Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка інформаційної системи для управління відпустками»

Виконав: студентка 2 курсу, групи 1ІСТ-23м спеціальності 126 – Інформаційні системи та технології
(шифр і назва спеціальності)

Світлана ГОЛОД
(ПІБ студента)

Керівник: к.т.н., доцент кафедри АІІТ
Ілона БОГАЧ
(науковий ступінь, вчене звання / посада, ПІБ керівника)

«16» грудня 2024 р.

Опонент: к.т.н., доцент каф.КН
Людмила КРИЛИК
(науковий ступінь, вчене звання / посада, ПІБ опонента)

«16» грудня 2024 р.

Допущено до захисту
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
(науковий ступінь, вчене звання)

«16» 12 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти II-ий (магістерський)
Галузь знань – 12 – Інформаційні технології
Спеціальність – 126 – Інформаційні системи та технології
Освітньо-професійна програма – Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«19» 09 2024 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Голод Світлані Вікторівні
(ПІБ автора повністю)

1. Тема роботи: Розробка інформаційної системи для управління відпустками
   Керівник роботи: к.т.н., доцент каф. АІІТ Богач І.В.
   Затвердженні наказом ВНТУ від «17» вересня 2024 року № 310.
2. Строк подання роботи студентом: до «16» 12 2024 року.
3. Вихідні дані до роботи: Оперативна пам'ять від 8гб, операційна система Windows 10 та вище, Node.js v18.12.0, React v18.0, інформація про працівників
4. Зміст текстової частини: Вступ; Аналіз предметної області та аналогів системи; Огляд програмних засобів для реалізації поставленої задачі; Розробка та тестування програмного забезпечення; Економічний розділ; Висновки; Список використаних джерел.
5. Перелік ілюстративного (або графічного) матеріалу: схема алгоритму роботи інформаційної системи для управління відпусток; модель бази даних інформаційної системи для управління відпусток; UML-діаграма прецедентів інформаційної системи для управління відпусток; UML-діаграма діяльності процесу створення запиту на відсутність; UML-діаграма класів інформаційної системи для управління відпусток

6. Консультанти розділів роботи

Розділ змістової частини роботи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 3	Ілона БОГАЧ, к.т.н., доцент кафедри АІІТ	18.09.24	16.11.24
4	Олександр ЛЕСЬКО, к.е.н., проф. каф. ЕПтаВМ	20.09.24	28.11.24

7. Дата видачі завдання: «18» 09 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	до 02.10.2024 р.	виконано
2	Обгрунтування вибору технологічного стеку	до 10.10.2024 р.	виконано
3	Розробка програмного забезпечення	до 10.11.2024 р.	виконано
4	Тестування розробленого програмного забезпечення	до 18.11.24 р.	виконано
5	Підготовка економічної частини	до 28.11.24 р.	виконано
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	до 30.11.24 р.	виконано
7	Попередній захист роботи	до 05.12.24 р.	виконано
8	Захист роботи		

Студент _____ (підпис) Світлана ГОЛОД (прізвище та ініціали)

Керівник роботи _____ (підпис) Ілона БОГАЧ (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Голод С. В. Розробка інформаційної системи для управління відпусками. Магістерська кваліфікаційна робота зі спеціальності 126 – Інформаційні системи та технології, освітня програма – Інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2024. 111 с.

На укр. мові. Бібліогр.: 33 назв; рис.: 24; табл. 25.

Магістерська кваліфікаційна робота присвячена розробці інформаційної системи для управління відпустками, яка спрямована на оптимізацію кадрового обліку. Запропонована система дозволяє підвищити ефективність управління кадровими ресурсами, забезпечити прозорість процесів ухвалення рішень і скоротити часові витрати на обробку заявок.

У даній роботі детально досліджується предметна область управління відпустками, здійснюється аналіз аналогічних систем. Особливу увагу приділено проектуванню бази даних, вибору відповідного технологічного стеку, реалізації серверної та клієнтської частин розроблюваної системи. Завершальним етапом роботи є тестування програмного забезпечення для забезпечення його відповідності заданим вимогам.

Ключові слова: інформаційна система, управління відпустками, оптимізація, MERN.

ABSTRACT

Holod S. V. Development of an Information System for Leave Management. Master's Qualification Thesis in the specialty 126 – Information Systems and Technologies, educational program – Information Technologies for Data and Image Analysis. Vinnytsia: VNTU, 2024. 111 p.

In Ukrainian language. Bibliography: 33 titles; Fig.: 24; table 25.

The master's qualification thesis is dedicated to the development of an information system for leave management, aimed at optimizing human resource accounting. The proposed system enhances the efficiency of human resource management, ensures transparency in decision-making processes, and reduces the time required to process leave requests.

This work provides an in-depth study of the domain of leave management and includes an analysis of analogous systems. Particular attention is given to the design of the database, the selection of an appropriate technology stack, and the development of the server and client parts of the system. The final stage of the work involves testing the software to ensure its compliance with the specified requirements.

Keywords: information system, leave management, optimization, MERN.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасних умовах цифрової трансформації організацій особливого значення набуває автоматизація управлінських процесів, зокрема управління відпустками, що є важливим елементом ефективного використання трудових ресурсів. Традиційні методи, які базуються на ручній обробці заявок або використанні електронних таблиць, є застарілими, часто призводять до помилок, викликаних людським фактором, а також не забезпечують необхідного рівня прозорості й точності. Впровадження інформаційних систем в процес управління відпустками дозволить вирішити ряд можливих проблем, наприклад, конфлікти в плануванні відпусток під час високих навантажень, недостатню видимість запитів, а також незаплановані відсутності.

Крім того, важливо враховувати зростаючі вимоги щодо дотримання трудового законодавства, яке регламентує порядок надання відпусток та інших соціальних пільг. Своєчасне та коректне управління відпустками є необхідною умовою для забезпечення відповідності нормативним вимогам, зокрема в частині надання гарантованих відпускних днів. Автоматизовані інформаційні системи надають можливість спростити та оптимізувати ці процеси, забезпечуючи точний облік робочого часу, оперативне подання заявок, їх обробку та затвердження, а також формування аналітичної звітності, що сприяє ефективному управлінню персоналом.

Таким чином, впровадження інформаційних систем для автоматизації управління відпустками не лише сприяє підвищенню продуктивності праці та мінімізації адміністративного навантаження, але й дозволяє організаціям отримувати надійну статистичну інформацію для подальшого аналізу й прийняття обґрунтованих управлінських рішень.

Метою роботи є розширення функціональних можливостей інформаційної системи для управління відпустками, враховуючи необхідність

підтримки гнучкості внутрішніх політик відсутностей, що дозволить конкурувати з іншими системами-аналогами на ринку.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

1. Провести аналіз існуючих підходів до управління відпустками в сучасних інформаційних системах;

2. Провести огляд технологій для побудови клієнт-серверних додатків;

3. Спроєктувати та розробити веб-додаток для управління відпустками;

4. Провести тестування розробленого програмного забезпечення.

Об'єктом дослідження є процес управління відпустками в організаціях.

Предметом дослідження є методи та програмні засоби автоматизації управління відпустками.

