляційна СУБД, яка поєднує гнучкість, продуктивність та розширені можливості. Її механізм індексування, тригери, правила, підтримка різних мов програмування та MVCC роблять її чудовим вибором для масштабованих і високонавантажених систем.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

Автоматизація HR-процесів динамічно розвивається, що спричинено цифровізацією та змінами у сфері управління персоналом. Багато компаній прагнуть оптимізувати кадрові завдання, роблячи їх більш ефективними та менш ресурсозатратними.

Системи автоматизації HR набирають популярності та пропонують широкий функціонал, що включає:

- ведення електронних таблиць;
- управління навчальними матеріалами;
- автоматизований документообіг;
- аналіз продуктивності персоналу.

**Діджиталізація HR як необхідність**

Цифрові рішення поступово перетворюються з додаткової опції на обов'язковий елемент сучасного HR. Впровадження автоматизованих систем дозволяє розвантажити спеціалістів, звільнивши їх від рутинної роботи та сконцентрувавшись на стратегічних завданнях, таких як мотивація персоналу та професійний розвиток співробітників.

**Виклики та перспективи**

Потреба в автоматизації управлінських HR-задач з'явилася відносно недавно, тому цей напрямок ще знаходиться в стадії розвитку.

До вибору HR-системи варто підходити ретельно, враховуючи:

- специфіку компанії – кожна організація має унікальні алгоритми управління;
- гнучкість системи – можливість адаптації під конкретні HR-завдання;
- аналітичні можливості – система повинна підтримувати збір, обробку та аналіз інформації про співробітників;

**Автоматизація HR-процесів та сучасні підходи до управління даними**

Автоматизація HR-процесів відкриває нові можливості для бізнесу, сприяючи оптимізації управління персоналом та підвищенню ефективності компанії. У міру розвитку технологій кадрові процеси стають цифровізованими, а їх інтеграція у корпоративну культуру поступово стає стандартом.

**Основні виклики автоматизації HR**

HR-фахівці стикаються з високою складністю управління, оскільки їм доводиться керувати великою кількістю завдань і процесів. Впровадження автоматизованих систем дозволяє:

- формалізувати та оптимізувати всі HR-процеси;
- здійснювати моніторинг та контроль діяльності персоналу;
- зменшити кількість помилок, порушень та зловживань.

Саме тому розробка інформаційної системи для автоматизації кадрових процесів є актуальним напрямком наукових досліджень.

**Важливість вибору СУБД для HR-систем**

При створенні кадрових систем важливу роль відіграє серверна частина, яка забезпечує зберігання інформації. Як правило, використовується СУБД, що забезпечує структуроване збереження та ефективну обробку даних.

Сучасні тенденції свідчать про зростаючу увагу до ОРБД (об'єктно-реляційних баз даних), оскільки вони поєднують гнучкість та масштабованість реляційних та NoSQL-рішень.

**Проблеми реляційних СУБД у масштабованих системах**

Реляційні СУБД традиційно зберігають дані у вигляді таблиць із фіксованою схемою. Однак у великих масштабних системах (Big Data) виникають труднощі:

- складність забезпечення відмовостійкості при великій кількості вузлів;
- зниження продуктивності через централізоване зберігання даних.

**NoSQL як альтернатива реляційним БД**

Як рішення проблем масштабованості реляційних СУБД виникли NoSQL-

бази даних. Вони вперше були розроблені такими компаніями, як Google та Amazon, для роботи з великими розподіленими системами.

Основні переваги NoSQL:

- використання багаторазової реплікації для високої відмовостійкості;
- гнучка схема зберігання даних;
- висока швидкість обробки неструктурованої інформації;

Основні недоліки NoSQL:

- відсутність повної підтримки транзакцій ACID;
- проблеми узгодженості даних через асинхронну реплікацію4
- потреба у специфічних математичних моделях, які ще не до кінця розроблені.

**Ключові параметри оцінки систем NoSQL**

При проектуванні HR-систем на основі NoSQL необхідно оцінювати:

- ймовірність читання застарілих даних під час розподілу оновлень;
- час очікування доступу до оновлених записів;
- кількість версій одного запису в різних вузлах;
- ймовірність заборони доступу до певних даних;

Ці параметри важливо враховувати на етапі проєктування, щоб уникнути ручного налаштування системи на етапі її розгортання та масштабування.

Хоча NoSQL-технології активно розвиваються, відсутність адекватних математичних моделей для оцінки узгодженості даних залишається відкритим питанням. Це підтверджує необхідність подальших досліджень у сфері управління базами даних, особливо для HR-автоматизації та великих інформаційних систем.

## 2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським рівнем вищої освіти)

**Система автоматизації кадрових процесів BambooHR**

BambooHR – це одна з найпопулярніших HR-систем для управління персоналом, яка орієнтована на малий та середній бізнес. Вона пропонує комплексний підхід до автоматизації кадрових процесів та має широкий набір функцій.

Основні можливості BambooHR:

- облік персоналу – централізоване зберігання інформації про співробітників (контакти, посади, робочі графіки, зарплати тощо);
- управління рекрутингом – інструменти для створення вакансій, відстеження кандидатів та автоматизації підбору персоналу;
- онбординг та адаптація – налаштування процесів введення нових співробітників у робочий процес;
- моніторинг робочого часу – відстеження відвідуваності, відпусток і лікарняних;
- оцінка продуктивності – функціонал для збору відгуків, аналізу результатів роботи та постановки цілей;
- звіти та аналітика – автоматичне формування HR-звітів та прогнозування тенденцій у компанії;
- мобільний додаток – зручний доступ до HR-інформації з будь-якого пристрою.

Переваги BambooHR:

- простий та інтуїтивно зрозумілий інтерфейс – система легко освоюється навіть без спеціальних знань;
- гнучка система налаштувань – можливість адаптації під потреби компанії;

- інтеграція з іншими сервісами – підтримка API для зв’язку з бухгалтерськими та фінансовими системами;
- захист персональних даних – підтримка сучасних стандартів безпеки;
- хмарне рішення – доступ до системи з будь-якого пристрою без необхідності встановлення ПЗ.

**Недоліки BambooHR**

- обмежений функціонал для великих компаній – може не підходити для масштабних організацій із складною структурою;
- відсутність глибокої кастомізації – деякі процеси не можна гнучко налаштувати під специфіку бізнесу;
- вартість – немає безкоштовної версії, а ціни можуть бути високими для невеликих компаній;
- обмежені можливості для payroll (зарплатних розрахунків) – потрібна інтеграція з іншими фінансовими сервісами.

BambooHR – це зручне рішення для автоматизації кадрових процесів малих та середніх компаній. Вона спрощує управління персоналом, допомагає оптимізувати HR-процеси та підвищує ефективність роботи відділу кадрів.

Проте для великих корпорацій або компаній із складною структурою можуть знадобитися більш гнучкі та функціональні HR-системи головне вікно якої показано на рисунку 2.1.

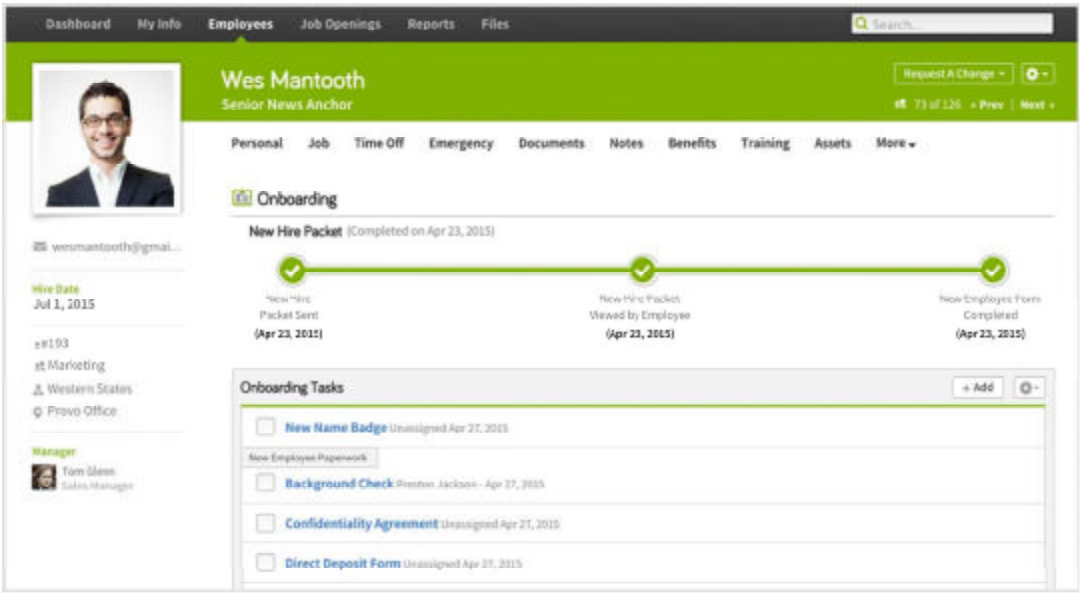



Рисунок 2.1 - Система BambooHR

BambooHR – це зручне та потужне рішення для автоматизації HR-процесів у малому та середньому бізнесі. Вона забезпечує оптимізацію управління персоналом, спрощує підбір, адаптацію, навчання та адміністрування співробітників.

Однак відсутність автоматизації оцінювання персоналу може стати суттєвим обмеженням для компаній, які активно використовують системи рейтингів та історію продуктивності.

**Hurma: сучасна система автоматизації HR-процесів**

Hurma – це нове комплексне рішення, яке поєднує процеси рекрутингу, HR та OKR (цілей та ключових результатів). Система пропонує гнучкі пакети для компаній різного розміру, підходячи як для малих команд, так і для великих підприємств.

Основні особливості Hurma:

- моніторинг настрою працівників – збір даних про задоволеність персоналу;
- організація структури компанії у вигляді дерева – візуальне представлення корпоративної ієрархії;

- синхронізація з Google Calendar – автоматичні нагадування про події компанії;
- особистий кабінет працівника – доступ до персональної інформації, графіку роботи, документів;
- автоматизація перевірки ефективності – контроль KPI, підтримка випробувальних термінів, привітань, вихідних співбесід;
- телефонний зв'язок із кадрами та керівництвом – швидке спілкування через систему;
- робота з OKR – постановка та контроль виконання цілей та ключових результатів;
- управління відсутністю – автоматизоване оформлення відпусток, лікарняних, дистанційної роботи;
- база вакансій та кандидатів – зручне зберігання інформації про відкриті вакансії та потенційних співробітників;
- публікація вакансій на сайті компанії – швидке розміщення нових робочих пропозицій;
- інтеграція з LinkedIn – автоматичний імпорт профілів кандидатів;
- ведення статистики процесу працевлаштування – аналітика щодо підбору персоналу та кадрових процесів.

**Недоліки Hurma**

- фільтри кандидатів обмежені – немає можливості відбирати кандидатів за певними навчальними курсами;
- обмежене сортування вакансій – можлива тільки фільтрація за назвою та датою створення;
- відсутність коментарів у завданнях – неможливо залишати нотатки щодо затвердження відпусток, лікарняних або дистанційної роботи.

Hurma – це інноваційна HR-система, яка автоматизує підбір персоналу, кадровий менеджмент та контроль ефективності працівників. Вона відмінно підходить для компаній, які прагнуть інтегрувати OKR-підхід у HR-управління.

Однак обмежені можливості фільтрації кандидатів та відсутність коментарів у деяких завданнях можуть викликати труднощі в HR-відділах із високою кадровою динамікою.

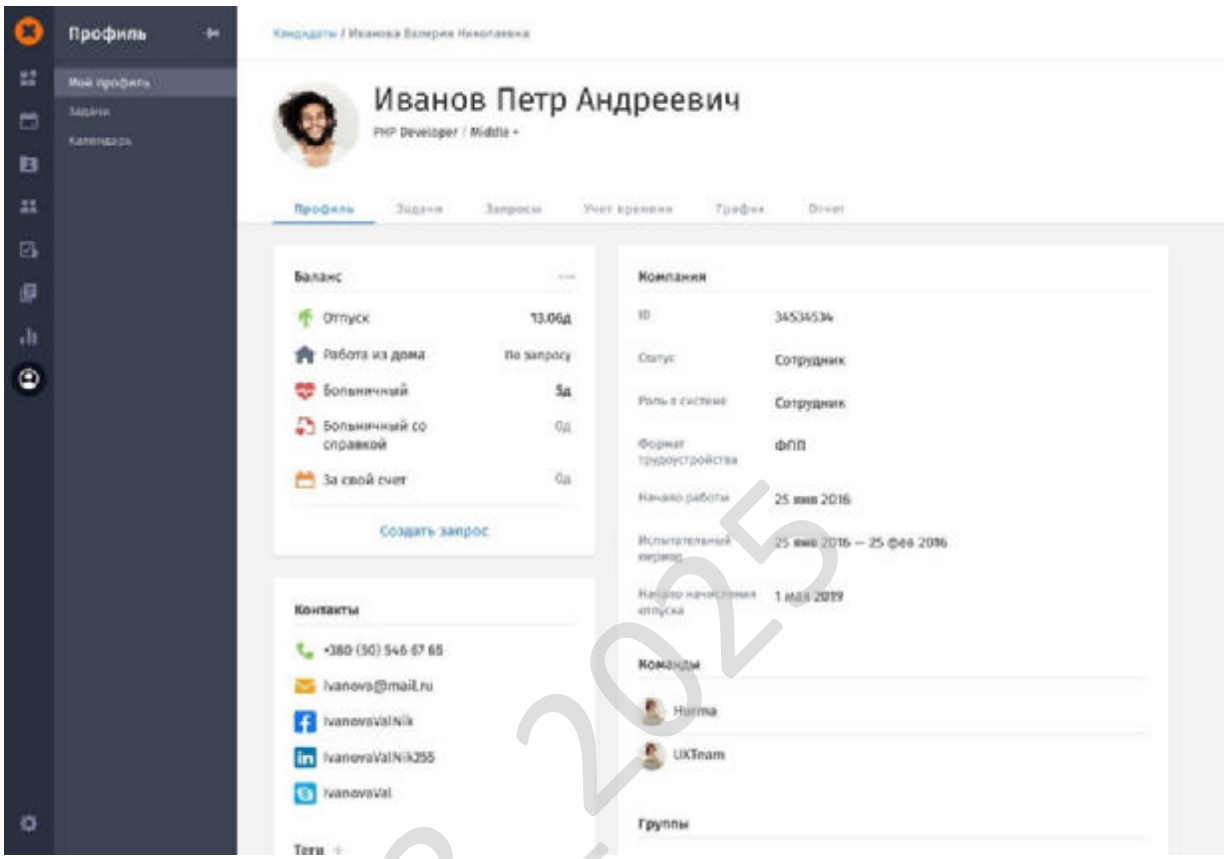



Рисунок 2.2 - Система Hurma System

**Workable: Сучасна система управління підбором персоналу**

Workable – це потужна платформа для автоматизації рекрутингових процесів, яка поєднує систему відстеження кандидатів (ATS) та інструменти підбору персоналу. Вона використовує машинне навчання для оптимізації процесу пошуку та найму співробітників.

**Основні функціональні можливості Workable:**

- швидкий підсумковий аналіз – автоматичний аналіз кандидатів та вакансій;
- автоматизація пошуку співробітників – вбудований конструктор анкет, резюме та вакансій;

- інтеграція з порталами працевлаштування – швидке розміщення вакансій на Job-платформах;
- інтеграція з LinkedIn та соціальними мережами – можливість автоматизованого пошуку кандидатів;
- брендування інтерфейсу – кастомізація дизайну відповідно до фірмового стилю компанії;
- синхронізація календаря та розкладу HR-відділу – покращений контроль за графіком інтерв'ю та зустрічей;
- імпорт існуючих баз даних – можливість перенесення попередніх рекрутингових даних;
- персональний менеджер та підтримка – служба підтримки через телефон, e-mail;
- мобільний додаток – зручний доступ до системи з будь-якого пристрою;
- розширення Google Chrome – швидкий пошук кандидатів та їхніх даних безпосередньо у браузері.

Workable – це функціональна та технологічна HR-система, яка автоматизує пошук і підбір персоналу. Її інтеграція з соцмережами, порталами працевлаштування та аналітичні інструменти значно спрощують процес рекрутингу.

Це ідеальне рішення для компаній, які шукають ефективний спосіб масштабувати HR-процеси та оптимізувати підбір кадрів за допомогою сучасних технологій.

**Система Zoho People**

Програмне забезпечення для управління персоналом, створене спеціально для малого та середнього бізнесу. Воно має інтуїтивно зрозумілий інтерфейс і готове до використання без додаткових налаштувань.

Основні можливості Zoho People:

- управління відсутністю співробітників;

- шаблони стандартних кадрових документів;
- доступність у вигляді мобільного додатку та веб-версії;
- контроль робочого часу;
- HR-аналітика та звітність;
- оцінка ефективності працівників;
- можливість використання електронного підпису.

Роль HR-менеджера надзвичайно важлива для успішної роботи компанії. Він виступає посередником між керівництвом та працівниками, допомагає розвивати бренд роботодавця, формує корпоративну культуру, мотивує команду та адаптує нових співробітників. Окрім цього, HR-фахівець займається кадровим документообігом, обліком відпусток, лікарняних, відряджень, організацією співбесід та корпоративних заходів. Від його роботи часто залежить ефективність усієї команди, тому всі завдання потрібно виконувати бездоганно.

Автоматизація процесів дозволяє зменшити рутинне навантаження, підвищити ефективність HR-відділу та приділити більше уваги стратегічним завданням, таким як мотивація персоналу, розвиток корпоративної культури та адаптація нових працівників.

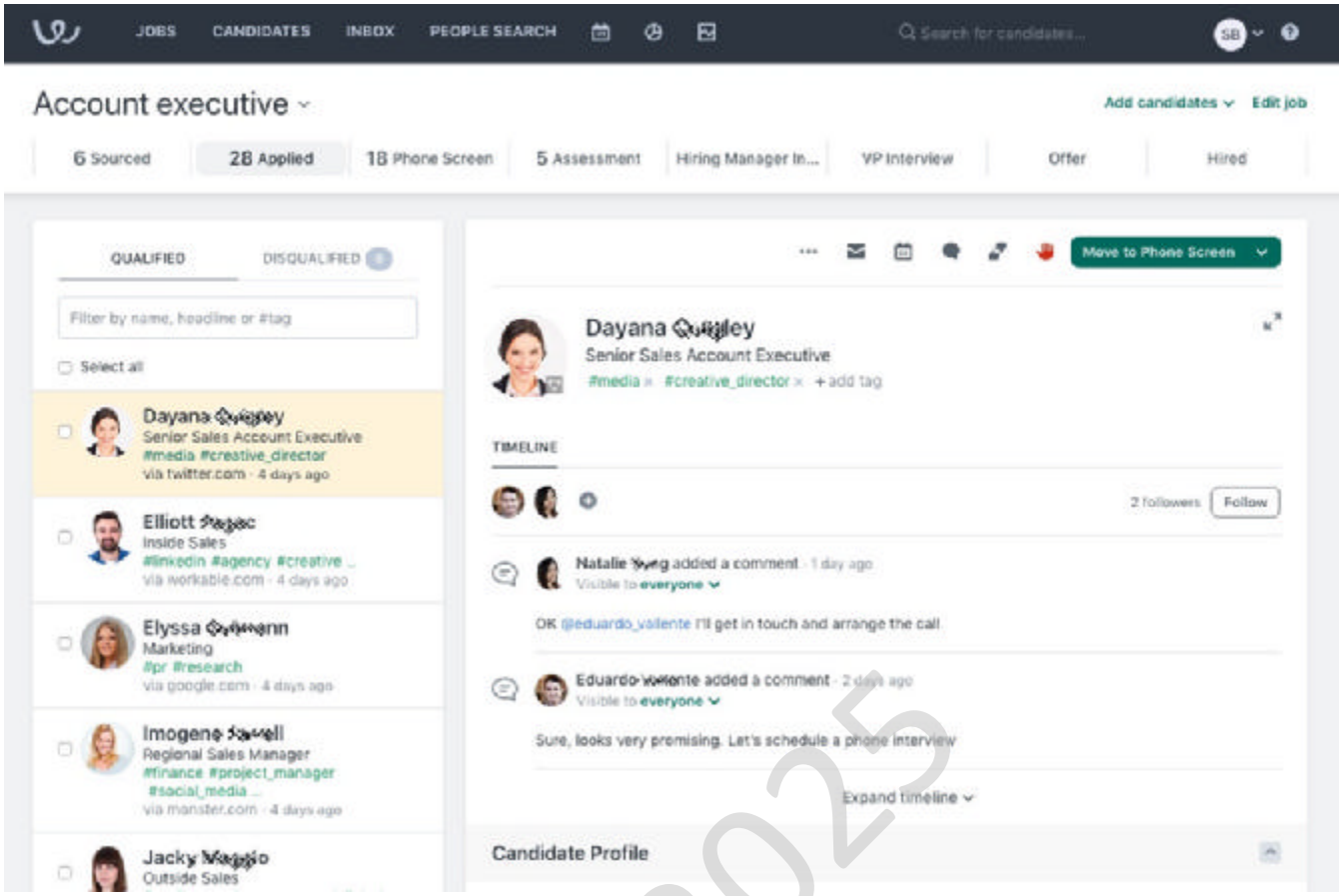



Рисунок 2.3 - Система Workable

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

У процесі аналізу HR-системи було виявлено ключову роль серверної частини, яка забезпечує зберігання та обробку інформації. В якості серверної інфраструктури використовується система управління базами даних (СУБД).

Сучасні тенденції свідчать про зростаючу увагу до об'єктно-реляційних баз даних (ОРБД).

Останні роки у сфері обробки даних домінували реляційні СУБД, у яких інформація зберігається у вигляді таблиць із чітко визначеною структурою. Однак при розробці великих систем (Big Data) використання реляційних підходів створило певні труднощі:

- ускладнена агрегація даних через необхідність зчитування великої кількості взаємопов’язаних таблиць, що спричинило проблему узгодженості даних;
- виник конфлікт між потребою зберігати великі обсяги неструктурованої інформації та необхідністю її структурування шляхом створення складних схем баз даних;
- для обробки великих масивів даних потрібно використовувати дорогі апаратно-програмні комплекси паралельних систем баз даних, такі як Teradata, Sun Oracle Database Machine тощо;
- за наявності значної кількості вузлів постає проблема забезпечення необхідного рівня відмовостійкості системи.

Для вирішення проблем, що накопичилися у реляційних базах даних, було розроблено альтернативні підходи до зберігання та обробки інформації, які отримали назву «об'єктно-реляційні бази даних» (ОРБД). Піонерами в цій сфері стали компанії Google і Amazon.

Одним із ключових аспектів роботи таких систем є координація реплік, яка включає показники ймовірності читання застарілих записів під час розповсюдження оновлень між вузлами, час очікування доступу до оновлених записів, кількість версій даних та швидкість їх обробки, а також ризик обмеження доступу до певних записів. Оцінка цих параметрів ще на етапі проєктування системи дозволяє автоматизувати вибір оптимальних налаштувань та уникнути складних коригувань під час налагодження, особливо при високих навантаженнях.

Попри те, що технології створення інформаційних систем на основі баз даних розвиваються вже давно, математичні моделі для оцінки відповідності реплік або відсутні, або не відповідають сучасним вимогам.

**Основні типи ОРБД та їх функції**

Незважаючи на різноманітність підходів, об'єктно-реляційні бази даних виконують ряд спільних завдань, зокрема:

- управління розміщенням реплік у кластері та їх узгодженням під час оновлення даних;
- координацію версій записів (об’єднання різних версій у єдину);
- відновлення реплік після усунення несправностей у системі.

Для налаштування та оновлення реплік ОРБД зазвичай використовують два основні методи: «головний-підлеглий» і «кільцевий»:

- метод головний-підлеглий передбачає, що всі дані зберігаються на головному сервері, а підлеглі вузли отримують оновлення періодично, зчитуючи їх із головного вузла. Усі зміни спочатку записуються на головному сервері, а потім поширюються по системі;
- кільцевий метод організовує взаємодію вузлів у формі циклу, що дозволяє їм обмінюватися змінами без централізованого керування.

**PostgreSQL та його архітектура**

PostgreSQL реалізує клієнт-серверну модель, де сервер обробляє запити, транзакції та управління доступом, а клієнти підключаються через різні інтерфейси, такі як libpq (офіційна C-бібліотека PostgreSQL), JDBC, ODBC та інші. Система працює в багатопроцесовому режимі, де кожен клієнт обслуговується окремим процесом. Така архітектура забезпечує ізоляцію транзакцій та високу стійкість до збоїв, що критично важливо для збереження цілісності даних.

**Підтримка ACID**

PostgreSQL підтримує принципи ACID завдяки використанню журналу транзакцій, механізму блокування на рівні рядків та багатоверсійного управління конкуренцією (MVCC). Завдяки MVCC система дозволяє виконувати читання даних без блокування записів, що значно покращує продуктивність у високонавантажених транзакційних середовищах. Журнал транзакцій забезпечує можливість відновлення операцій у разі збою або помилки системи, гарантуючи збереження цілісності та надійності даних.

## 2.3 Розгорнута постановка завдання

Згідно з технічним завданням магістерської роботи буде розроблено програмне забезпечення, яке відображає роботу інформаційної системи (ІС) з використанням концепції об'єктно-реляційних баз даних (ОРБД). У процесі виконання проєкту необхідно реалізувати наступні етапи:

a) Провести аналіз існуючих аналогічних систем для визначення їхніх переваг та недоліків. Отримані результати врахувати при подальшій розробці.

b) Обґрунтувати вибір методики побудови системи контролю роботи технологічного обладнання у виробничих умовах в автоматизованому режимі. Розробити функціональну та структурну схеми системи.

c) Створити програмне забезпечення, що відповідатиме вимогам технічного завдання. Розробити блок-схеми алгоритмів програми та її окремих модулів.

d) Реалізувати зручний користувацький інтерфейс, який забезпечить виведення повідомлень про помилки користувача та нестандартні ситуації.

e) Розробити рекомендації щодо організаційних та методичних заходів, необхідних для успішного впровадження та експлуатації системи у промислових умовах.

f) Підготувати висновки щодо виконаного обсягу робіт та отриманих результатів.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

**Аналіз та вибір типу СУБД і мови програмування**

Етап аналізу передбачає вивчення системних вимог і виявлення потенційних проблем. Найбільш точне розуміння цього процесу дають такі терміни, як аналіз вимог (дослідження системних потреб) та об'єктно-орієнтований аналіз (вивчення об'єктів предметної області). В рамках об'єктно-орієнтованого аналізу основна увага приділяється ідентифікації та опису ключових об'єктів (або концепцій) у контексті предметної області.

Аналіз вимог може включати опис основних процесів або сценаріїв використання програмного забезпечення, які можна представити у вигляді конкретних прикладів. Об'єктно-орієнтований підхід, у свою чергу, дозволяє систематизувати предметну область, визначаючи її основні об'єкти, їхні атрибути та взаємозв’язки. Декомпозиція предметної області полягає у виділенні ключових понять, важливих для вирішення завдання, що в результаті формує модель предметної області, представлену у вигляді діаграм, що ілюструють її основні концепції та зв’язки між ними.

Найдоцільнішим підходом є виділення процесів, пов’язавши їх із наявними структурними елементами. Існуючі елементи побудовані за функціональним принципом - вони спрямовані на виконання конкретної задачі чи створення кінцевого продукту. Відповідно, розподіл процесів здійснюється в рамках існуючої системи управління.

Опис процесів у межах предметної області може бути поданий у вигляді словесних прецедентів, оформлених у структурованому форматі. Прецедент - це набір сценаріїв, які моделюють як успішні, так і невдалі варіанти використання системи виконавцем для досягнення певної мети.

На основі запитів зацікавлених сторін були сформовані відповідні функціональні вимоги.

Зокрема, функціональні можливості, пов’язані з керуванням, повинні передбачати такі базові операції, як додавання, редагування та видалення даних.

**Опис інформаційної системи**

У випускній кваліфікаційній роботі розроблена інформаційна система для управління людськими ресурсами. Вона зберігає дані про співробітників, включаючи їхню освіту, досвід роботи, робочі графіки, управління відпустками та участь у розподілених проєктах.

Система містить адміністративну панель, що дозволяє керувати всією інформацією про персонал та надає аналітичні дані, зокрема статистику зайнятості працівників у різних відділах та середній показник плинності кадрів.

Проєкт реалізований з використанням Node Express як бекенд-фреймворку для Node.js, а для інтерфейсу застосовуються HTML, Bootstrap, CSS і JavaScript.

Основні структурні модулі системи:

- облікові записи для адміністраторів та співробітників;
- рівневий доступ до даних, що визначає права користувачів відповідно до їхніх привілеїв;
- реєстрація співробітників та адміністраторів;
- тайм-менеджмент, що дозволяє відстежувати робочий час усіх працівників;
- управління заробітною платою;
- облік загального стажу роботи та професійного досвіду співробітників;
- контроль участі співробітників у розподілених проєктах всередині організації.

**Основні групи користувачів**

Адміністратор має повний доступ до системи:

- реєструє співробітників;

- налаштовує їхні привілеї;
- контролює відвідуваність;
- переглядає та коригує зарплату;
- призначає або змінює проєкти;
- схвалює або відхиляє запити на відпустку.

Співробітник отримує доступ до таких можливостей:

- відмічати свою присутність;
- переглядати історію роботи, зарплатню;
- переглядати свій профіль (освіта, досвід роботи);
- бачити, з ким працює в команді;
- подавати заявки на відпустку та відстежувати їхній статус.

Лідер проєкту має можливість:

- переглядати навички співробітників;
- оцінювати їхню продуктивність;
- переглядати команду проєкту.

Бухгалтер відповідає за фінансові операції:

- створює платіжні відомості;
- встановлює зарплату та бонуси4
- підвищує зарплату;
- надсилає зарплатні виписки працівникам електронною поштою.

**Розробка архітектури інформаційної системи**

Архітектура програмного забезпечення визначає його компоненти, функціональність та взаємодію між ними. Розроблений додаток побудований за клієнт-серверною архітектурою та включає механізми ідентифікації, авторизації користувачів і зберігання даних.

Система працює на основі мережевих запитів, що передаються між клієнтською та серверною частинами.

Принцип роботи клієнт-серверної взаємодії:

- Клієнт формує та надсилає запит на сервер.
- Сервер обробляє запит, виконує необхідні операції з даними та

звертається до бази даних.

- База даних повертає результат запиту серверу.
- Сервер формує відповідь і надсилає її клієнту.
- Клієнт отримує відповідь, відображає її користувачеві та очікує наступних дій.

Цей цикл повторюється, поки користувач не завершить роботу із системою.

**Обґрунтування вибору архітектури**

Клієнт-серверна архітектура була обрана через її масштабованість, гнучкість і можливість віддаленого доступу до системи. Такий підхід забезпечує підтримку великої кількості користувачів та дозволяє керувати ресурсами централізовано.

Система складається з трьох основних частин:

- робота з базою даних – збереження, пошук та оновлення інформації;
- бізнес-логіка – обробка запитів користувачів, валідація даних та управління правами доступу;
- клієнтська частина – інтерфейс користувача для взаємодії з системою;
- обробка та зберігання даних виконується на сервері, а клієнт лише надсилає запити на отримання або зміну інформації.

**Проєктування системи**

Розробка архітектури включає два основні підходи:

1. Об'єктно-орієнтоване проєктування (ООП):
   - Визначення програмних об'єктів та їхньої взаємодії.
   - Створення UML-діаграм, таких як діаграма послідовності для відображення потоку викликів методів між об'єктами.
   - Побудова діаграми класів, яка описує взаємозв’язки між об’єктами.
2. Проєктування бази даних:

- Визначення схем таблиць та їхніх взаємозв’язків.
- Оптимізація запитів для ефективного зберігання та отримання даних.

Запропонована архітектура дозволяє створити ефективну, масштабовану та безпечну інформаційну систему, що відповідає вимогам користувачів та забезпечує високу продуктивність у багатокористувацькому середовищі.

### 3.2 Розробка структурної схеми

Якщо тобі потрібно створити структурну схему сайту, можна зробити це у вигляді ієрархічної діаграми, де кожен елемент відображає певну сторінку або функціональний модуль.

Структурна схема системи включає в себе різні компоненти, які взаємодіють між собою для забезпечення функціональності системи. Нижче наведено короткий опис кожного блоку та їхній взаємодії:

**WEB UI (користувацький інтерфейс):**

– відповідає за візуальне взаємодію з користувачами;

– взаємодіє з сервером через запити, ініціюючи запити на отримання або оновлення даних.

**NodeJS:**

– використовується середовище виконання для серверної сторони;

– обробляє HTTP-запити від WEB та взаємодіє з іншими компонентами системи.

**Express.js:**

– веб-фреймворк для Node.js, який допомагає створювати API;

– обробляє маршрутизацію та керуєзапитами.

**Controller:**

– обробляє бізнес-логіку за дані, отриманими від Express.js;

– керує взаємодією між WEB і базою даних.

**MongoDB:**

– нереляційна база даних, в якій зберігаються дані;

– Controller взаємодіє з MongoDB для зберігання, читання та оновлення даних усих даних.

**Rest:**

– використовується для створення API взаємодії між WEB UI та серверною частиною системи;

– визначає стандарти для обміну даними компонентами через HTTP-протокол.

**JSON:**

– використовується для форматування та усіх обмінів даними між компонентами;

– дані, що передаються між WEB-UI, NodeJS, та базою даних, можуть бути у форматі JSON.

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

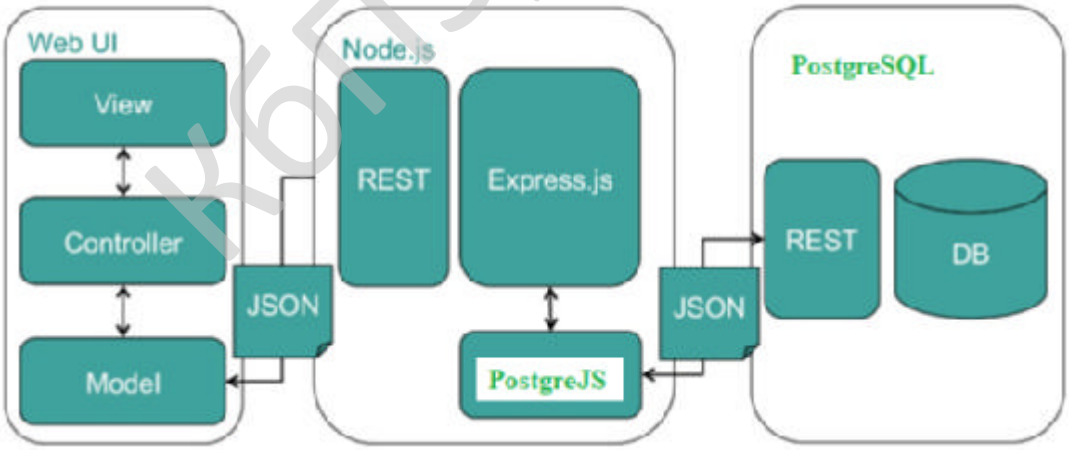



Рисунок 3.1 – Структурна схема

Ця структурна схема ілюструє взаємодію всіх компонентів, що утворюють єдину систему, яка забезпечує роботу веб-додатка, включаючи серверну частину,

базу даних та інші ключові функціональні модулі.