Методи дослідження. У роботі використовуються такі методи дослідження як аналіз, порівняння, опис, моделювання, узагальнення та методи представлення результату.

Науково-технічний результат полягає в оптимізації алгоритмів обробки заявок на відсутність для підвищення швидкості та точності управління персоналом, а також в можливості інтеграції системи з ERP-системою IT-компанії, що дозволяє синхронізувати управління відпустками з кадровим обліком і фінансовими розрахунками.

Практична цінність роботи полягає у створенні інформаційної системи для управління відпустками, яка автоматизує процеси подання та обробки заявок, забезпечує можливість аналізу використання відпусток і лікарняних, а також дозволяє легко модифікувати та адаптувати систему відповідно до потреб організації.

Апробація результатів роботи. Основні положення й результати роботи були представлені автором на Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025) [1] та попередні дослідження також було представлено на LII науково-технічної конференції підрозділів Вінницького національного технічного університету [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛОГІВ СИСТЕМИ

1.1 Види відпусток передбачені законодавством України

Згідно з Конституцією України, Кодексом законів про працю України, та Законом України про «відпустки», кожен співробітник може розраховувати на відпустку із збереженням грошової компенсації, а також на не оплачувану відпустку за власним бажанням [3].

До основних типів відпусток відноситься щорічна відпустка, додаткова відпустка у зв’язку із навчанням, відпустка без збереження заробітної плати, творча та соціальна відпустки. Кожен із типів відпусток має встановлені терміни та умови надання.

Щорічна відпустка поділяється на два види – основна і додаткові. Розглянемо правила надання основного виду щорічної відпустки:

– Тривалість не менше ніж двадцять чотири календарних дні на рік з моменту укладення трудового договору, однак для працівників віком до вісімнадцяти років тривалість даного виду відпустки становить тридцять один календарний день;

– Тривалість щорічної оплачуваної відпустки збільшується залежно від стажу роботи, відповідно до вимог національного законодавства або внутрішніх правил організації [4];

– Працівник, що працює перший рік у організації може використати повну тривалість відпустки після півроку безперервної роботи.

Щорічна додаткова відпустка за умови шкідливого чи важкого характеру праці надається працівникам, що працюють в умовах, шкідливих для здоров'я, згідно з переліком, затвердженим Кабінетом Міністрів України. Тривалість даного виду відпустки може досягати тридцяти п’яти календарних днів.

Тривалість щорічної додаткової відпустки за особливий характер роботи визначається через колективний чи трудовий договір і може досягати тридцяти

п’яти днів для працівників, що працюють у важких фізичних або інтелектуальних умовах. Для тих, хто працює в умовах ненормованого робочого дня, відпустка може становити до семи днів.

Відпустка пов’язана із навчанням може надаватися тим працівникам, які навчаються в навчальних закладах різного рівня, що залежать від форми навчання та виду закладу. Терміни відпустки варіюються.

Працівники, які займаються підготовкою дисертацій, підручників або іншої інтелектуальної діяльності можуть розраховувати на творчу відпустку, терміни якої обговорюються в індивідуальному порядку.

Соціальні відпустки — це категорія відпусток, що надаються працівникам для вирішення питань, пов'язаних із сімейними обов'язками, здоров'ям або соціальними потребами. Вони включають відпустки при вагітності та пологах, для догляду за дитиною, а також додаткові відпустки для батьків, які мають дітей.

Розглянемо детальніше типи та умови надання соціальних відпусток:

– Відпустка при вагітності та пологах. Тривалість даної відпустки складає до сімдесяти календарних днів до пологів та п’ятдесят шість після них. За певних умов, якщо пологи ускладнені, можливе подовження терміну до семидесяти днів.

– Співробітникам, які є матерями, надається відпустка для догляду за дитиною до досягнення нею тьох років.

– Додаткові відпустки для працівників, які мають дітей. Відповідно до чинного законодавства, матері, які мають двох і більше дітей віком до п’ятнадцяти років, або батьки-одинаки, можуть отримати додаткову відпустку тривалістю до десяти днів.

У разі, передбаченому трудовим законодавством, працівники можуть отримати відпустку без збереження заробітної плати. Даний тип відпустки надається як за сімейними обставинами, так і з інших причин, якщо угода між працівником і роботодавцем не передбачає іншого терміну. Однак, така відпустка має обмеження у розмірі п’ятнадцяти календарних днів на рік, за вийнятком періоду дії карантину.

Відпустки надаються згідно з графіками, затвердженими власником та профспілкою. Можливий поділ щорічної основної відпустки працівником із неподільною частиною у розмірі чотирнадцяти календарних днів.

Детальна узагальнена схема класифікації відпусток залежно від їх призначення та правового регулювання наведена на рисунку 1.1 [5].