Система складається з різних компонентів, що тісно взаємодіють між собою, забезпечуючи її повноцінну функціональність.

Нижче наведено короткий опис кожного блоку та їхній взаємодії:

WEB UI (користувацький інтерфейс):

– забезпечує візуальну взаємодію користувача із системою;

– обмінюється даними з сервером через HTTP-запити, ініціюючи отримання або оновлення інформації.

**Express.js:**

– веб-фреймворк для Node.js, що дозволяє швидко створювати API;

– відповідає за маршрутизацію та обробку HTTP-запитів.

**Controller:**

– реалізує бізнес-логіку на основі даних, отриманих через Express.js;

– забезпечує взаємодію між WEB UI та базою даних.

**PostgreDB:**

– об’єктно-реляційна база даних для зберігання інформації;

– Controller використовує PostgreDB для запису, читання та оновлення даних.

**REST:**

– застосовується для створення API, що забезпечує зв’язок між WEB UI та серверною частиною;

– визначає стандарти обміну даними через HTTP-протокол.

**JSON:**

– використовується для структурованого обміну даними між компонентами системи;

– формат JSON застосовується при передачі інформації між WEB UI, Node.js та базою даних.

### 3.3 Розробка функціональної схеми

Функціональна схема розробленої системи представлена на рисунку 3.2. Вона допомагає зрозуміти ролі кожного компонента та їхню взаємодію для забезпечення роботи системи як єдиного цілого. Розглянемо детальніше кожен блок.

**User Interface (UI)**

Компонент User Interface відповідає за візуальну взаємодію користувача з веб-додатком.

Він включає набір технологій і ресурсів, що забезпечують зручний та естетично привабливий інтерфейс.

**HTML (HyperText Markup Language)**

Основна мова розмітки для створення структури веб-сторінок:

– використовується для визначення текстових блоків, зображень, форм та інших елементів;

– визначає, які компоненти будуть відображатися на сторінці.

**JavaScript (JS):**

– використовується для створення інтерактивності та динамічної взаємодії з користувачем;

– дозволяє реагувати на події, змінювати вміст сторінки, виконувати валідацію введених даних;

– використовується для взаємодії з сервером через AJAX-запити, забезпечуючи асинхронне завантаження даних.

**CSS (Cascading Style Sheets):**

– відповідає за стилізацію та оформлення HTML-елементів;

– визначає кольорову гаму, розміри, шрифти та розташування елементів на сторінці;

– забезпечує адаптивний дизайн, що підлаштовується під різні пристрої.

**Fonts (Шрифти):**

– вибір шрифтів впливає на зручність читання та візуальну привабливість

інтерфейсу;

– використання різних шрифтів допомагає створити унікальний стиль веб-додатка;

**Images (Зображення):**

– використовуються для покращення візуального сприйняття інформації;

– можуть слугувати фоном, ілюстраціями, логотипами тощо;

– оптимізація зображень допомагає покращити продуктивність веб-додатка.

**Блок RequestManagement**

Цей блок визначає архітектуру та функціональність серверної частини веб-додатка. Він використовує фреймворк Express.js, менеджер маршрутів, контролери, API для взаємодії з клієнтською частиною, бібліотеку Mongoose для роботи з базою даних MongoDB та моделі для структурування даних.

Розглянемо кожен компонент цього блоку.

**Express.js:**

Express.js – це веб-фреймворк для Node.js, який спрощує створення веб-додатків та API;

– маршрутизація – визначає маршрути та їх обробники для HTTP-запитів;

– обробка запитів і відповідей – керує HTTP-запитами, обробляє введені дані та формує відповіді.

**Route Manager (Менеджер маршрутів)**

Відповідає за управління маршрутами в Express.js:

- реєстрація маршрутів – визначає шляхи та обробники для запитів;
- керування потоком запитів – спрямовує запити відповідно до визначених маршрутів.

**Controllers (Контролери)**

Контролери виконують бізнес-логіку додатка на серверному боці:

– обробка запитів – виконує логіку, пов’язану з відповідними маршрутами.

– Взаємодія з моделями – отримує та оновлює дані в базі даних через

моделі.

**API (Application Programming Interface)**

API визначає інтерфейс для взаємодії між компонентами веб-додатка:

– структурування даних – формує стандартизований формат даних для обміну;

– взаємодія між частинами системи – забезпечує отримання та передачу даних;

**Mongoose (Бібліотека для роботи з MongoDB)**

Використовується для моделювання даних у MongoDB:

– визначення схем даних – створює моделі для зберігання та валідації документів;

– робота з базою даних – виконує операції читання, запису, оновлення та видалення даних у PostgreDB.

**Models (Моделі)**

Опис: Моделі в Express.js використовуються для представлення та взаємодії з даними у базі даних.

**Блок Module**

Цей блок об'єднує кілька ключових модулів веб-додатка: Email, ExcelExport, Auth та Tasks, кожен із яких відповідає за виконання окремих функцій у системі.

**Email (Електронна пошта)**

Відповідає за роботу з електронною поштою у веб-додатку.

– відправка електронних листів – забезпечує надсилання повідомлень користувачам;

– обробка сповіщень – взаємодіє з іншими модулями для автоматичної відправки повідомлень;

**ExcelExport (Експорт у Excel)**

Забезпечує можливість експортування даних у формат Excel:

– генерація Excel-файлів – створює файли у форматі, сумісному з Excel;

– коректне відображення даних – забезпечує правильне форматування інформації у таблицях;

**Auth (Автентифікація та авторизація)**

Відповідає за перевірку користувачів та керування їхнім доступом:

– перевірка ідентифікації – автентифікує користувача на основі введених даних;

– надання доступу – визначає рівень прав користувача у системі.

**Tasks (Завдання)**

Керує створенням та управлінням завданнями у системі:

– створення та редагування завдань – дозволяє додавати нові та змінювати існуючі завдання:

– відстеження стану завдань – визначає статус виконання завдань та їхні параметри.



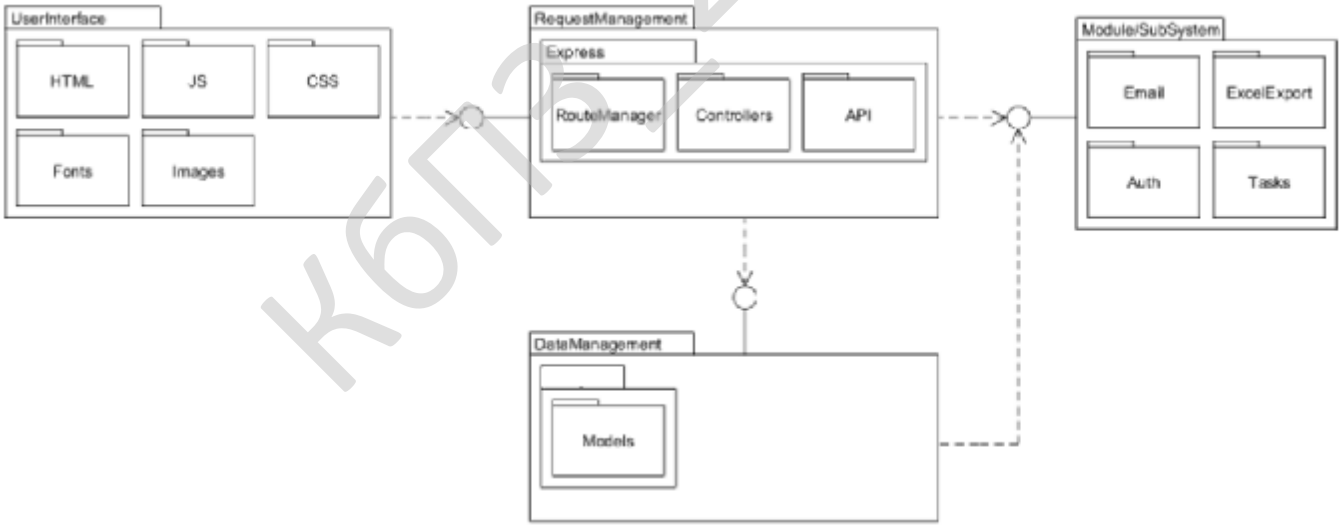



Рисунок 3.2 – Функціональна схема системи

### 3.4 Розробка діаграми процесів

Діаграми процесів, подібно до діаграм послідовності, належать до групи діаграм взаємодії. Вони візуалізують взаємодію об'єктів у вигляді графічного

представлення або сітки, демонструючи послідовність подій та акцентуючи увагу на зв’язках між елементами системи.

Процес моделювання виконання операції проходить за такими етапами:

- визначення параметрів, результатів та інших об’єктів, доступних для операції;

- якщо операція проста, її можна відобразити безпосередньо в коді, який або залишається у фоновому режимі моделі, або виноситься у вигляді анотації;

- для складних алгоритмічних операцій використовується діаграма діяльності, що моделює їх виконання;

- якщо операція вимагає детального проектування, варто розглянути можливість її спільної реалізації. Надалі можна розширити структурні та поведінкові аспекти взаємодії за допомогою діаграм класів та взаємодії.

Відношення між об’єктами - це зв’язок між екземплярами класів, що визначає особливості їхньої взаємодії та рівень видимості. Прикладом такого зв’язку є посилання. Повідомлення між об'єктами передаються у вигляді підписів над з'єднувальними лініями зі стрілками. Одна лінія може містити кілька повідомлень, а їх послідовність у потоці керування позначається порядковими номерами.

Діаграми співпраці, як і діаграми послідовності, використовуються для відображення взаємодії як між об'єктами предметної області, так і між компонентами програмного забезпечення.

Існують різні методи організації взаємодії між клієнтом і сервером. Одними з найпоширеніших технологій є WebSocket та AJAX. AJAX дозволяє змінювати вміст веб-сторінки без її повного перезавантаження, що підвищує динамічність веб-застосунків. Ця технологія базується на обміні даними в режимі реального часу та оновленні необхідних компонентів за потреби. Вперше AJAX був представлений у 1999 році, завдяки чому підтримується навіть у старих версіях браузерів.

WebSocket - це стандартна технологія HTML5, яка забезпечує

встановлення повнодуплексного TCP-з'єднання між веб-браузером (клієнтом) та сервером. Вона дозволяє усунути обмеження в обміні даними між браузерами та серверами, забезпечуючи швидку та ефективну взаємодію. Для використання цієї технології необхідно, щоб браузер підтримував WebSocket.

Для визначення оптимальних технічних рішень, вибору інструментів і технологій, а також розробки дизайну програми була створена діаграма процесів. Вона допомагає оцінити різні можливості використання та проаналізувати альтернативні варіанти реалізації системи.

Діаграму процесів зображено на рисунку 3.3.

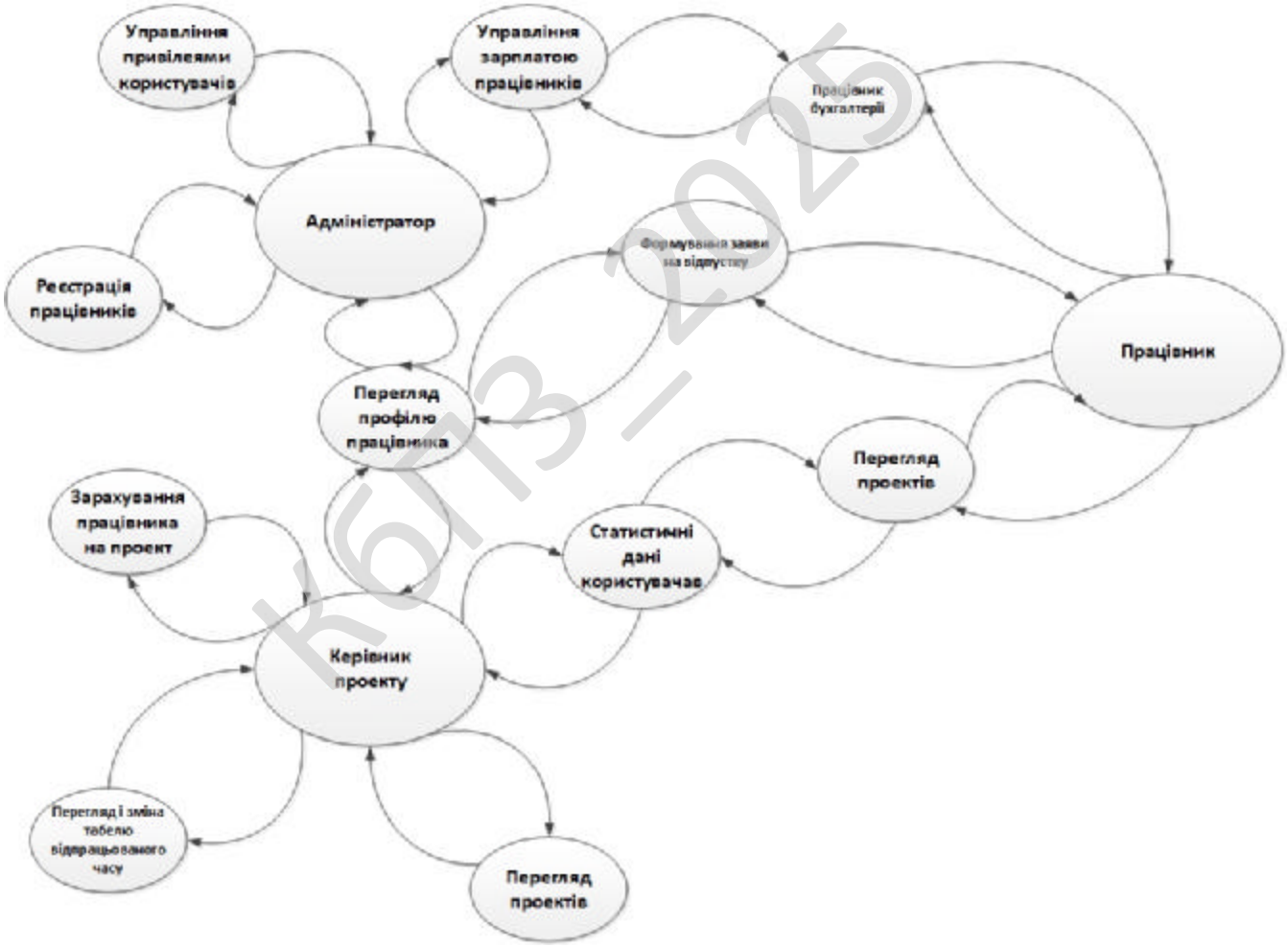



Рисунок 3.3 – Діаграма процесів

Таким чином, після аналізу опису системи, а також розгляду її структурної та функціональної схем, разом із діаграмою взаємодії процесів, можна перейти до

детального опису блок-схем основної програми та підпрограм, які застосовуються для реалізації системи.

# 4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

**Опис технологій для створення системи**

Аналіз сучасних веб-технологій показав, що платформа Node.js підходить для побудови системи. Node.js — це середовище JavaScript, створене на движку JavaScript Chrome V8. Node.js використовує неблокуючу модель введення-виведення на основі подій, що робить його легким і ефективним.

Модель, прийнята в Node.js (рисунок. 4.1), принципово відрізняється від звичайних платформ для побудови серверів, в яких масштабованість досягається завдяки багатопоточності. Завдяки подієво-орієнтованій архітектурі зменшується споживання пам'яті, підвищується пропускна здатність.



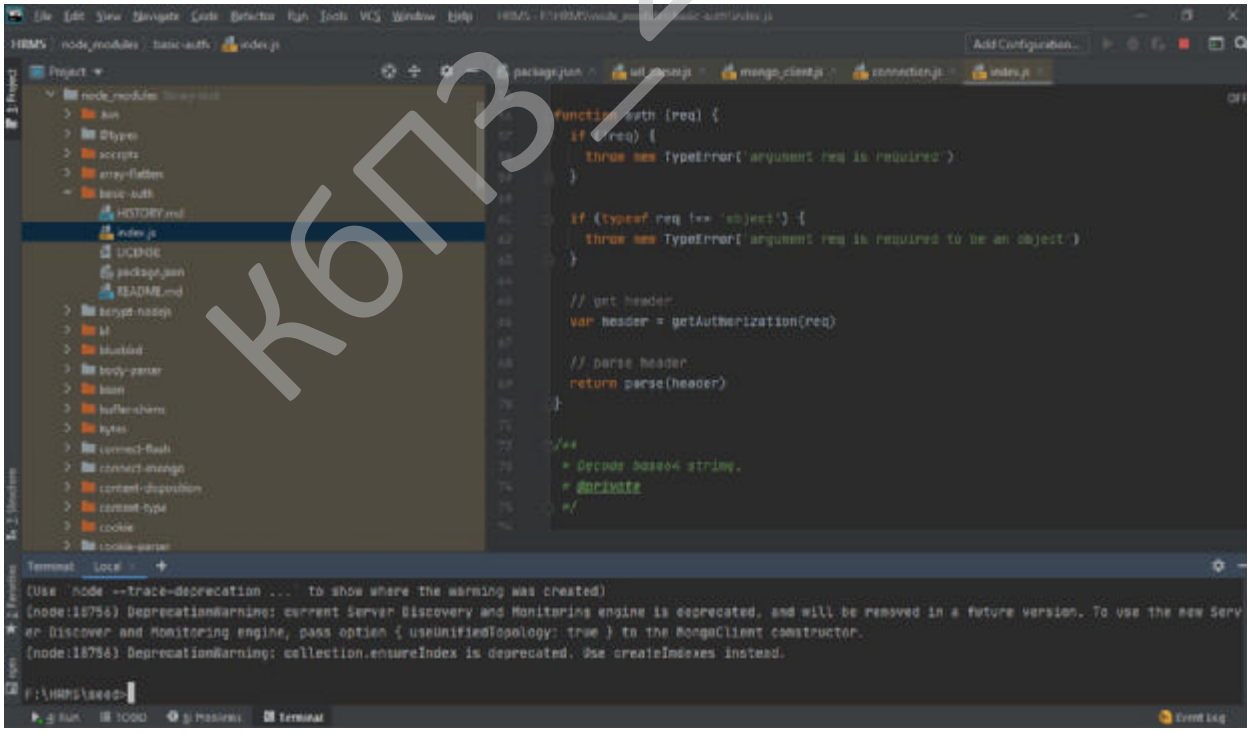


Рисунок 4.1 - Середовище розробки з Node.js

У Node.js при старті створюється єдиний програмний потік. Node.js виконується в цьому потоці. Це означає, що дві або більше частини додатка одночасно не можуть виконуватися. Можливість “одночасного” виконання коду надається за допомогою використання асинхронної моделі, побудованої на черзі подій (event-loop). На рисунку 4.2. зображена діаграма розгортання.

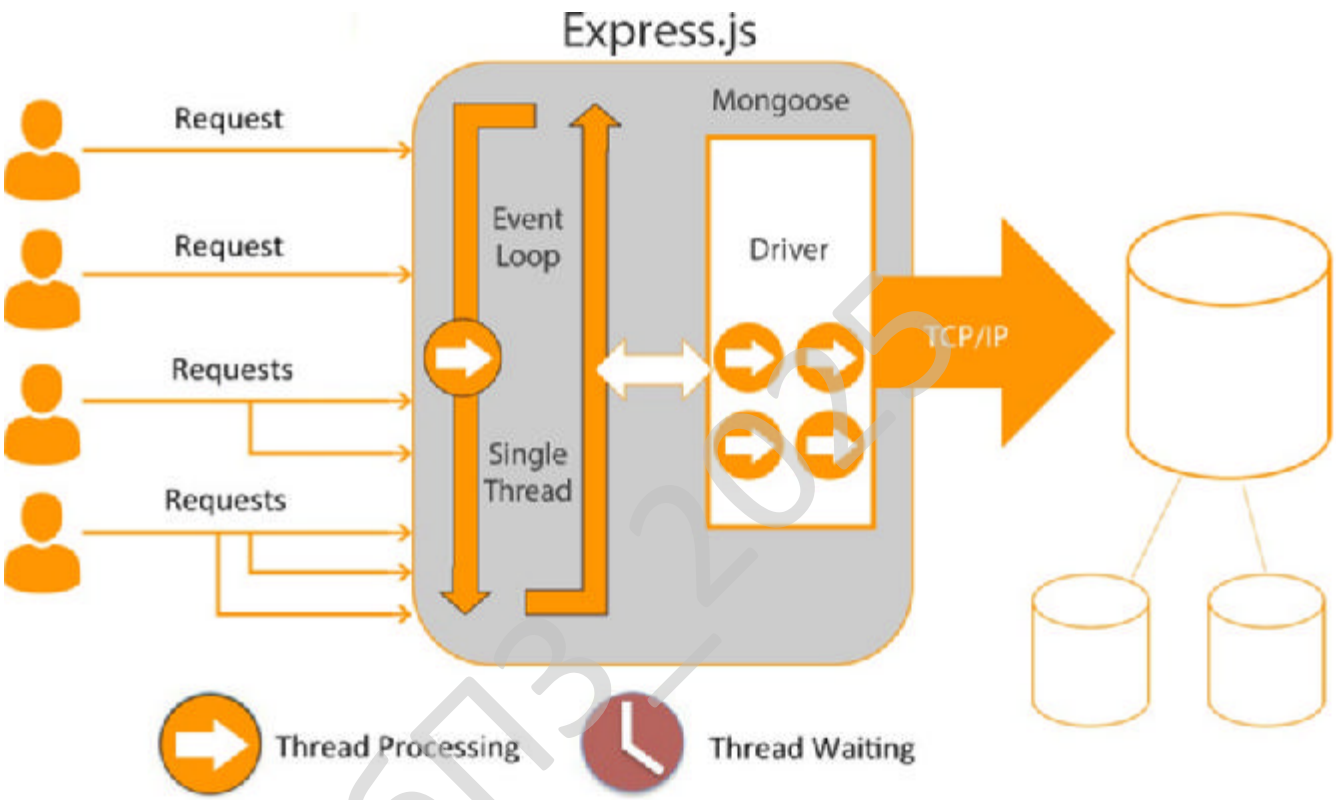



Рисунок 4.2 - Діаграма розгортання, включаючи Node.js Express, PostgreDB

База даних PostgreDB була обрана як СУБД з наступних причин:

- можливість обробки неструктурованих даних;
- відсутність або частковий зв'язок між колекціями;
- збереження даних не вимагає високої продуктивності.

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

Загальна архітектура веб-додатку. Система розробки базується на клієнт-серверній архітектурі. Клієнт робить запит серверу на виконання певних дій або надання інформації, сервер отримує запит, виконує його і надсилає відповідь.

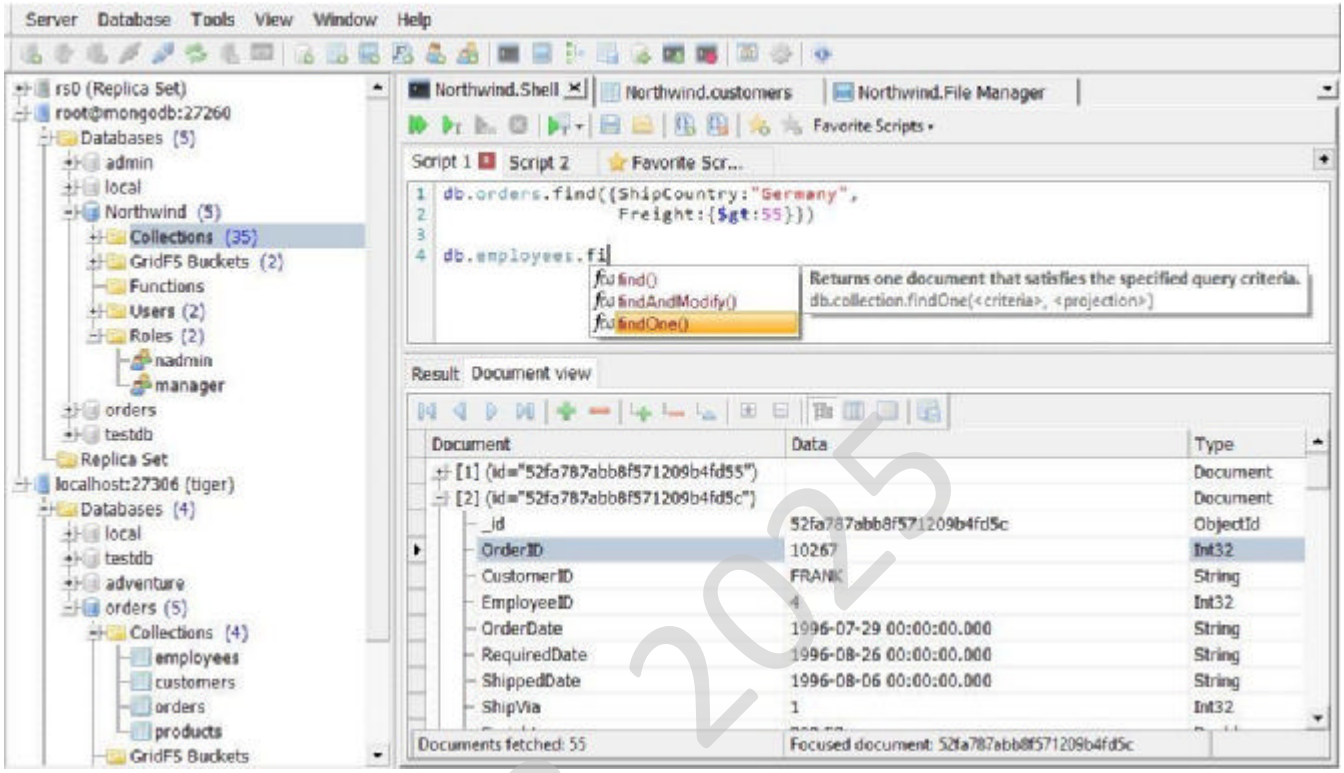



Рисунок 4.3 - PostgreDB Manager

Серверна частина системи управління кадровими процесами обробки запитів повинна мати три основні компоненти, які будуть відповідати за: обробку запиту, попередню обробку надісланих даних і обробку маршруту. У компоненті, відповідальному за обробку запиту, об’єкт даного запиту створюється та пересилається компоненту, відповідальному за обробку надісланих даних, після чого екземпляр, створений для обробки запиту, звільняється.

Функції, що відповідають проміжній обробці надісланих даних, у «Express» називаються «проміжним програмним забезпеченням». Вони об'єднуються в ланцюги і після виконання всього «проміжного програмного забезпечення» конкретного ланцюжка управління передається компоненту, що відповідає за обробку маршруту. Звільнення екземпляра обробки запиту та подальша передача

управління компоненту, що відповідає за обробку маршруту, здійснюється за допомогою функції зворотного виклику.

Побудова серверної частини за допомогою програмної платформи Node.js і обєктно-реляційної бази даних PostgreDB дозволяє створювати високошвидкісні додатки, орієнтовані на велику кількість одночасних підключень, вимагаючи невеликого обсягу оперативної пам’яті.

**Програмна реалізація системи**

Для розробки проекту було вирішено використовувати мову документів JavaScript з платформою Node.js. Це безкоштовна платформа, яка дозволяє писати програми на JavaScript і містить багато корисних модулів, тому немає необхідності писати код «з нуля».

Node.js складається з середовища виконання та бібліотек. При створенні програми були використані такі бібліотеки JavaScript:

- ekpress.js – фреймворк для практичного створення веб-додатків [12];

- socket.io - бібліотека JavaScript для реалізації з'єднання Web-Socket у веб-додатках і обміну даними в реальному часі. Він складається з двох частин: клієнта, який працює в браузері, і сервера для node.js. Обидва компоненти мають подібний API [14];

- WebStorm - незалежний редактор коду з підсвічуванням синтаксису, темами і гарячими клавішами, код, написаний на JavaScript, легко вбудовується в будь-яку веб-сторінку [10];

- monk.js – обгортка для бази даних MongoDB, яка дозволяє швидко та зручно взаємодіяти з колекціями баз даних [13].

WebSocket був обраний для реалізації зв’язку в реальному часі між клієнтом і веб-сервером. При такому з'єднанні обидві частини (клієнт і сервер) є рівноправними і мають можливість перевірити з'єднання за допомогою керуючих кадрів PING і PONG. Той, хто хоче перевірити з'єднання, надсилає кадр PING із довільним тілом. Його приймач повинен відповісти кадром PONG з тим самим тілом протягом прийнятного часу.

Обробники подій розроблені та реалізовані для обміну даними між стороною клієнта та сервером. Список обробників подій на стороні клієнта показано на рисунку 4.4.

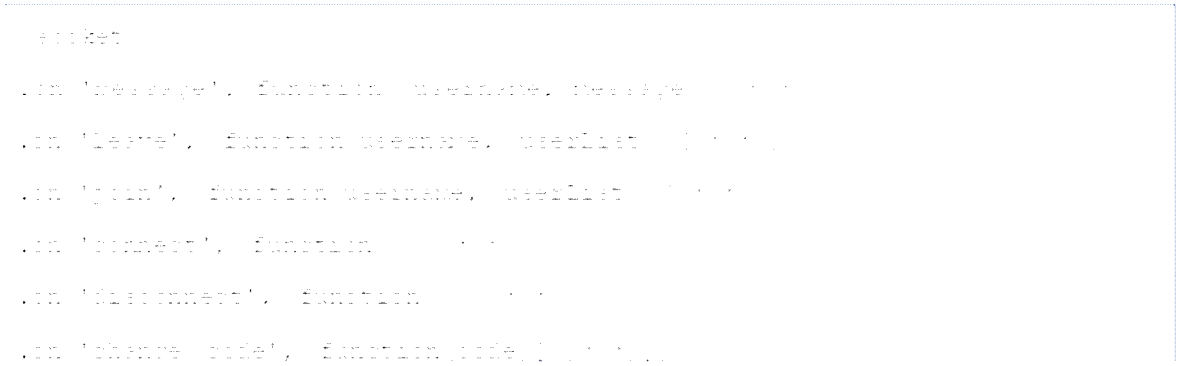


Рисунок 4.4 - Список обробників подій на стороні клієнта

Обробники подій на стороні клієнта можна більш детально описати наступним чином:

- повідомлення - подія при відправленні повідомлення в чаті веб-додатку;
- вихід - подія при закритті всіх вкладок браузера учасника, виході з його документа;
- приєднання - подія підключення учасника до системи;
- connect - подія під час активного з'єднання з сервером;
- disconnect - подія, коли з'єднання з сервером втрачається;
- change - подія, коли змінюється доступ до проекту;
- change lang - подія при зміні мови;
- зміна прав - подія, коли змінюються права певного учасника.

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

```
socket
.on('add user', function(user, cb) {/*…*/})
.on('message', function(text, cb) {/*…*/})
.on('disconnect', function(username, cb) {/*…*/})
.on('change code', function(code, cb) {/*…*/})
.on('change lang', function(lang) {/*…*/})
```

Рисунок 4.5 - Список обробників подій на стороні сервера

Слід розуміти, що хоча назви деяких подій на стороні клієнта та на стороні сервера однакові, реалізація подій принципово відрізняється. Так, наприклад, при обробці події повідомлення на стороні клієнта повідомлення, отримане від сервера, повинно відображатися в чаті, тоді як сервер обробляє цю подію, надсилаючи повідомлення всім учасникам чату.

На етапі проектування було вирішено використовувати документо-орієнтовану СУБД PostgreDB.

З перерахованих СУБД PostgreDB є найпопулярнішою для обраної платформи. Це розширювана, високопродуктивна СУБД із відкритим вихідним кодом без схем, яка зберігає всі дані у двійковому форматі JSON (BSON). Головною відмінністю PostgreDB є її гнучке використання у веб-додатках. Колекції з полями, показаними на рисунку 4.6, призначені для зберігання даних документів, створених користувачами.

У PostgreDB база даних складається з колекцій. Як правило, колекція відображає сутність предметної галузі. Колекція складається з «документів» (далі — записи). Запис — це об’єкт JSON з обов’язковим ідентифікатором ObjectId, який PostgreDB автоматично встановлює та контролює його унікальність, аж до унікальності в усьому світі [12]. ObjectId — це тип даних. У порівнянні з реляційною СУБД колекція — це таблиця, а запис — рядок у таблиці.

Тип даних, що зберігається в записі, є довільним, наприклад: рядок; Масив (структура даних, доступ до якої здійснюється за цілим індексом); логічний; дата;

Ціле число; Нуль. Об’єкт (структура даних, що дозволяє отримати доступ до елементів за ключем).

PostgreDB підтримує реляційну цілісність. Це завдання вирішується на рівні коду сервера додатків, який реалізується за допомогою засобів ODM.


Рисунок 4.6 - Модель документів бази PostgreDB

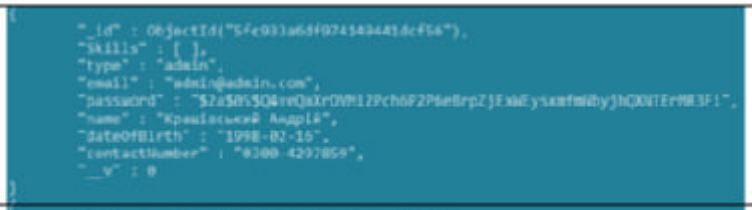


Колекція містить ідентифікатор користувача, який створив деталь, ідентифікатор проектів, деталі поточного отримання реквізитів платежу та ідентифікатор запису в колекції. Було вирішено, що нова вкладка документа сприймається як картка того ж користувача і що з'єднання з клієнтом припиняється лише тоді, коли всі вкладки, відкриті цим документом, закриті.

Складність полягає в тому, що за стандартом WebSocket нова вкладка браузера є повноцінним з'єднанням клієнт-сервер. Таким чином, таблиця підключених користувачів створюється для конкретного проекту, щоб відстежувати

всі сокети, які підключив користувач. Модель збережених даних наведена в таблиці 4.1.