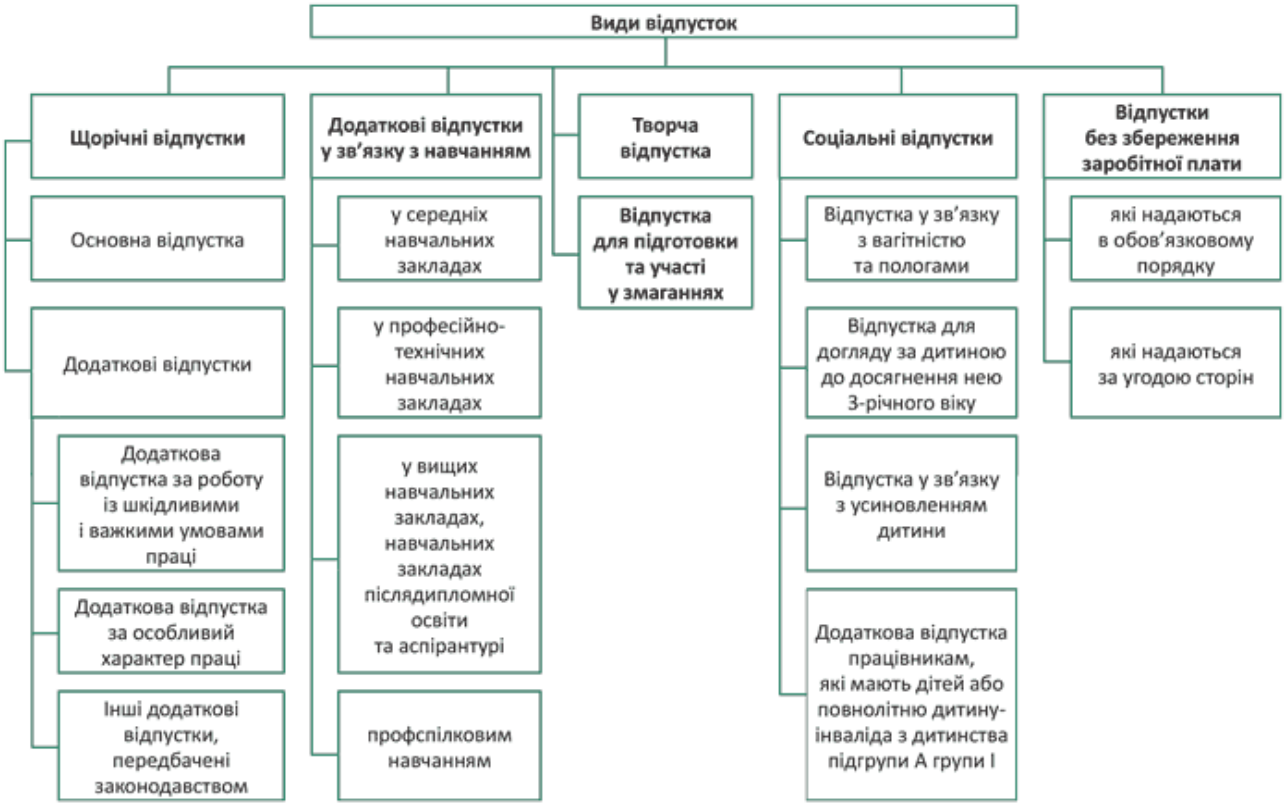

Рисунок 1.1 – Класифікація відпусток

Узгодження графіка відпусток є важливим процесом, що вимагає врахування як інтересів працівника, так і потреб організації. У випадках, коли кілька працівників подають запити на відпустку одночасно, пріоритет може надаватися особам із більшим трудовим стажем, вагомими соціальними підставами чи важливими сімейними обставинами.

Особливе значення має організація відпусток у критичних сферах, таких як медицина, енергетика чи транспорт, де відсутність працівників може безпосередньо вплинути на безпеку або безперервність роботи. У таких випадках роботодавці зобов'язані забезпечити ефективне планування заміщень.

Окрім основних видів, існують також спеціалізовані відпустки для певних категорій працівників. Наприклад, учасники бойових дій, особи з інвалідністю внаслідок війни або працівники, які проходять реабілітацію після травм, можуть отримати додаткові пільгові відпустки.

Щодо соціальних відпусток, законодавство України передбачає механізми захисту прав працівників, наприклад:

– Роботодавець не має права відмовити працівниці у відпустці по вагітності та пологах чи в догляді за дитиною.

– Чоловіки також мають право на відпустку для догляду за дитиною, що забезпечує рівність прав між статями.

Для уникнення конфліктних ситуацій між працівниками та роботодавцями, трудове законодавство передбачає чіткі правила надання відпусток, включно з необхідністю завчасного інформування про бажання скористатися відпусткою, зазвичай не менше ніж за два тижні.

Роботодавець має право передбачити в колективному договорі, внутрішніх нормативних актах або в індивідуальному трудовому договорі додаткові види відпусток, які не охоплені чинним законодавством. Такі відпустки можуть надаватися за погодженням сторін і враховують специфіку діяльності підприємства або потреби працівників. Наприклад, це можуть бути відпустки для участі в конференціях, волонтерських програмах, чи відпочинку у зв'язку із професійними досягненнями.

Надання таких відпусток дозволяє роботодавцям підвищувати лояльність працівників, мотивувати співробітників до ефективнішої праці та сприяти створенню позитивного корпоративного середовища.

Таким чином, законодавство України детально регулює порядок надання відпусток, їхні види та умови, забезпечуючи працівникам можливість поєднувати роботу з особистими потребами. Крім основних видів, роботодавці можуть запроваджувати додаткові відпустки, що сприяє підвищенню лояльності працівників і створенню комфортних умов праці.

1.2 Облік відпусток у IT-компаніях

IT-компанії, як правило, мають більш гнучкі підходи до обліку відпусток, особливо з огляду на специфіку віддаленої роботи. У більшості випадків компанії дозволяють співробітникам працювати з дому, що іноді призводить до необхідності перегляду політики відпусток для забезпечення оптимального балансу між роботою та відпочинком.

Безлімітні відпустки є новою тенденцією в управлінні персоналом, що набуває популярності серед IT-компаній та організацій, що орієнтуються на гнучкість і лояльність до своїх працівників. Цей підхід дозволяє співробітникам брати відпустку в будь-який час і на будь-який термін, не обмежуючи їх кількістю днів, на відміну від традиційних політик із визначеною кількістю днів відпустки на рік.

Вперше такий підхід до обліку відпусток запропонувала компанія Netflix у 2004 році. Після чого безлімітні відпустки впровадили Uber, GitHub, Dropbox та інші, за умови збереження продуктивності роботи команди. Безлімітне нарахування відпусток набуває популярності і в Україні, наприклад, компанії YouScan, eTeam, HealthJoy та iDeals Solutions успішно впровадили даний підхід у HR-процеси.

Однак є й негативний досвід даного виду відпусток. Наприклад компанія Kickstarter повернулася до традиційного лімітованого нарахування відпусток після аналізу використання відпусток. Членам команди не було зрозуміло, скільки днів відпустки вважатиметься прийнятним для використання, тому вони відпочивали навіть менше, ніж під час нарахування чітко визначених доступних відпускних днів [6].

Існують різні підходи до нарахування та обліку відпусток у IT-компаніях. Дослідження проведене у 2020 році компаніями LvBS і Clario показує статистичні дані обліку відпусток у IT-компаніях України. У даному дослідженні прийняли участь 157 респондентів з різних регіонів України [7].

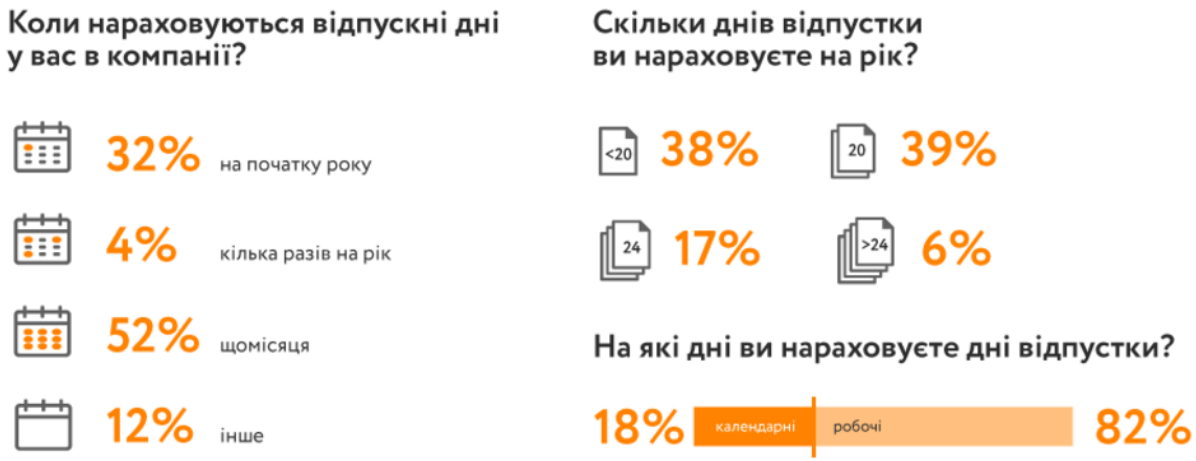

Рисунок 1.2 – Результати дослідження «Відпустки у компаніях». Частина 1

Як бачимо з інфографіки у IT-компаніях України основна щорічна відпустка у кількості 24 дні гарантовані законодавством надається лише у 17% опитаних компаній, у більшості ж випадків розмір відпустки становить 20 і менше днів. Значна частина компаній нараховують відпускні дні щомісяця, які доступні на робочі дні, не врахочуючи вихідні.