Таблиця 4.1 - Приклад списку формування списку працівників

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

Для того, щоб мати представлення зв’язку між сутностями, була розроблена схема бази даних, на якій показані псевдозв’язки, рисунок. 4.7, нотація Гордона Евересту [21].

Додамо до схеми, що імена колекцій і псевдозв’язків можуть представляти дещо інше значення, ніж таблиці в реляційних базах даних, оскільки там є вкладені об’єкти. Прикладом є колекція під назвою product_type.

Типи псевдозв'язків документа з іншими сутностями з урахуванням неструктурованості даних:

- HasOne: має один тип документа.

- HasOne: має одну мову.

- ManiToMani: належить до одного або кількох продуктів (BelongsTo), оскільки є загальним. З іншого боку, один продукт має багато документів різними мовами (HasMani).

- HasMani: є багато змін.

- BelongsTo: Належить до того самого завантаження, електронної пошти або замовлення.

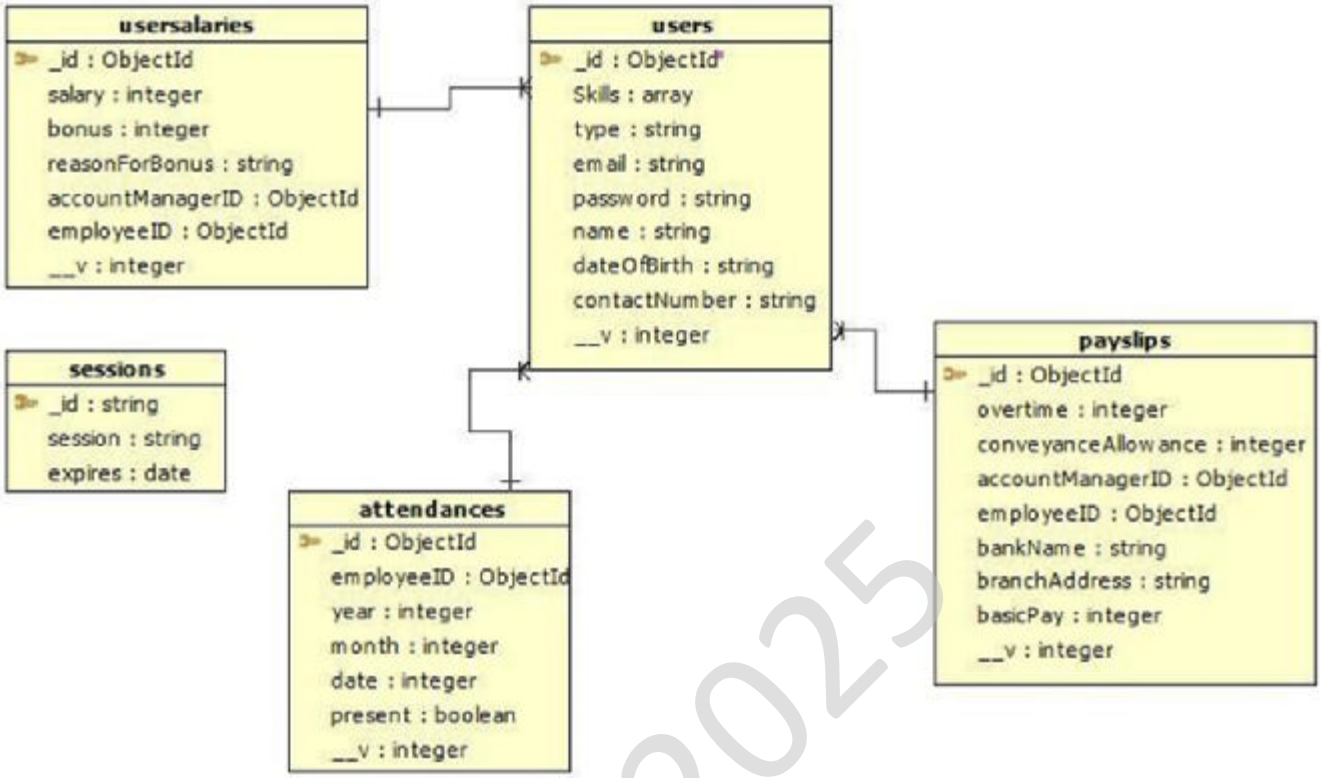



Рисунок 4.7 - Схема бази даних з позначенням псевдо-зв’язків

Наступні поняття використовуються для опису функцій дизайну схеми в PostgreDB:

- Вбудовування - повне збереження об'єкта(ів) однієї сутності в іншій.

- Посилання - унікальний ідентифікатор ObjectId зовнішньої сутності. Використання інструменту PostgreJS дозволяє автоматично генерувати ObjectId під час вставки вкладеного об’єкта. Наприклад, коли ви додаєте продукт до типу товару. Під час проектування були визначені деякі конструктивні особливості схеми бази даних у PostgreDB, а саме:

- з композитними псевдоключами втрачається гнучкість роботи з даними;

- зберігання лише ідентифікатора об’єкта зовнішньої сутності вимагатиме додаткового запиту для отримання всіх даних зовнішньої сутності.

Вкладеність об’єкта сутності має сенс, коли вкладений об’єкт з’являється разом із батьківським.

Вкладеність об’єкта сутності відображає зв’язок «один до багатьох» у реляційній схемі.

Посилання забезпечують більшу гнучкість із частими змінами вкладених об'єктів.

Рядок посилання на об’єкт сутності відображає зв’язок «багато до багатьох» у схемі відношення. Атрибути основної сутності Users наведені в таблиці 4.2, а на рисунку 4.8 показано структуру проекту інформаційної системи зберігання даних.

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 |
| datcOfBirth | string | Дата народження |
| contactNumber | string | Контактний номер телефону |
| __v | datc | Дата нагадування актуальності |
| department | string | Підрозділ |
| dcsignation | string | Посада |



При запиті користувача пошук за атрибутами: ім'я, контактний телефон, відділ, ярлик. Результат має повернути інформацію про користувача з відповідними даними результату.

## 4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.8.

Як тільки користувач потрапив на головну сторінку сайту він повинен пройти авторизацію або реєстрацію у випадку відсутності аккаунта. На рисунку 4.8 зображено блок-схему алгоритму роботи сторінки реєстрації користувача.

На сторінці реєстрації користувач має можливість пройти реєстрацію нового аккаунта для доступу до закритої частини сайту. Для успішної реєстрації користувач повинен надати свою фотографію, ім’я, поштову адресу та бажаний пароль. Після введення всіх даних система перевіряє заповненість усіх полів та присутність завантаженого зображення і, або виводить помилку, якщо щось було пропущено, або створює два записи у таблицях бази даних. Перший запис містить введену користувачем інформацію про його аккаунт, а другий містить інформацію потрібну для активації даного аккаунта, що зберігається у таблиці з інформацією про аккаунти, що очікують активації. Далі система надсилає листа на введену користувачем поштову адресу з посиланням, яке містить код активації аккаунта, і повідомляє користувача, що на вказану ним адресу було відправлено листа з кодом активації.

Після того як користувач перейшов по отриманому посиланню система перевіряє інформацію у цьому посиланні, а саме код активації аккаунта і якщо даний код буде збігатися з кодом у рядку в таблиці з неактивованими аккаунтами – система видалить інформацію про неактивований аккаунт користувача та встановить, що даний аккаунт користувача активовано.

Потім система перенаправляє користувача на сторінку авторизації на якій він має можливість увійти на свій аккаунт та розпочати роботу.

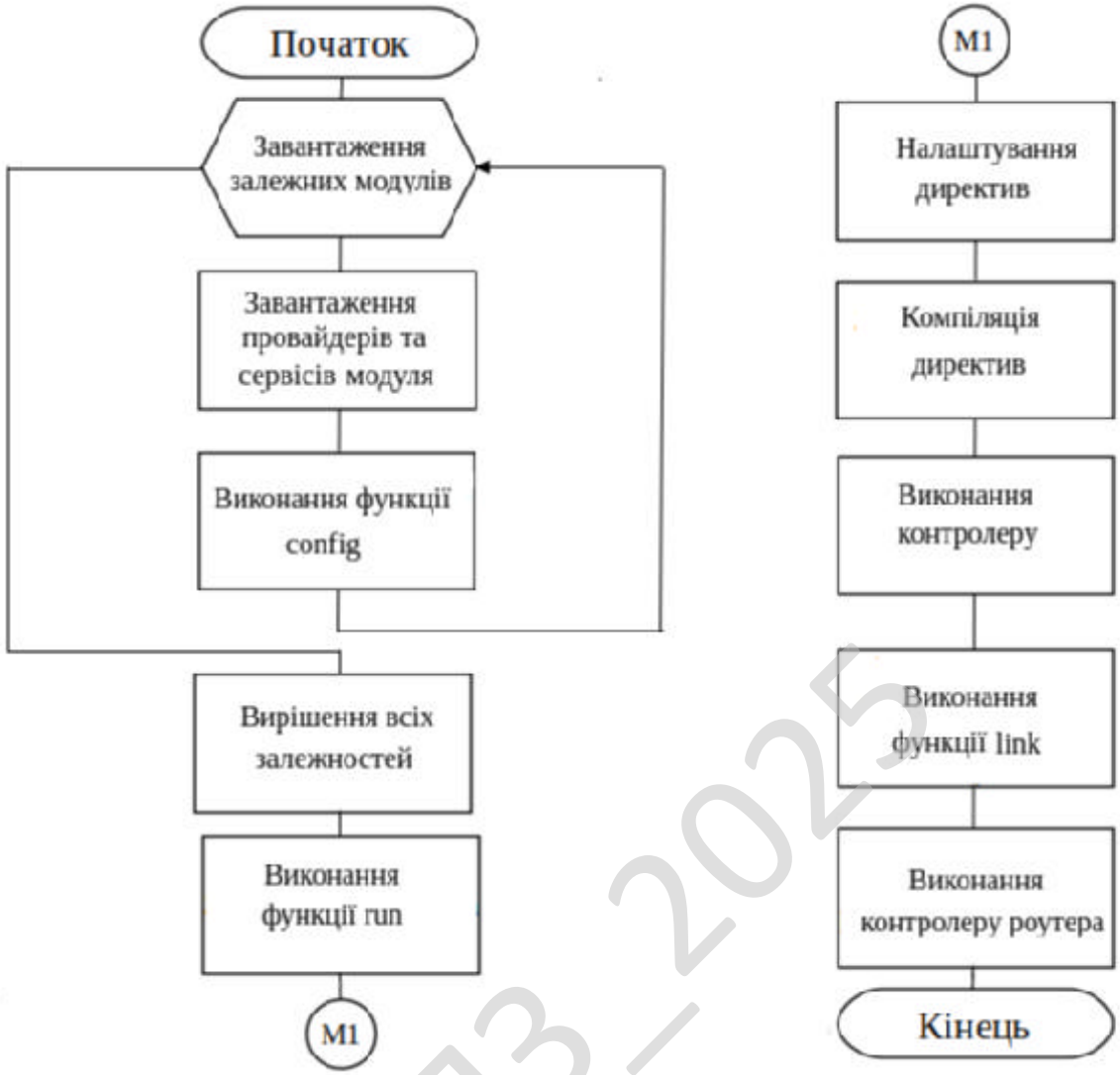



Рисунок 4.8 – Блок-схема алгоритму роботи ІС HR

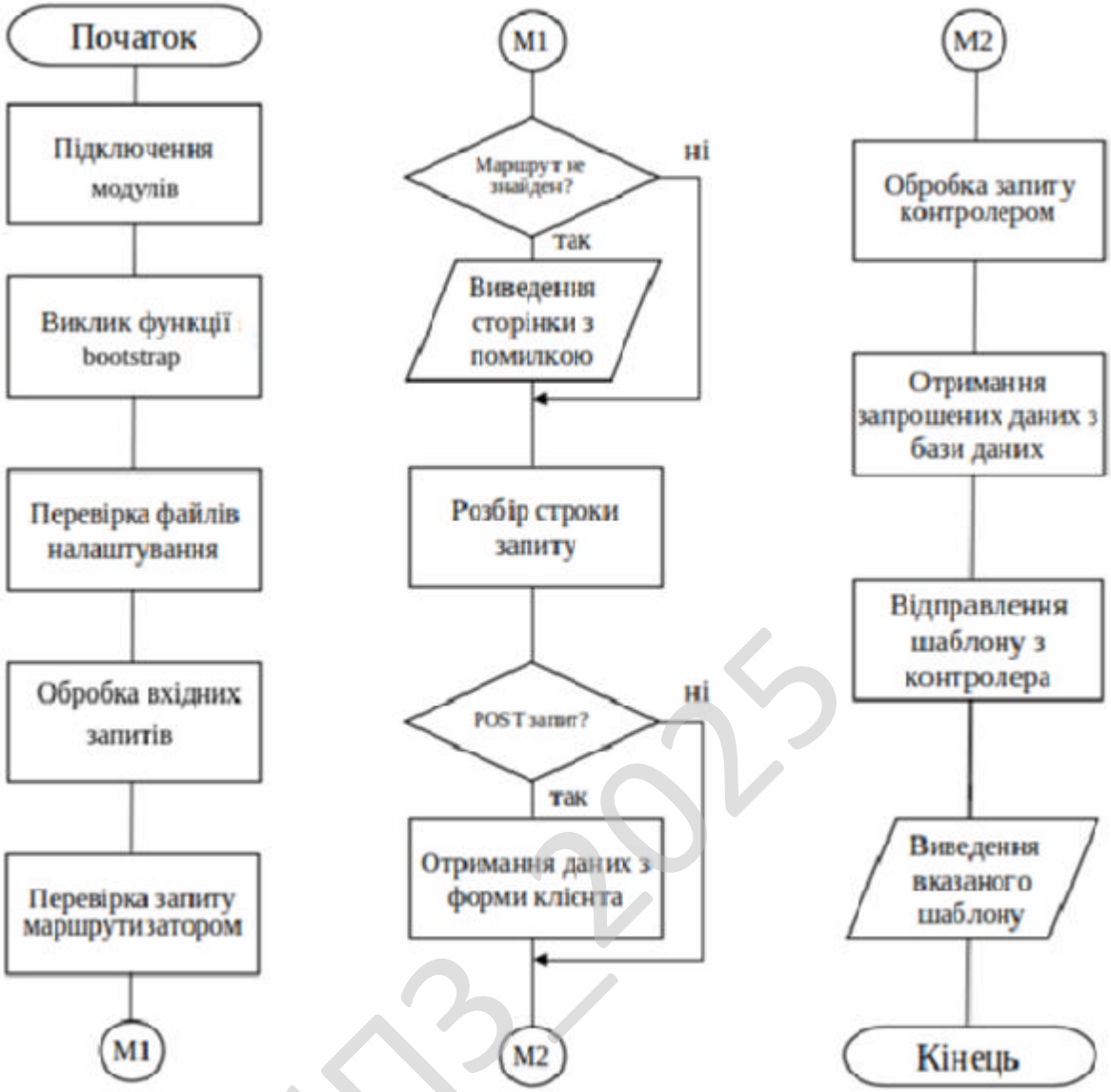



Рисунок 4.9 –Блок-схема алгоритму роботи основної програми



## 4.2 Захист розробленого програмного забезпечення

**Класифікація загроз безпеки інформації**

Загроза інформаційної безпеки — це подія або дія, які можуть спричинити зміни в роботі системи, пов'язані з порушенням захищеності інформації, що в ній обробляється.

Вразливість інформації - це можливість такого стану, коли створюються умови для реалізації загроз інформаційній безпеці.

Атака на інформаційну систему – це вжиті дії зловмисником, який має

знайти та використати вразливі місця.

Іншими словами, атака на КС – це реалізація загрози безпеці інформації в ній.

Проблеми, які виникають із безпекою передачі інформації при роботі в комп'ютерні мережі можна розділити на три основні типи [17]:

- захоплення інформації - цілісність інформації зберігається, але вона порушується конфіденційність;

- редагувати інформацію - вихідне повідомлення змінено або повністю замінюється іншим і надсилається адресату;

- заміна авторства інформації. Ця проблема може мати серйозні наслідки.

Наприклад, хтось може надіслати лист від імені когось іншого (цей вид шахрайства зазвичай називають спуфінгом) або веб-сервер може видаватися електронним магазином, приймайте замовлення, номери кредитних карток, але не надсилати ніяких товарів.

Специфіка комп’ютерних мереж з точки зору їх уразливості пов’язана з особливо за наявності інтенсивної інформаційної взаємодії між територіально розосереджених і неоднорідних (різних типів) елементів.

Практично всі основні структурні та функціональні елементи КС є вразливі: робочі станції, сервери (хост-машини), з'єднувальні мости(комутація), канали зв'язку тощо.

Існує велика кількість різноманітних загроз безпеці інформації. У літературі є багато різних класифікацій, де в’язкість критеріїв розподілу використовує типи виникаючих небезпек, ступінь злого наміру, джерела загроз тощо. Одна із найпростіших класифікації показані на рисунку 4.10.

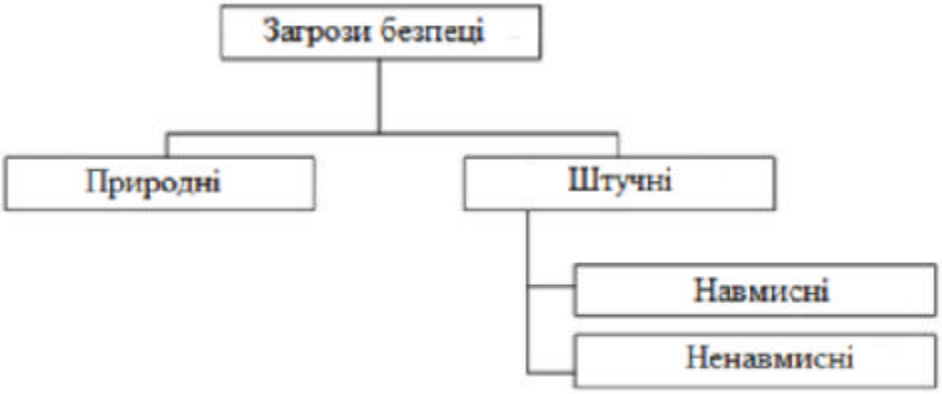



Рисунок 4.10 - Загальна класифікація загроз безпеки

Природні загрози – це загрози, викликані впливом на інформаційну систему та її елементи об'єктивних фізичних процесів або природних елементів, які незалежні від людини.

Штучні загрози – це загрози інформаційної системи, викликані діяльністю людей. Серед них, виходячи з мотивації провадження, можна виділити:

- ненавмисні (ненавмисні, випадкові) загрози, викликані помилками в проектування інформаційної системи та її елементів, помилки в програмному забезпечення, кадрові помилки тощо;
- навмисна (навмисна) загроза, пов’язана з егоїстичними прагненнями людей (виконавці).

Загрози інформаційної системи можуть бути зовнішні або внутрішні (компоненти інформаційної системи - її обладнання, програми, персонал).

Обов'язковим є аналіз негативних наслідків реалізації загрози виявлення можливих джерел загроз, вразливостей, що сприяють їх прояву та методів реалізації. А потім ланцюжок переростає в схему, показану на рисунку. 4.11.

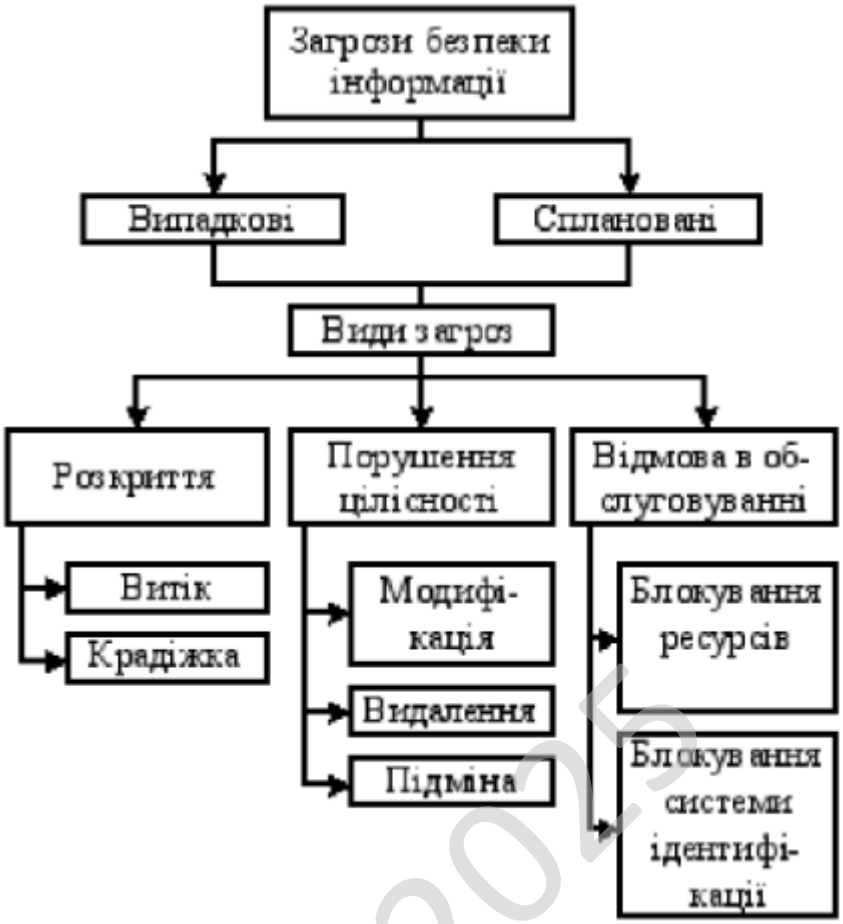



Рисунок 4.11 - Види загроз безпеки інформації в комп'ютерних мережах

**Завдання безпеки:**

- захист інформації в каналах зв'язку та криптографічних базах даних методи;

- автентифікація об'єктів даних і користувачів;

- (перевірка осіб, які здійснюють підключення);

- виявлення порушення цілісності об'єкта даних;

-забезпечення охорони технічних засобів і приміщень, в яких воно виконується;

- обробка конфіденційної інформації, від витоків через вторинні канали і можливо, вбудовані в них електронні пристрої запису інформації;

- забезпечення захисту комп'ютерного програмного забезпечення та засобів методики впровадження програмних вірусів і закладок;

- захист від несанкціонованого втручання через канал зв'язку від сторонніх

осіб до засобів шифрування, але їхня мета – порушити секретність інформування та дезорганізація абонентських пунктів;

- організаційно-технічні заходи, спрямовані на забезпечення безпеки конфіденційні дані.

Згідно розробленій ІС схемі було проаналізовано схему передачі даних яка представлена на рисунку 4.12.

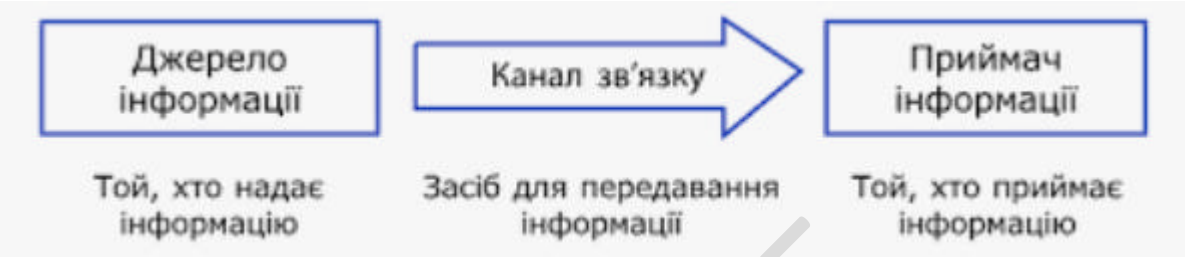



Рисунок 4.12 - Схема передачі даних в системі ІС та виявлення потенційних загроз

**База даних сертифікатів** є частиною фізичного сервера, який зберігає все в цифровому вигляді - підписи, до яких логічний сервер має повний і частковий доступ користувача зі свого мобільного пристрою.

У межах локальної мережі (серверної мережі) дані вважаються умовно захищеними.

Небезпеку становить незахищений канал користувача. Один з класичних сценаріїв – man-in-the-middle, тобто можливість інших осіб підключитись в канал між сервером та користувачем та видавати себе за когось з цих ключових осіб, беручи на себе роль невидимого посередника. Для того, щоб не допустити витік інформації, слід використовувати схеми з шифрування даних.

**Схеми шифрування**

**Симетричні криптосистеми**

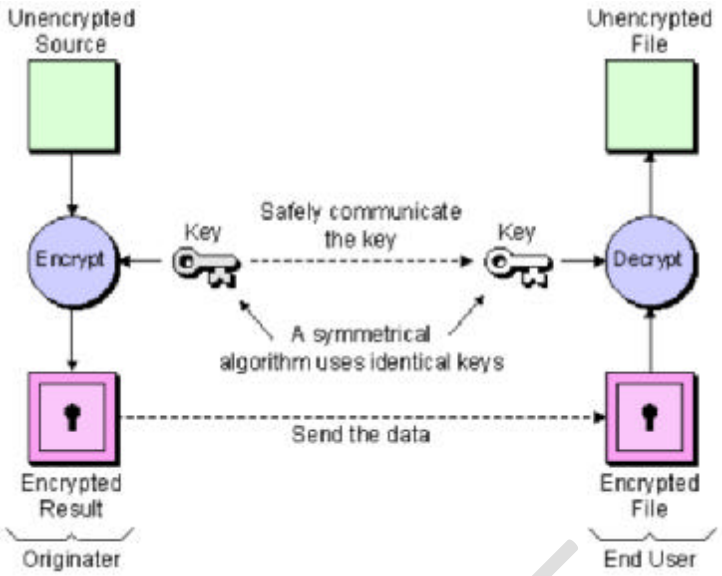



Рисунок 4.13 - Симетрична криптосистема

Симетричне шифрування передбачає використання одного і того ж ключа і для зашифрування, і для розшифрування. До симетричних алгоритмах застосовуються дві основні вимоги: повна втрата всіх статистичних закономірностей в об'єкті шифрування і відсутність лінійності. Прийнято розділяти симетричні системи на блокові і потокові. У блокових системах відбувається розбиття вихідних даних на блоки з подальшим перетворенням за допомогою ключа.

**Система з відкритим ключем**

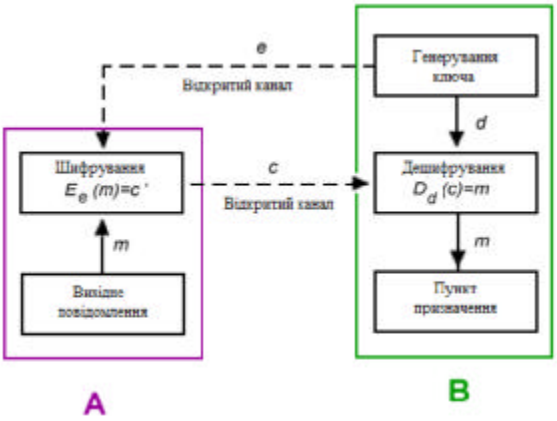



Рисунок 4.14 - Система з відкритим ключем

**Вразливість схем з відкритим ключем**

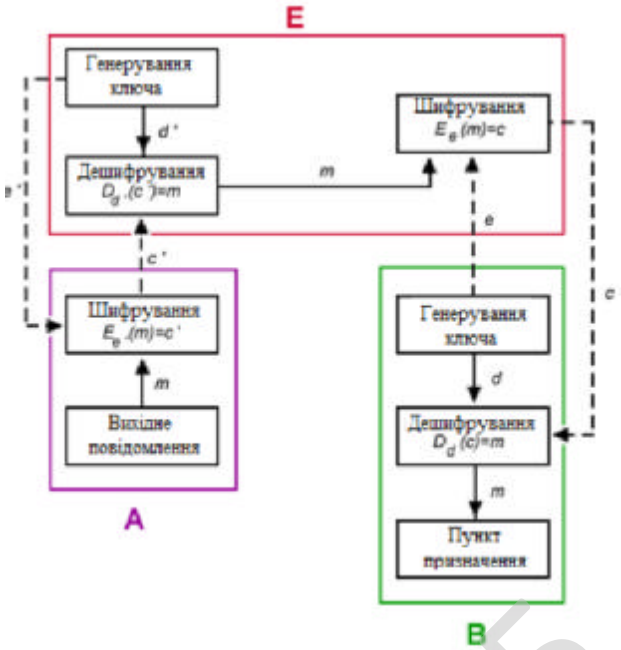



Рисунок 4.15 - Вразливість схем з відкритим ключем

**Інфраструктура відкритих ключів**

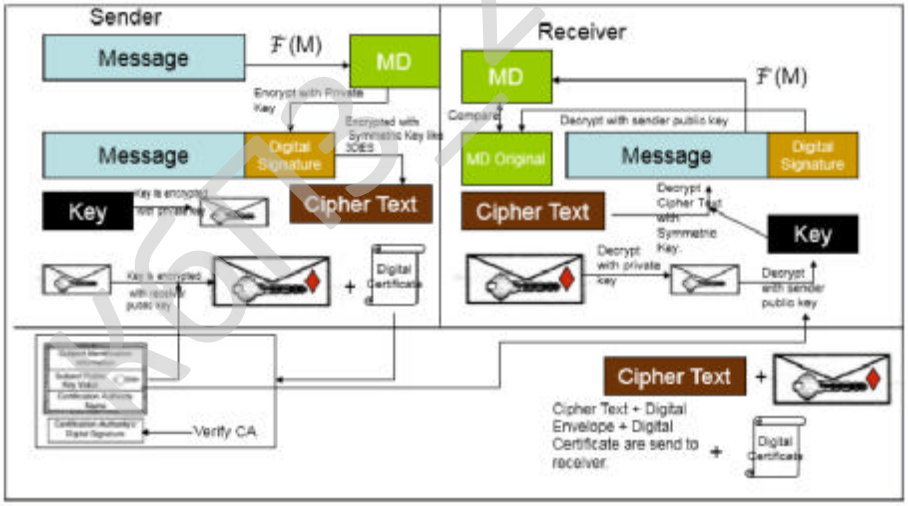



Рисунок 4.16 - PKI

**Алгоритми шифрування**

**AES**

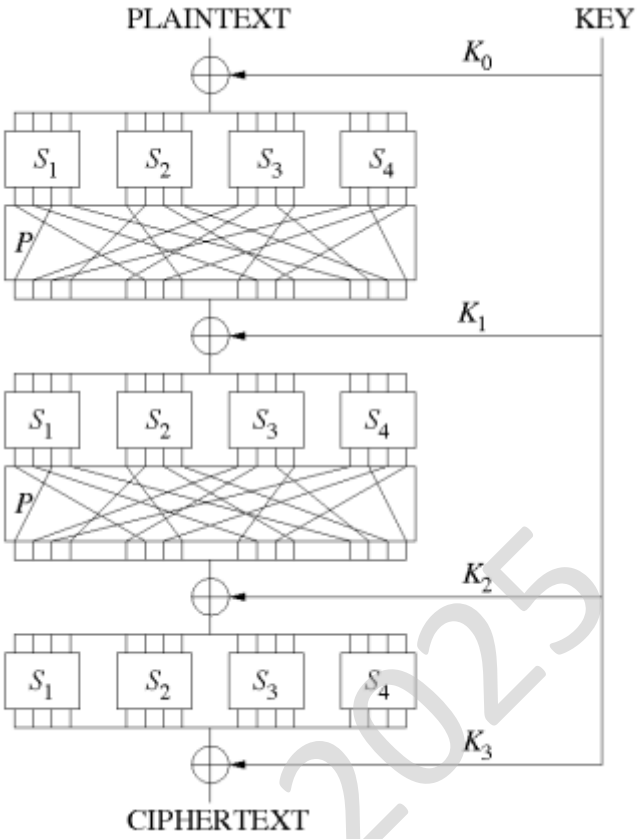



Рисунок 4.17 - SP

Алгоритм:

- KeyExpansion - генерація раундових ключів;

- InitialRound.

Використовуючи S-box (будується згідно властивостей поля Галуа GF(28 ), відбувається незалежна заміна байт масиву State.

Береться зворотне число b в полі Галуа.

$$\begin{Vmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 & \mathbf{0} & 0 & 1 & \mathbf{1} & 1 & 1 \\ 1 & 1 & \mathbf{0} & 0 & 0 & \mathbf{1} & 1 & 1 \\ 1 & 1 & \mathbf{1} & 0 & 0 & \mathbf{0} & 1 & 1 \\ 1 & 1 & \mathbf{1} & 1 & 0 & \mathbf{0} & 0 & 1 \\ 1 & 1 & \mathbf{1} & 1 & 1 & \mathbf{0} & 0 & 0 \\ 0 & 1 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 1 & 0 \\ 0 & 0 & \mathbf{0} & 1 & 1 & \mathbf{1} & 1 & 1 \end{Vmatrix} * \begin{Vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix}$$

Рисунок 4.18 - Операції перетворення

Таким чином, надається захист від атак, основаних на простих алгебраїчних властивостях.

**Вибір системи шифрування**

Тож давайте розберемо описані вище системи та алгоритми. Із систем є найоптимальнішою система, яка використовує PKI, оскільки забезпечує надійний канал для передачі ключів сеансу між сервером і клієнтом. Як сертифікат матиме певний ключ, який знають лише користувач і база даних сертифікати (у нашому випадку частина сервера) і база даних сертифікатів зберігає ключ у парі з відкритим ключем користувача. Потім система передачі будуть мати такий вигляд (для наочності розділимо їх на 2 етапи: авторизація та пряме з'єднання між 2 вузлами):

- авторизація;
- клієнт підписує кодову фразу «SH0» (що означає запит на авторизацію) його секретний ключ;
- сервер отримує повідомлення та надсилає його в базу даних сертифікатів;
- якщо база даних сертифікатів знайде вказаний підпис, це відбудеться авторизації, а саме: сервер отримує відкритий ключ клієнта з бази даних і надсилає йому сеансовий ключ AES, зашифрований цим ключем, включаючи фразу «SH1»; інакше генерується виняток "SH4";
- сеанс "спілкування";
- після отримання ключа сесії клієнт і сервер можуть передати дані один

одному за допомогою кодової фрази "SH2";

- в кінці клієнт надсилає «SH3» із закодованою фразою «Вихід», який завершує сеанс і знищує ключ сеансу відповідно до схеми.

Раз на рік система сертифікації повинна оновлювати ключі щоб звести до мінімуму ймовірність злому. Майте на увазі, що існують також прямі мікроконтролери, які відповідають за зчитування та контроль приладів. Ми були з самого початку визначив сервер локальної мережі як "зону довіри", але цілком імовірно, що правопорушник може перебувати поруч із сервером, але відносно недовго, тобто такий час, що неможливо знайти ключ до такого тривіального алгоритму шифрування як Base64. Цей алгоритм цілком підходить для цієї мети завдяки тому, що він не перевантажує мікроконтролери відносно низькою потужністю, це забезпечує певний захист інформації, не створюючи надмірності затримки.