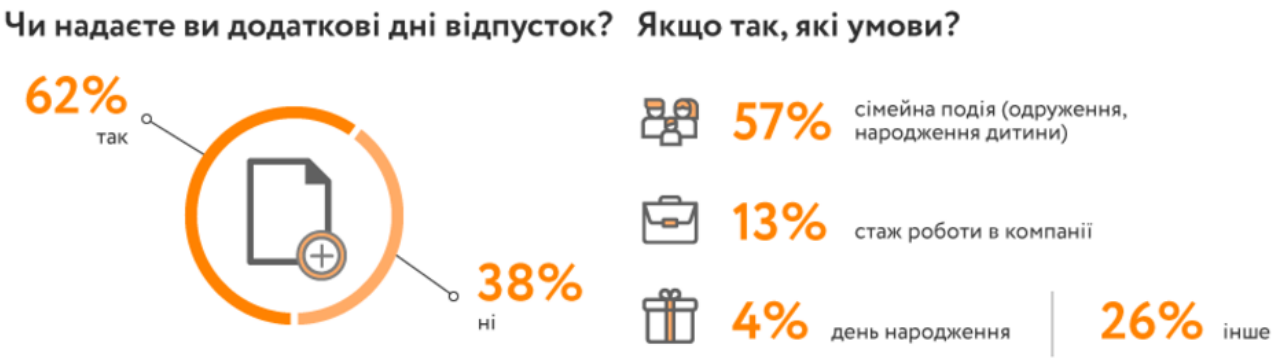

Рисунок 1.2 – Результати дослідження «Відпустки у компаніях». Частина 2

Також більшість компаній надає додадкові відпустки, здебільшого у зв’язку із сімейними подіями – такими як одруження, народження дитини та інші. Важливим моментом є компенсація накопичених відпускних днів при звільнені працівника. Згідно з результатами дослідження повністю компенсується лише частина відпустки (у 60% опитаних), у 14% невикористані дні не компенсуються взагалі.

Облік відпусток у ІТ-компаніях відрізняється від традиційних галузей через специфіку робочого процесу, гнучкий графік та можливість віддаленої роботи. Основні особливості цього обліку включають:

– Гнучкість у нарахуванні та використанні відпусток. У ІТ-компаніях часто застосовуються більш гнучкі моделі нарахування відпусток, де співробітники можуть використовувати відпустку за власним графіком, зберігаючи баланс між роботою та особистим життям. Це також може включати політику безлімітних відпусток, де кількість днів відпустки не обмежена, але вимоги щодо продуктивності та виконання завдань залишаються незмінними.

– Автоматизація процесу управління. Для забезпечення точного обліку та мінімізації адміністративного навантаження, ІТ-компанії активно використовують спеціалізовані інформаційні системи, які автоматично обчислюють кількість відпусток, враховують відсутність за різними причинами, а також генерують звіти для керівництва. Такі системи дозволяють зберігати прозорість і точність даних щодо відпусток.

– Зв'язок з віддаленою роботою. Оскільки багато ІТ-компаній працюють в режимі віддаленого доступу, це створює необхідність адаптації політики відпусток до нових умов. Наприклад, система обліку може враховувати гнучкий робочий час або надання відпусток залежно від особистих потреб співробітника, без прив'язки до традиційного офісного часу.

– Індивідуальний підхід до працівників. ІТ-компанії часто адаптують політику відпусток до індивідуальних потреб своїх співробітників, таких як сімейні обставини, особисті проекти чи участь у розвитку нових технологій. Це дозволяє створити умови для збереження мотивації продуктивності працівників.

Таким чином, сучасні підходи до управління відпустками в ІТ-компаніях, зокрема гнучкість у нарахуванні та використанні відпусток, врахування специфіки віддаленої роботи та індивідуальних потреб співробітників, роблять необхідним впровадження ефективних інструментів для обліку та автоматизації цих процесів.

1.3 Аналіз аналогів розроблюваної системи

У даному підрозділі проведено огляд існуючих систем для управління відпустками, які широко використовуються у сучасних організаціях. Аналіз аналогічних рішень дозволяє визначити сильні та слабкі сторони існуючих систем, а також виявити можливості для вдосконалення. Це також допомагає зрозуміти, які функціональні можливості є найбільш затребуваними у таких системах, і на що варто звернути увагу при розробці власного рішення.

1.3.1 Система керування HR-процесами Hurma

Hurma – це комплексна система для управління HR-процесами, включно з управлінням відпустками та лікарняними. Вона автоматизує багато аспектів кадрового менеджменту, надаючи компаніям інструменти для планування та моніторингу робочого часу співробітників, обліку відпусток, відстеження ефективності персоналу, а також найму і адаптації нових працівників. На рисунку 1.3 представлено приклад інтерфейсу системи [8].

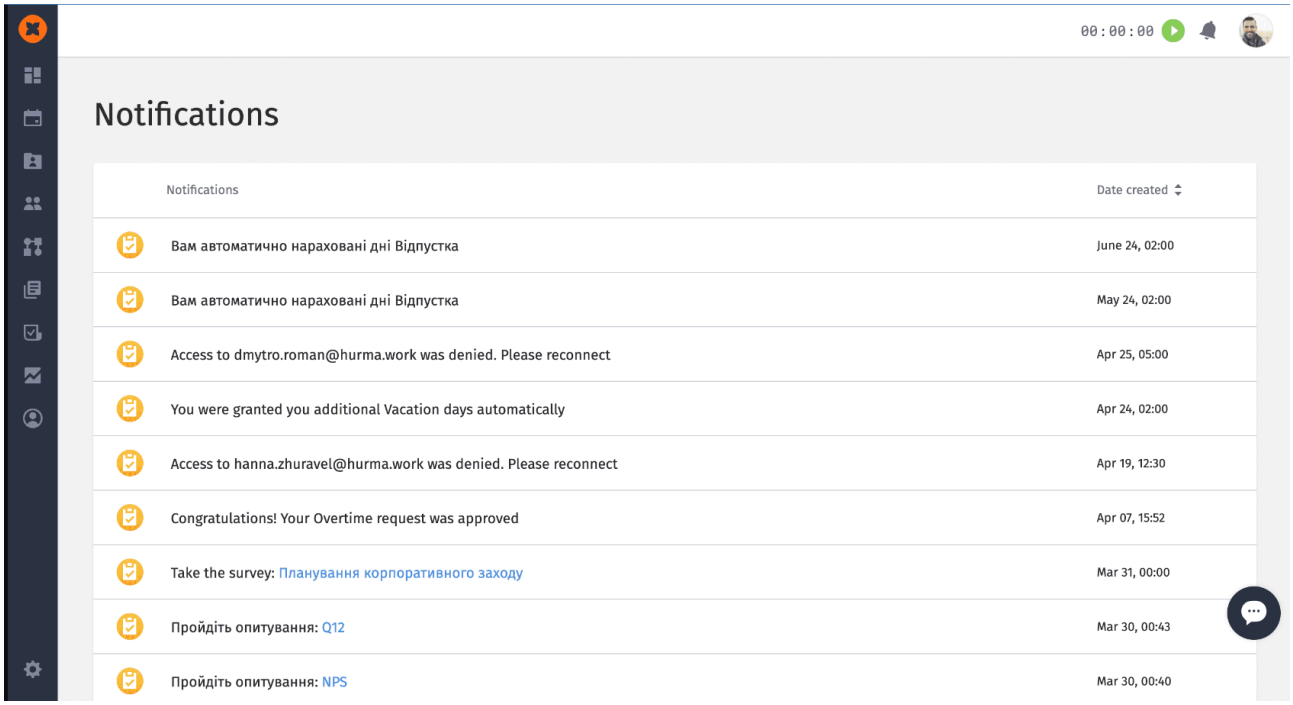

Рисунок 1.3 – Приклад інтерфейсу системи Hurma

Розглянемо переваги та недоліки даної системи. До переваг можна віднести:

– Інтуїтивно зрозумілий інтерфейс. Система має простий і зручний дизайн, що полегшує використання як для HR-відділу, так і для співробітників.