**Висновки**

Під час роботи над цією частиною була перевірена робота схем шифрування:

- симетричні системи;
- системи відкритих ключів;
- інфраструктура відкритих ключів;

Була ідентифікована( змішана схема шифрування з використанням Base64, AES, RSA, протестували його та описали програмну реалізацію, яка задовольняє їх.

# 5 МЕТОДИКА 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.



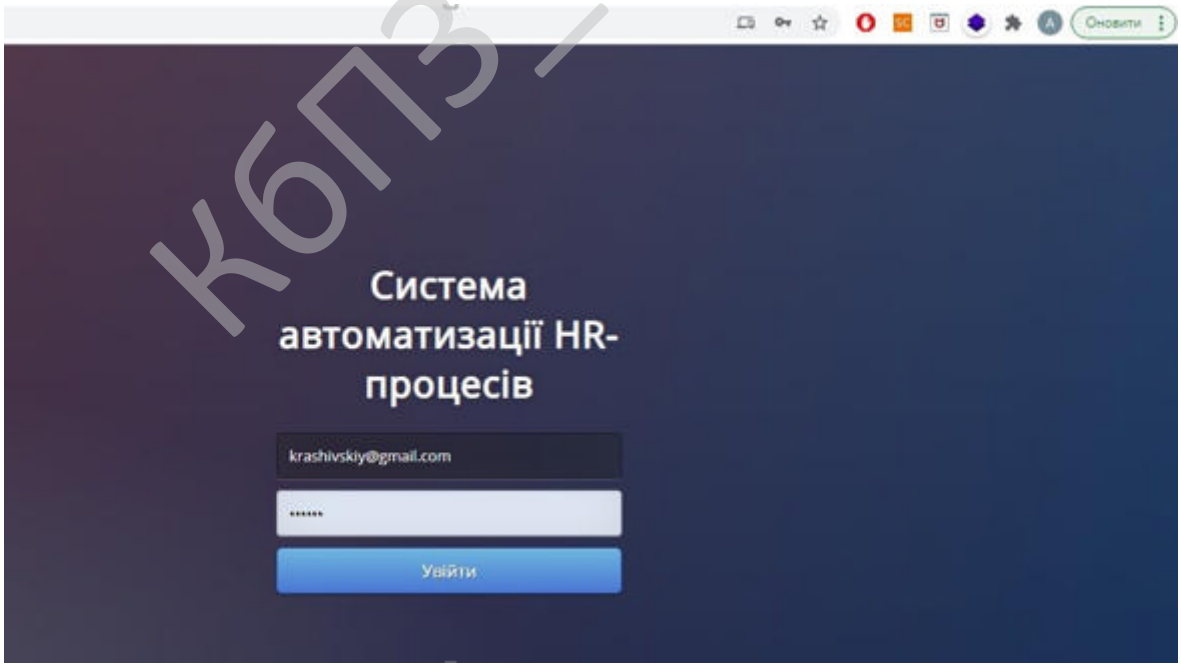



Рисунок 5.1 – Головна сторінка для входу в систему

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

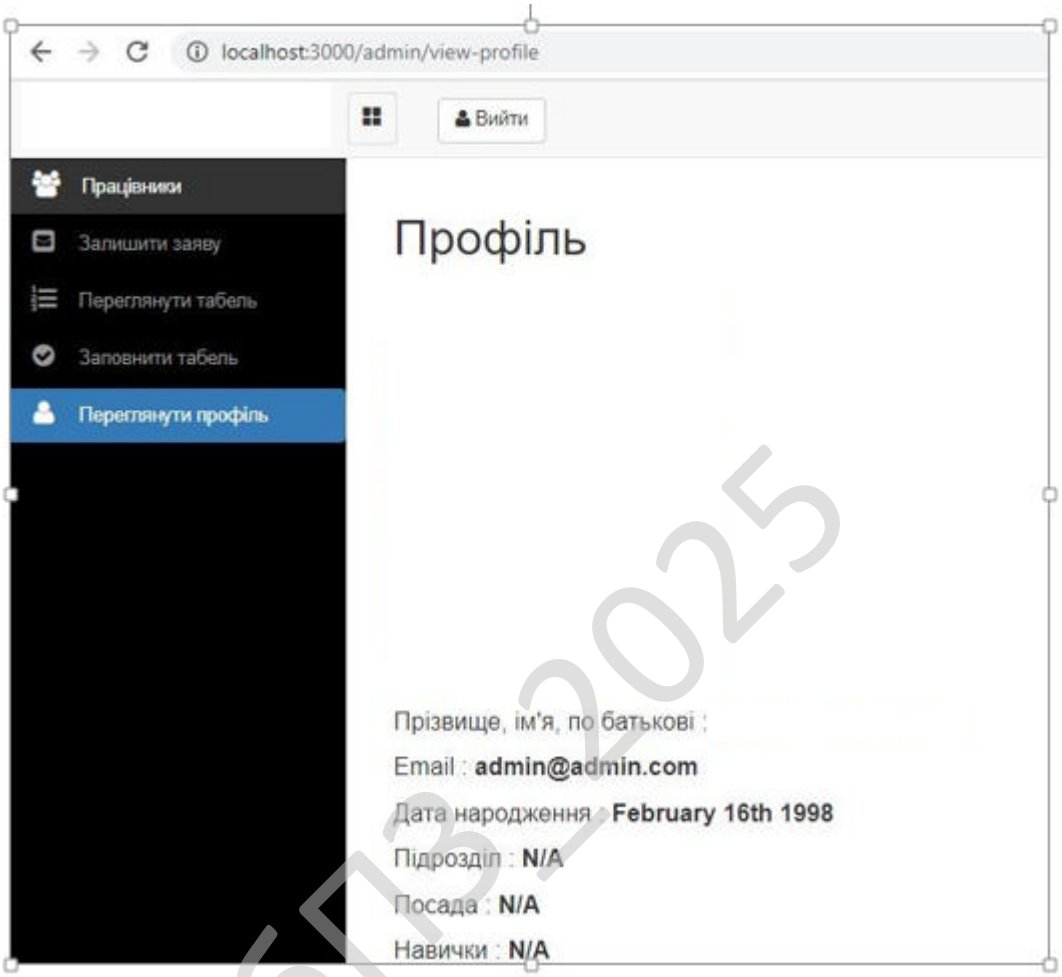



Рисунок 5.2 – Реалізація функціональності для адміністратора



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 PostgreDB.

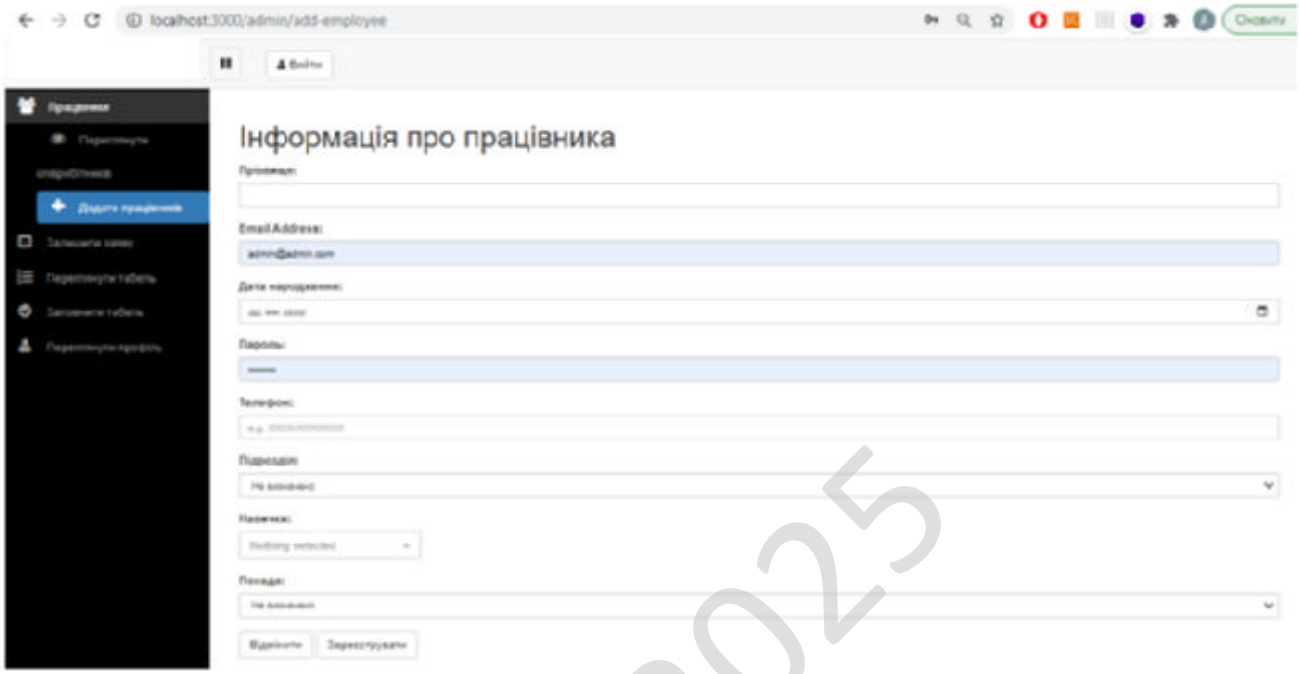



Рисунок 5.3 – Форма реєстрації нового працівника

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.

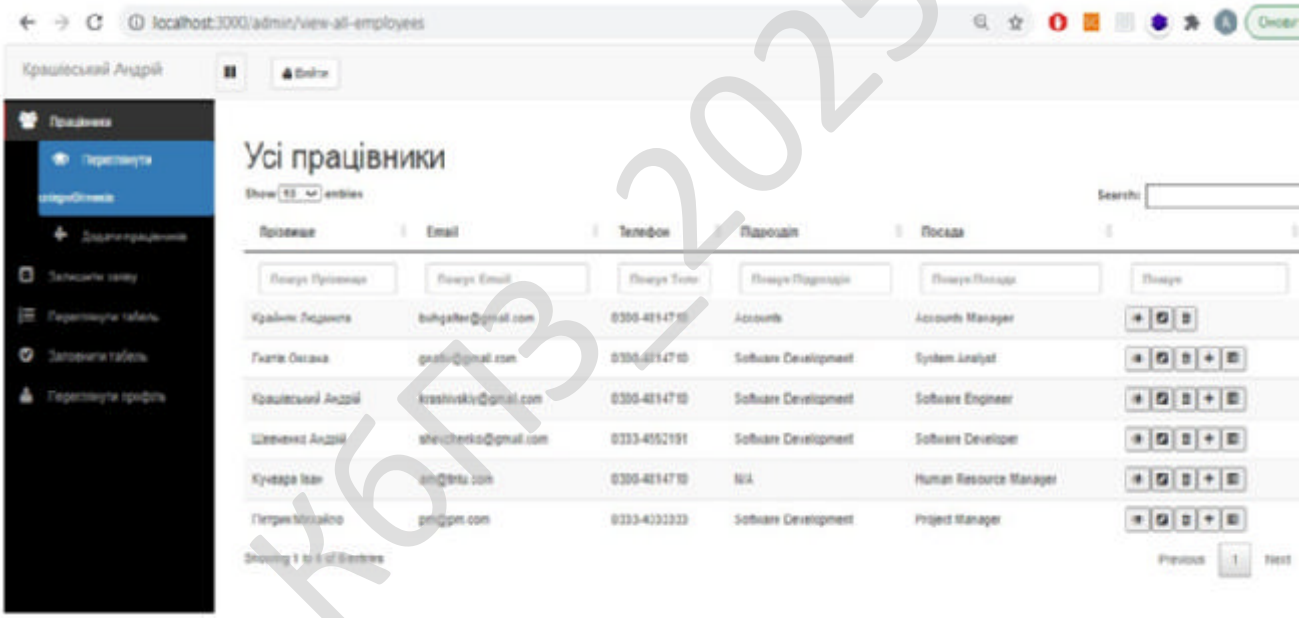



Рисунок 5.5 – Сторінка відображення інформації про працівників, які зареєстровані в системі

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

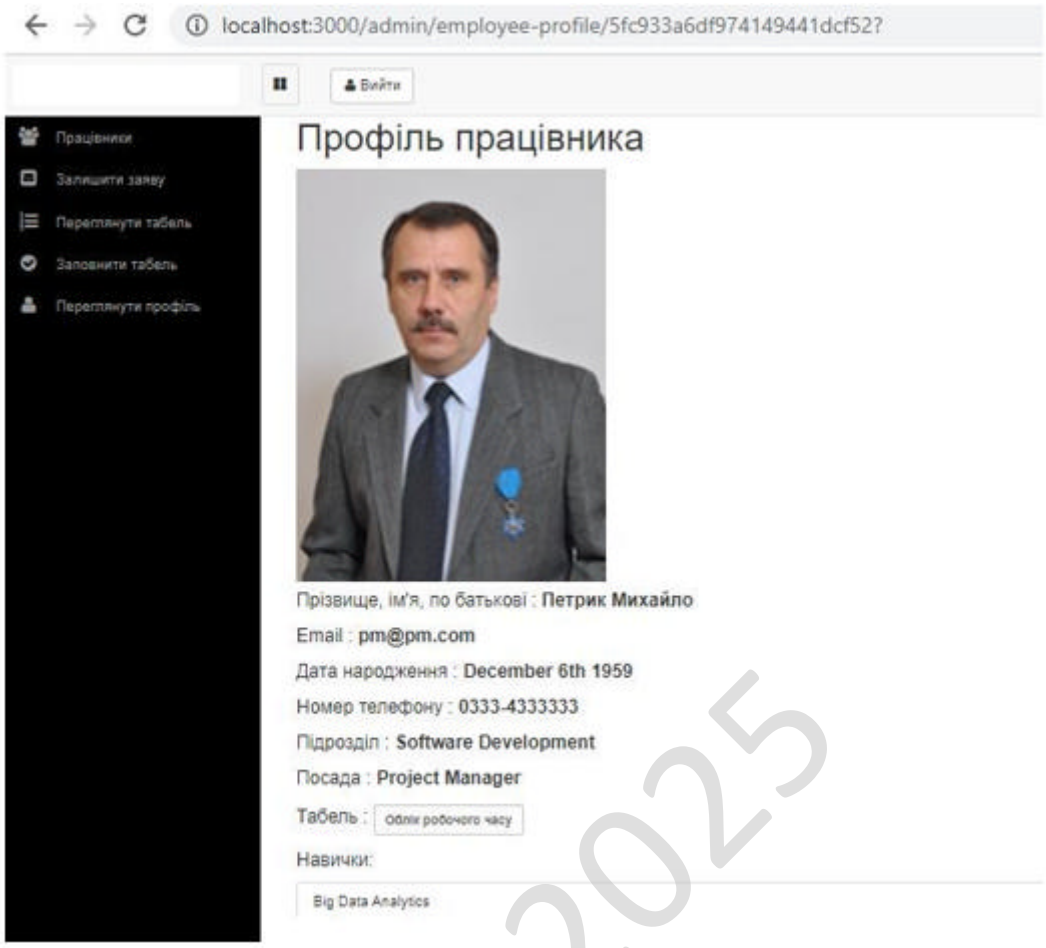



Рисунок 5.6 – Сторінка відображення інформації про працівника

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.

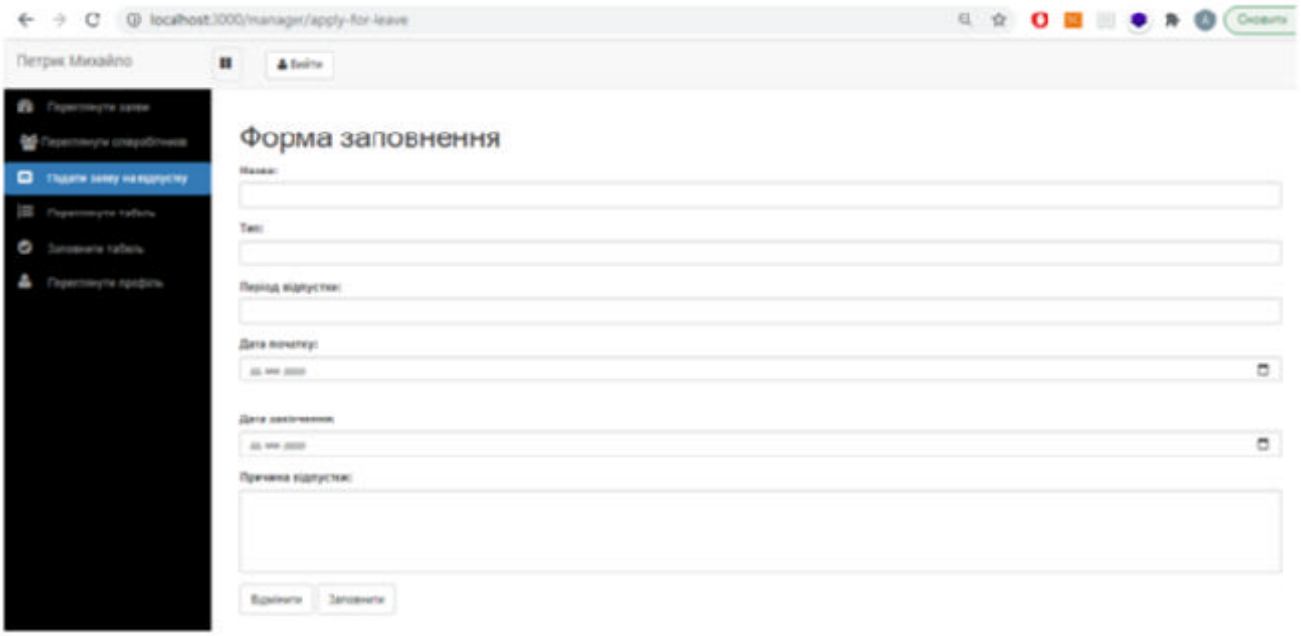



Рисунок 5.7 – Форма подачі заяви на відпустку

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

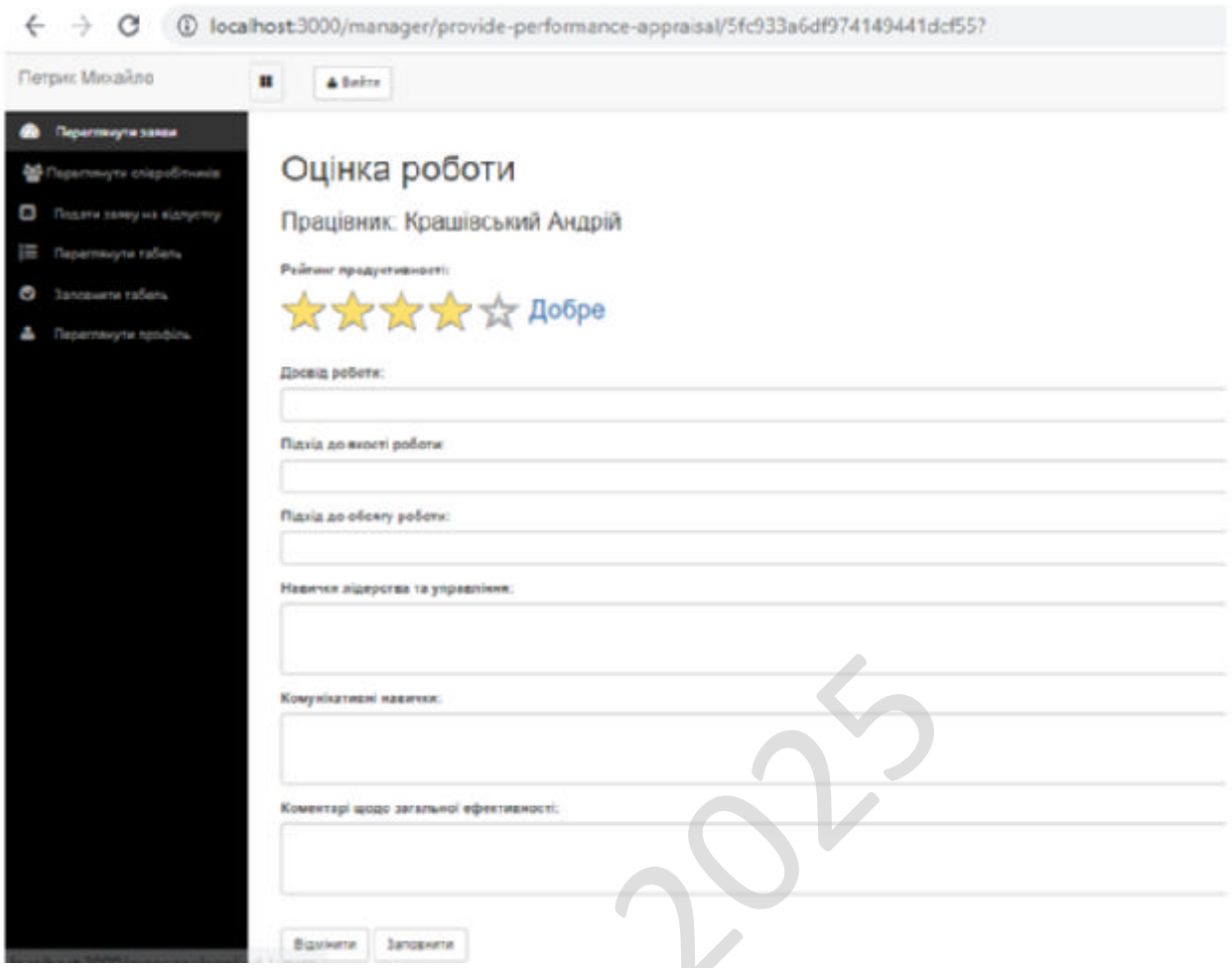



Рисунок 5.8 – Форма оцінки результативності роботи працівників

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.

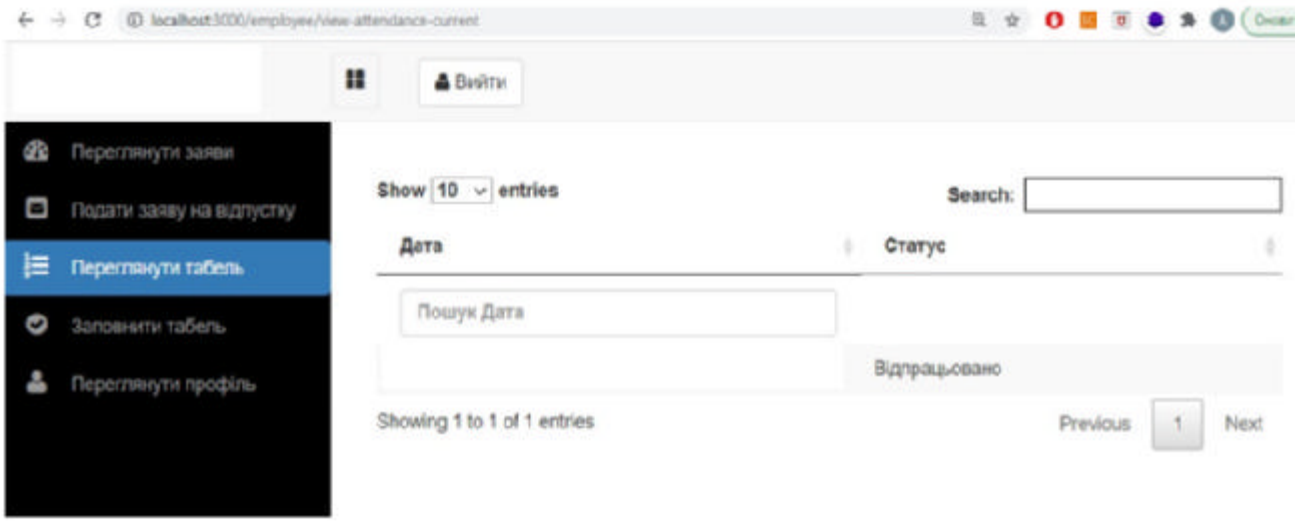



Рисунок 5.9 – Сторінка перегляду відпрацьованого часу працівником

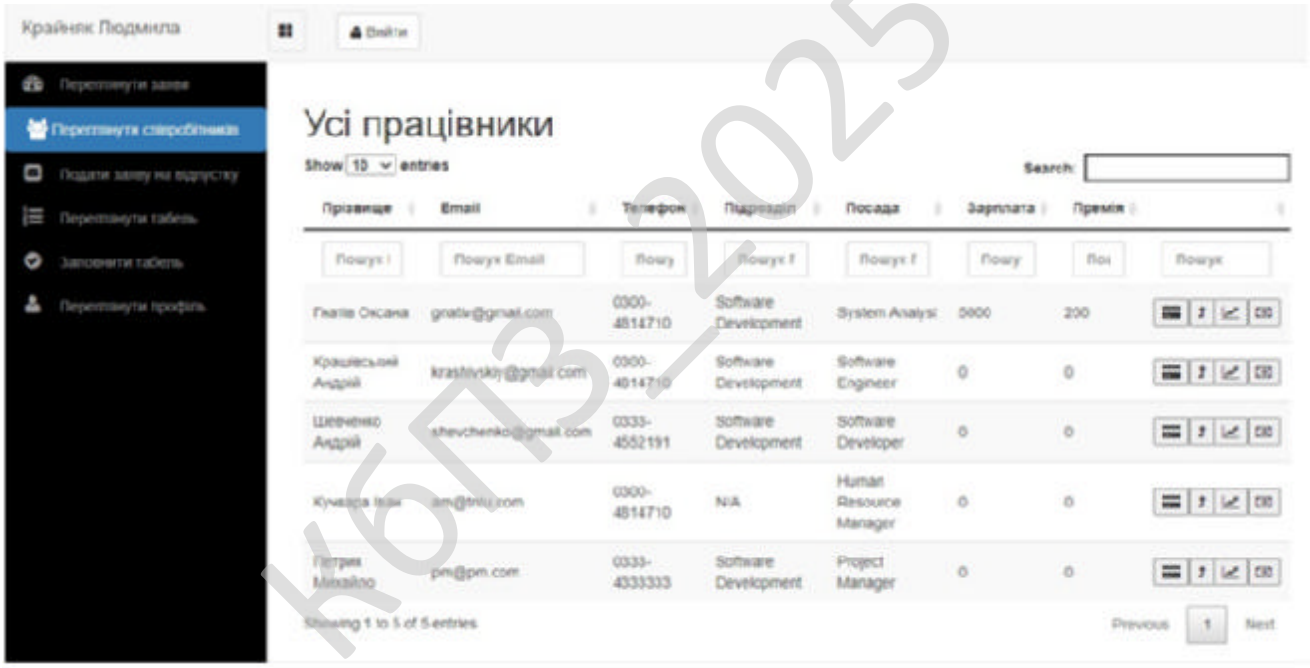



Рисунок 5.10 – Сторінка обліку оплати праці співробітників

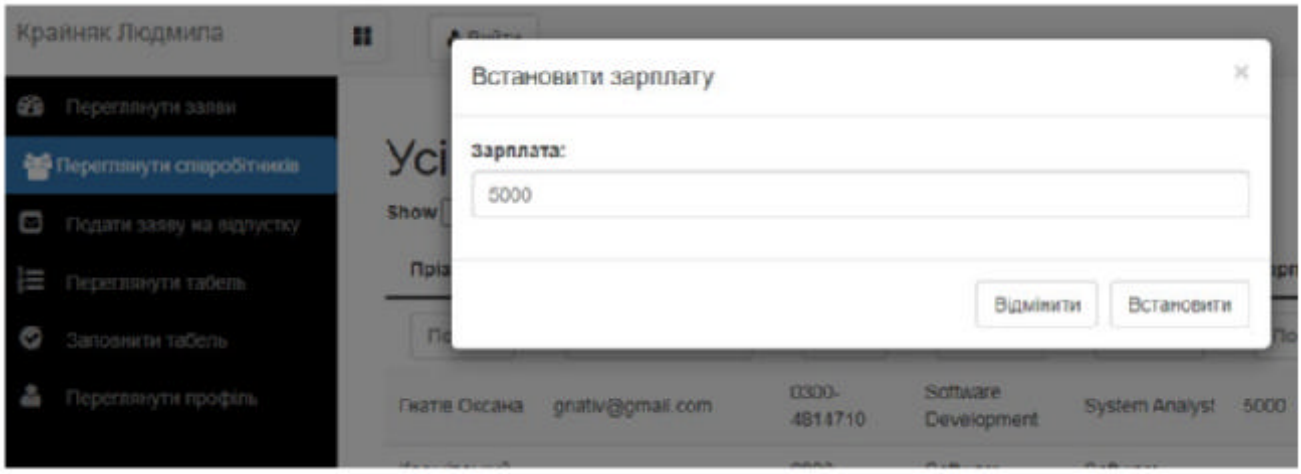



Рисунок 5.11 – Сторінка встановлення зарплати для окремого працівника

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.

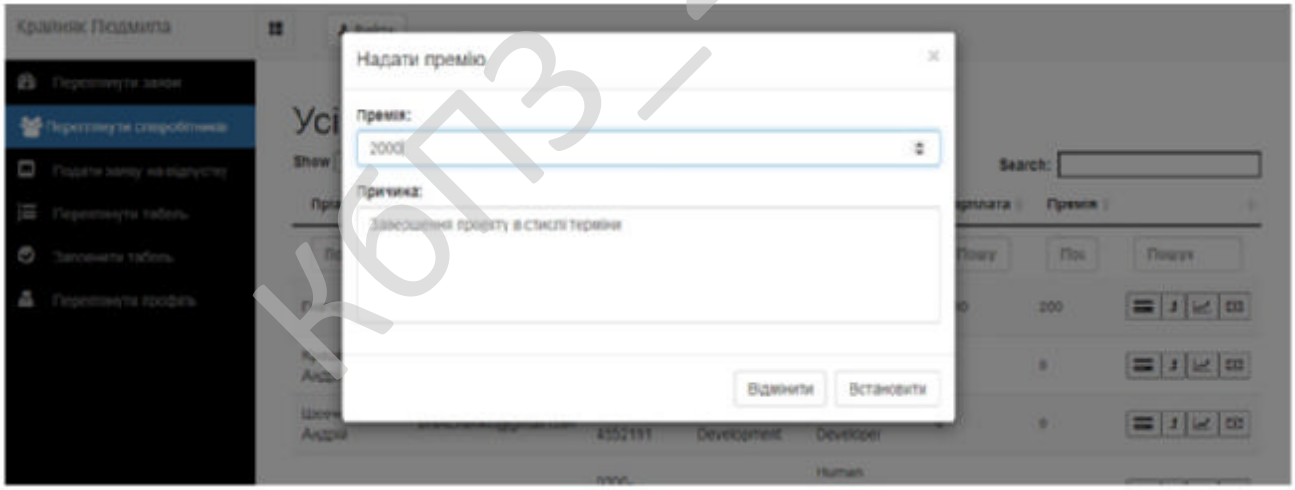



Рисунок 5.12 – Сторінка встановлення премії для окремого працівника

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 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної 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.

Завершуючи роботу, можна прийти до висновку, що SQL - це високорівнева мова запитів, призначена для роботи з базами даних. Вона дозволяє модифікувати дані, складати і виконувати запити, виводити результати у вигляді звітів.

Система управління базами даних PostgreSQL, що є однією з найрозвиненіших в своїй категорії, дозволяє повноцінну реалізацію баз даних на основі SQL, забезпечує всі стандарти SQL, підтримує великий набір вбудованих типів даних, більш того, користувач може створювати нові необхідні йому типи.

У даній роботі ми розглянули особливості побудови та роботи з об’єктно-реляційною моделлю даних в інструментальній системі управління базами даних PostgreSQL.

Переваги розробленої бази даних полягають у можливості легкого оновлення даних. Її структура передбачає зберігання даних за довгий період, причому завдяки всього лише кільком змінам вони не втрачають актуальності.

У 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 ОРБД.

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 NodeJS, з підтримкою бази даних PostgreDB;

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

Розроблені під час виконання роботи додатки дозволяють чітко зрозуміти для яких проектів краще підходять реляційні а коли ОРБД, 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.

Робота може бути удосконалена шляхом створення форм введення даних, з можливістю виконувати запити, проводити масову корекцію даних та створювати звіти. Також можна забезпечити захист системи та оптимізувати її роботу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Мова програмування JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.ua/

2. Introduction to Node.js [Електроний ресурс] – Режим доступу до ресурсу:https://nodejs.dev/

3. Серверне програмування веб-сайтів [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Server-side

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

5. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих // “Node.js в действии”.

6. Янг А., Мек Б., Кантелон М. // “Node.js в дії. 2-е видання”.

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/

21.Тренди веб-разробки [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/451572/

22.Веб додаток [Електронний ресурс] – Режим доступу до ресурсу: https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/

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

47. JavaScript Frameworks 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/migrate-mongo

48. Web Technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/web-technology/

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

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

Додаток А
(обов’язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення інформаційної системи побудованої на основі об’єктно-реляційної СУБД PostgreSQL.

Підставою для розробки служить завдання на випускну кваліфікаційну роботу, видане на кафедрі програмування та захисту інформації (нак. №46-02 від 17.01.2025 року).

## 3 Мета та призначення розробки

Метою випускної кваліфікаційної бакалаврської роботи є розробка програмна реалізація ІС додатку з використанням об’єктно-реляційної PostgreSQL.

## 4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської дипломної 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

При розробці ПЗ потрібно використовувати наступні технології та мови програмування: Середовище NodeJS та ОРСУБД PostgreDB.

Середовище програмування – PHP-Shtorm.

#### 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 Перелік документів, що розробляються

– Структурна схема системи керування – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок схема алгоритму програми – 1 аркуш.
– Блок-схема підпрограми – 1 аркуш.
– Пояснювальна записка – 78 аркушів.

## 8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи.

Постановка задачі на виконання кваліфікаційної роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної роботи.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 9 Порядок контролю та приймання

9.1 Подання кваліфікаційної роботи на попередній захист 20.05.2025 р.

9.2 Подання кваліфікаційної роботи на захист 05.06.2025 р.

Додаток Б
(обов’язковий)

**Міністерство освіти і науки України**
**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ
Керівник кваліфікаційної бакалаврської роботи
_____ Босько В.В

# Програмне забезпечення інформаційної системи побудованої на основі об’єктно-реляційної СУБД PostgreSQL

Лістинг програми

Код документу 12
Носій: CD/DVD-диск

Загальна кількість аркушів: 15

Літера: РП



Кропивницький – 2025 року

**Admin.as**

```
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('/');
}
```