– Автоматизація процесів. Hurma автоматично обробляє запити на відпустки та лікарняні, надсилаючи повідомлення керівникам і HR-менеджерам, що знижує кількість ручної роботи.

– Інтеграція з іншими системами. Платформа легко інтегрується з різними інструментами для управління проектами та календарями, що спрощує планування відпусток у команді.

– Чіткий контроль відпусток співробітників. Програмне забезпечення дозволить відстежувати графіки відпусток усіх працівників в одному місці. Таким чином уповноважена особа може швидко переглянути заплановані відсутності всіх учасників команди або відділу та переконатися, що запити на відпустку взаємозамінних співробітників не збігаються.

– Візуалізація відсутностей у календарі програми. У календарі можна ознайомитися із графіками всіх колег – годинами роботи, запланованими зустрічами, вихідними та іншими видами відсутності.

До недоліків системи можна віднести:

– Ціна. Система може бути дороговартісною для малих підприємств, особливо з додатковими функціями або налаштуваннями.

– Складність налаштування. Початкове налаштування системи може зайняти значний час, особливо для компаній з великим штатом працівників або складною структурою.

– Обмежена кастомізація. Хоч Hurma пропонує широкий набір функцій, деякі користувачі зазначають обмежені можливості щодо гнучкого налаштування під специфічні потреби компанії.

Таким чином, Hurma є потужним інструментом для автоматизації HR-процесів, але для малих або специфічних компаній вона може бути надмірно складною та дороговартісною.

1.3.2 Система керування HR-процесами BambooHR

BambooHR – це популярна онлайн-система управління персоналом, яка забезпечує широкий спектр функцій для управління відпустками, рекрутингу, управління даними працівників та іншими HR-процесами. Система розроблена переважно для малих та середніх підприємств. На рисунку 1.4 наведено приклад інтерфейсу модулю управління відпустками системи BambooHR [9].

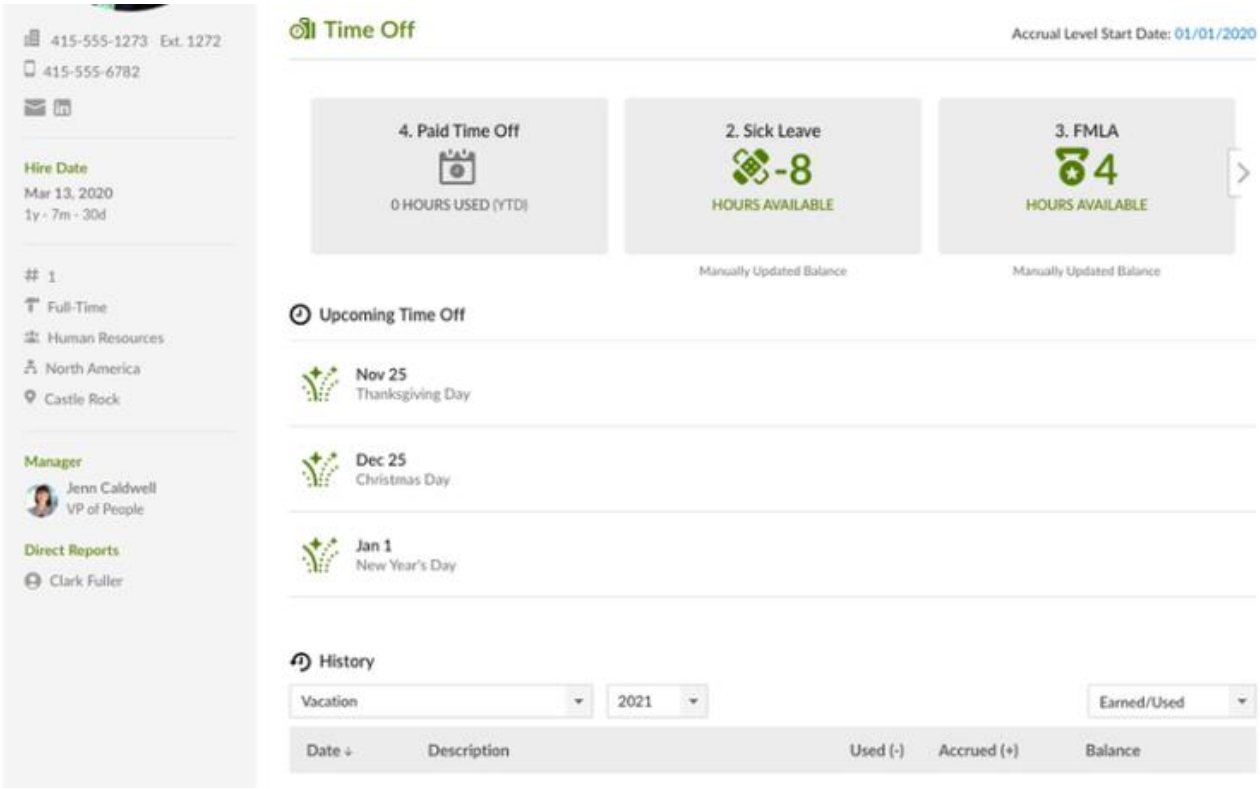

Рисунок 1.4 – Приклад інтерфейсу модулю управління відпустками системи BambooHR

Розглянемо переваги та недоліки даної системи. До переваг можна віднести:

– Облік відпусток (PTO tracking). Система автоматично розраховує, відстежує залишки днів відпустки та забезпечує зручність у поданні запитів на відпустку. Працівники можуть легко подавати запити, а керівники — відстежувати їх статус і узгоджувати.

– Управління зарплатами та пільгами. Платформа інтегрує облік зарплат і пільг в єдину систему. Це зменшує необхідність дублювання даних, дозволяючи легко розраховувати зарплати, коригувати податки та керувати пільгами в одному місці.

– Звітність та аналітика. BambooHR надає розширені аналітичні інструменти для формування звітів щодо відпусток, продуктивності працівників та інших HR-даних. Ця функція допомагає менеджерам приймати обґрунтовані рішення на основі даних.

– Покращення досвіду співробітників. Інструменти для оцінки задоволеності працівників і моніторингу продуктивності створюють прозорий процес управління персоналом. Система також забезпечує централізовану комунікацію, допомагаючи зміцнити зв'язок між працівниками та керівництвом.

До недоліків даної системи можна віднести:

– Незручність у розмежуванні ролей. Працівники не можуть виконувати базові дії, наприклад додати власне фото, оскільки такі зміни можуть вносити лише менеджери, що створює додаткові бар’єри в роботі.

– Випадки збоїв у системі знижують загальне задоволення від користування та негативно впливають на робочі процеси.

– Облік працівників, що залежить від початку календарного року, а не дати найму працівника.

BambooHR є однією з найбільш поширених HR-систем для малих і середніх підприємств, яка орієнтована на оптимізацію управління персоналом. Її функціонал охоплює широкий спектр задач: від обліку відпусток і вихідних до автоматизації рутинних HR-процесів. Крім того, забезпечує можливість формування детальних звітів і аналітики, що сприяє підвищенню ефективності прийняття управлінських рішень. Однак, як і будь-яка система, BambooHR має свої обмеження, які варто враховувати при виборі відповідного програмного забезпечення для управління персоналом, зокрема необхідність адаптації до специфічних потреб компанії та можливі обмеження в налаштуванні ролей.

1.3.3 Система керування HR-процесами PeopleForce

PeopleForce — це хмарна платформа для управління HR-процесами. Вона включає інструменти для рекрутингу, управління персоналом, аналітики, та автоматизації рутинних завдань. Система дозволяє створювати бази кандидатів, координувати робочі процеси в командах, та вести звітність. Вартість підписки варіюється залежно від обраного тарифу та модулю.

У складі даної платформи є модуль CoreHR, що забезпечує зокрема облік відпусток та відсутностей працівників. На рисунку 1.5 наведено приклад інтерфейсу модую для обліку вихідних та відпусток CoreHR від PeopleForce [10].

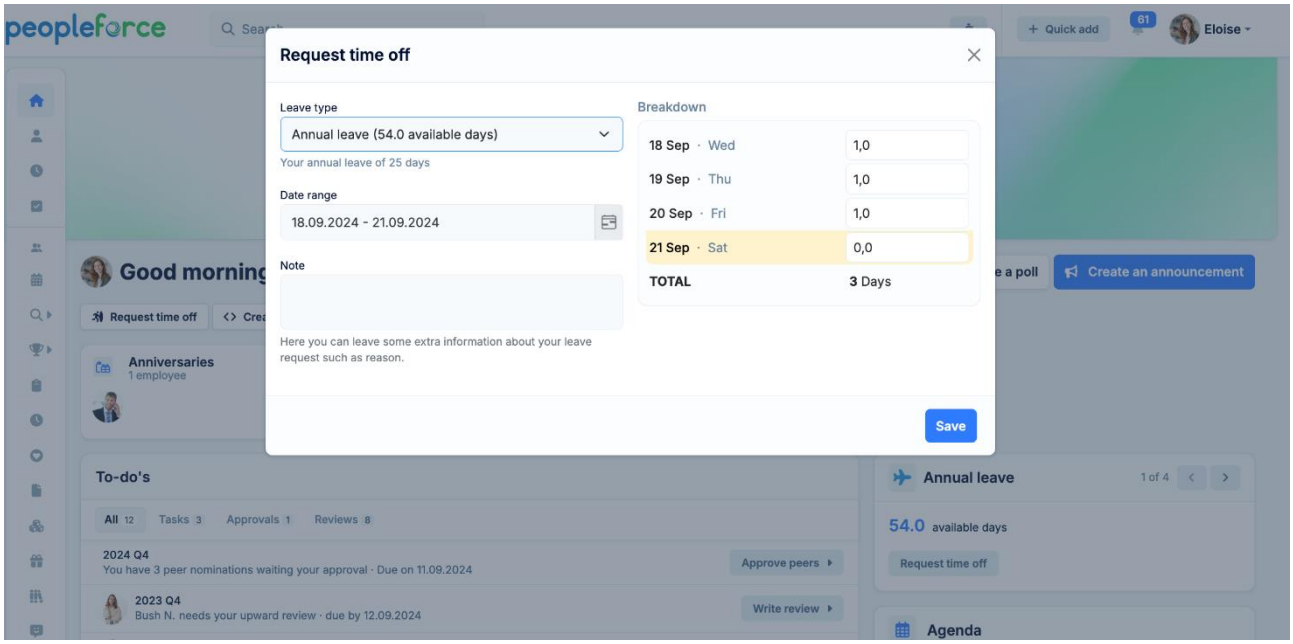

Рисунок 1.5 – Приклад інтерфейсу CoreHR від PeopleForce

Розглянемо можливості та основний функціонал модулю CoreHR від PeopleForce в розрізі управління відпусками:

– Створення запитів на відсутність як з головної сторінки так із профілю співробітника;

– Нотифікація про нові запити, зміну статусу існуючого та відповіді на електронну пошту співробітника;

– Автоматичний розрахунок балансу днів відпустки, залежно від обраного періоду, що дозволяє співробітникам легко відстежувати залишок доступних днів. Дана функція усуває ризик людських помилок на зменшує навантаження на HR-відділ.

– Можливість формувати звіти щодо використання відпусток, відсутностей, що допомагає в стратегічному плануванні ресурсів компанії;

– Представлення єдиного інтерактивного таймлайна, який об'єднує інформацію про відсутності співробітників.

Інтеграція модуля CoreHR з іншими HR-інструментами PeopleForce забезпечує цілісність і безперервність даних, що є важливим для ефективного управління персоналом. Завдяки даній можливості вся інформація про відсутності та відпустки співробітників автоматично інтегрується в загальну картину HR-процесів організації. Це дозволяє зберігати точність та актуальність даних, що є важливим для забезпечення ефективного управління робочими ресурсами.

Одним з основних недоліків PeopleForce є певна складність в адаптації платформи для невеликих компаній. Враховуючи те, що система пропонує великий набір функцій і модулів, для користувачів, які потребують базового функціоналу, може виникнути перевантаження інтерфейсу та налаштувань. Це може стати перешкодою для швидкого освоєння системи новими користувачами або малими командами, які не потребують всього спектра можливостей.

Також важливо зазначити, що PeopleForce є хмарною платформою, що може викликати проблеми для організацій, які мають обмеження на зберігання даних в хмарі через політики безпеки чи законодавчі вимоги. Це може стати бар'єром для впровадження платформи в деяких організаціях, що може вимагати додаткових витрат на адаптацію до вимог локального законодавства.

Ці фактори можуть стати перешкодою для компаній, які потребують більш доступних рішень для управління HR-процесами.

1.4 Порівняння можливостей запропонованого рішення з існуючими системами

У даному підрозділі проведено порівняння функціональних можливостей розробленої системи для управління відпустками з системами-аналогами, розглянутими у підрозділі 1.3. Враховуючи специфіку та вимоги до простоти й ефективності управління відсутностями, розроблювана система, має низку значних переваг у порівнянні з розглянутими рішеннями. Результати порівняльної характеристики наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння функціональних можливостей запропонованого рішення з системами-аналогами.

Платформа	Формування кількох політик	Інтеграція з ERP-системою організації	Відстеження запитів в реальному часі	Формування звітності
Hurma	Ні	Частково	Так	Так
BambooHR	Ні	Частково	Ні	Так
PeopleForce	Ні	Частково	Так	Так
Запропоноване рішення	Так	Так	Так	Так

Як бачимо, розглянуті системи-аналоги не мають можливості формування кількох політик відсутностей, натомість дана можливість реалізована у запропонованій системі, що дозволяє забезпечити гнучкий розподіл політик між співробітниками, які працюють як на повній, так і на частковій зайнятості, а також враховувати особливі потреби для певних категорій працівників. Усі аналоги мають можливість інтеграції з ERP-системою організації завдяки відкритому API, однак дане рішення потребує значних змін у структурі існуючої ERP-системи. Більшість аналогів забезпечують відстеження запитів на відсутність у реальному часі, як і запропоноване рішення, а також мають можливість формування звітності.

1.5 Висновки до розділу

У даному розділі проведено аналіз предметної області та систем-аналогів. В результаті аналізу предметної області визначено основні види відпусток, їхні умови та особливості надання відповідно до чинного законодавства. Також розглянуто сучасні підходи та особливості обліку відпусток в IT-компаніях, що дозволяє оцінити специфічні вимоги до автоматизації даного процесу в умовах сучасного ринку праці.

Дослідження систем-аналогів, зокрема Hurma, BambooHR та PeopleForce, виявило їхні переваги, зокрема автоматизацію процесів, інтеграцію з іншими HR-системами та зручність використання. Однак також виявлено аспекти, які потребують вдосконалення: складність адаптації цих систем до специфічних вимог окремих компаній, а також обмежені можливості гнучкої конфігурації політик відпусток.

У рамках порівняння також наведено характеристику запропонованого рішення, що дозволяє здійснити оцінку переваг і недоліків кожної з платформ. Порівняння показало, що запропоноване рішення має переваги у контексті гнучкості в налаштуванні політик відпусток, що забезпечує ефективність і зручність використання в специфічних умовах підприємств.

2 ОГЛЯД ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПОСТАВЛЕНОЇ ЗАДАЧІ

Важливим етапом розробки інформаційної системи для управління відпустками є вибір та обгрунтування технологій. Правильний вибір програмних засобів дозволяє мінімізувати ризики, пов'язані з проблемами сумісності, забезпечити гнучкість при впровадженні нових функціональних можливостей, а також скоротити час на розробку та тестування.

У даному розділі розглянуто технологічний стек MERN, який обрано для розробки інформаційної системи. MERN включає чотири основні компоненти: MongoDB, Express.js, React та Node.js [11]. Кожен з цих інструментів виконує специфічні функції, що забезпечують ефективну взаємодію між частинами системи та дозволяють створювати продуктивні, масштабовані веб-застосунки.

Основою технологічного стеку MERN є мова програмування JavaScript, яка забезпечує уніфікований підхід до розробки як клієнтської, так і серверної частин програмного забезпечення.

JavaScript — це високорівнева мова програмування з динамічною типізацією, орієнтована на подієво-кероване програмування та асинхронність. Вона є інтерпретованою та працює в середовищах виконання, таких як веб-браузери та Node.js. JavaScript підтримує об'єктно-орієнтовану, функціональну та імперативну парадигми програмування, що робить її універсальним інструментом для розробки як клієнтської, так і серверної частин застосунків.

Дана мова програмування використовується у різноманітних сферах розробки програмного забезпечення. Найбільш популяні напрямки та приклади розроблених засносунків наведено нижче:

– Соціальні мережі. Facebook, Twitter та Instagram, активно використовують JavaScript для реалізації динамічних інтерфейсів і оновлень у реальному часі;

– Односторінкові застосунки. Gmail, Google Maps та GitHub, застосовують JavaScript-фреймворки для створення односторінкових веб-систем;

– Платформи потокового мовлення. Netflix, Spotify та YouTube, використовують JavaScript для забезпечення плавного та інтерактивного стримінгового досвіду для своїх користувачів.

– Застосунки для співпраці. Slack і Trello, використовують JavaScript для створення інтерактивних середовищ із оновленнями в реальному часі.

– Інтернет-магазини. Amazon і eBay, застосовують JavaScript для створення динамічного, зручного інтерфейсу із функціями рекомендації товарів і оновлення даних у реальному часі.

– Фінансові платформи. PayPal та онлайн-банкінг, використовують JavaScript для забезпечення безпечних фінансових транзакцій і створення сучасного, привабливого інтерфейсу.

Незважаючи на значні переваги JavaScript існує її значний недолік – Динамічна типізація може призводити до помилок, що виявляються лише під час виконання кодц, ускладнюючи відлагодження системи підчас розробки. Для вирішення даної проблеми розроблено TypeScript.

TypeScript – це мова програмування з відкритим кодом, яка є надбудовою над JavaScript, забезпечуючи статичну типізацію та додаткові можливості для розробників. Вона транслюється у стандартний JavaScript, що забезпечує сумісність із будь-яким середовищем виконання JavaScript. TypeScript забезпечує кращу підтримуваність, масштабованість і надійність коду, що робить його особливо ефективним для великих команд і складних програмних проєктів.

TypeScript забезпечує статичну типізацію, дозволяючи визначати типи змінних, функцій і об’єктів, що значно спрощує виявлення помилок на етапі компіляції. Завдяки таким можливостям, як класи, інтерфейси, модулі та абстракції, забезпечується ефективна організація коду у великих проєктах.

Розглянемо детально кожну складову стеку технолгій MERN: їхні особливості та переваги, які обгрунтовують вибір для технологій для розробки.

2.1 MongoDB як система керування базою даних

NoSQL бази даних – тип баз даних, що відрізняються від традиційних реляційних систем управління базами даних і призначені для зберігання та обробки великих обсягів структурованих, напівструктурованих і неструктурованих даних. NoSQL підкреслює, що ці бази не обмежуються використанням SQL для маніпулювання даними, а також можуть включати різні моделі зберігання і доступу до даних.

MongoDB — це документно-орієнтована NoSQL система керування базами даних, яка зберігає дані у форматі JSON-подібних документів (BSON, що є розширенням формату JSON) [12].

Розглянемо переваги MongoDB:

1. Однією з головних переваг MongoDB є її схеми, які є динамічними, тобто дані можуть зберігатися без строго визначених схем. Це дозволяє зберігати документи з різними структурами в одній колекції, що особливо корисно для проектів, де структура даних може змінюватися.

2. MongoDB підтримує горизонтальну масштабованість через механізм шардінгу. Це дозволяє розподіляти дані між кількома серверами, що дає змогу ефективно працювати з великими обсягами інформації та забезпечує високу доступність системи.

3. Завдяки використанню індексів, MongoDB забезпечує високу швидкість операцій читання та запису, що особливо важливо для сучасних веб-застосунків із високим навантаженням.

4. MongoDB забезпечує зручну інтеграцію з різними мовами програмування та фреймворками, зокрема Node.js, що робить її гарним вибором для стеку MERN.

5. MongoDB надає вбудовану підтримку геопросторових індексів та запитів, що дозволяє зберігати та обробляти просторові дані, такі як координати для картографічних або локаційних застосунків.

6. MongoDB включає потужний механізм агрегації, що дозволяє обробляти та аналізувати великі обсяги даних за допомогою фільтрації, групування та виконання операцій без необхідності витягування даних на клієнтську сторону.

MongoDB підтримує широкий спектр типів даних, що дозволяє зберігати та обробляти різноманітну інформацію в гнучких документах [13]. Основні типи даних, які підтримує MongoDB, включають:

1. Числові типи (Int32, Int64 та Double – число з плаваючою комою подвійної точності)

2. Рядки (String) – текстовий тип даних, який зберігається у вигляді UTF-8 рядків. Використовується для зберігання текстових значень.

3. Булевий тип (Boolean) – логічний тип даних, що може зберігати значення true або false.

4. Масиви (Array) – тип даних, що дозволяє зберігати кілька значень одного типу або різних типів у вигляді впорядкованого списку.

5. Об'єкти (Embedded Documents) – вбудовані документи, що дозволяють зберігати структуровані дані в межах одного документа. Це забезпечує високу гнучкість при моделюванні складних структур даних.

6. Дата (Date) – тип даних, що зберігає час у мілісекундах з початку епохи Unix (1 січня 1970 року). Використовується для зберігання часових міток.

7. Null – тип даних, що представляє відсутність значення. Використовується для позначення порожніх або невизначених полів.

8. Бінарні дані (BinData) – тип даних для зберігання бінарних файлів або інших двійкових даних, таких як зображення чи файли.

9. ObjectId – спеціальний тип даних, який використовується для унікальної ідентифікації документів в колекціях MongoDB. Це 12-байтовий ідентифікатор, який зазвичай використовується як первинний ключ для документів.

10. Регулярні вирази (Regex), що дозволяє виконувати пошук за шаблонами в рядках.

11. JavaScript. MongoDB дозволяє зберігати функції або вирази JavaScript, що може бути корисно для виконання складних операцій безпосередньо на сервері.

12. Тимчасові типи (Timestamp) – тип даних для зберігання точних часових міток, що зазвичай використовуються для відстеження змін в колекціях або для реплікації.

13. GeoJSON – тип даних для зберігання геопросторової інформації у форматі GeoJSON, включаючи точку, лінію, полігон та інші географічні фігури.

Згідно з даними дослідження JetBrains MongoDB займає перше місце серед набільш використовуваних за 2022 рік як NoSQL база даних та четверте у загальному рейтингу баз даних [14]. Також варто зазначити, що порівняно з 2021 роком зменшився відсоток використання найбільш використовуваної бази даних MySQL (з 61% до 52%). На рисунку 2.1 наведено рейтинг баз даних за 2022 рік.

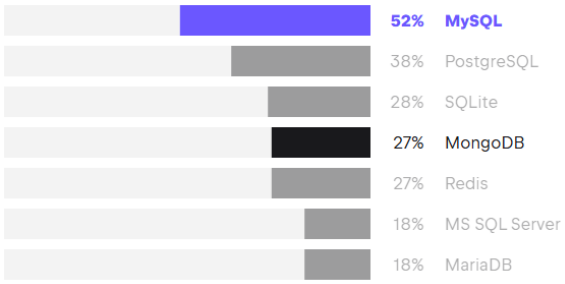

Рисунок 2.1 – Рейтинг популярності баз даних за 2022 рік

MongoDB продовжує набирати популярності серед розробників і підприємств, зберігаючи лідируючі позиції серед NoSQL баз даних (рис.2.2). За даними 2023 року, MongoDB не тільки залишається найбільш використовуваною серед NoSQL рішень, а й значно покращила свої позиції у загальному рейтингу баз даних, піднявшись з п’ятого на третє місце в порівнянні з 2021 роком. На рисунку 2.2 наведено рейтинг популярності баз даних за 2023 рік, де MongoDB демонструє значне зростання і займає третє місце серед усіх баз даних, що підкреслює її стабільний успіх та тенденцію до подальшого впровадження у глобальній IT-індустрії.

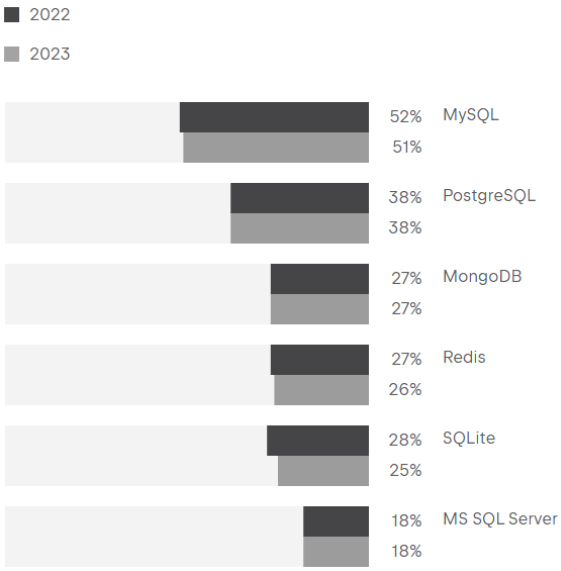

Рисунок 2.2 – Рейтинг популярності баз даних за 2023 рік

Зростання популярності MongoDB відображає загальну тенденцію в індустрії до вибору технологій, що забезпечують високу продуктивність та гнучкість у обробці даних, що постійно змінюються. Це робить MongoDB важливим інструментом для розробки майбутніх технологічних рішень, оскільки вона дозволяє ефективно працювати з великими обсягами неструктурованих та напівструктурованих даних, забезпечуючи при цьому масштабованість і адаптивність до змін у вимогах до зберігання та обробки інформації.

На додаток до згаданих переваг MongoDB, варто зазначити її активну роль у побудові хмарних рішень. Вона інтегрується з такими провайдерами, як AWS, Azure і Google Cloud Platform, забезпечуючи можливість автоматизованого масштабування, реплікації та резервного копіювання даних. Такі можливості дозволяють оптимізувати витрати на інфраструктуру та забезпечувати високу доступність навіть у розподілених системах.

Одним із ключових факторів популярності MongoDB є її орієнтованість на сучасні архітектури, зокрема мікросервіси. З технічної точки зору, MongoDB підтримує потужний механізм транзакцій, який дозволяє виконувати багатоетапні операції з гарантованою атомарністю навіть у розподілених системах.

2.2 Серверна платформа для розробки веб-застосунків Express.js

Express.js – фреймворк для створення веб-застосунків та API на платформі Node.js [15]. Завдяки своєму простому та лаконічному інтерфейсу, Express.js дозволяє розробникам швидко і ефективно створювати серверні рішення з високим рівнем налаштування та функціональності. Він є одним з найбільш популярних інструментів для розробки веб-серверів і є частиною технологічного стеку MERN, що робить його важливим компонентом для реалізації складних веб-застосунків.

Основними перевагами використання Express.js є:

1. Мінімалізм і гнучкість. Express.js пропонує простий API, що дозволяє зосередитись на логіці додатку, одночасно надаючи гнучкість для розширення функціональності. Це дозволяє зменшити кількість коду та полегшує підтримку програми, що є важливим аспектом при розробці складних веб-систем.

2. Підтримка маршрутизації. Express.js надає потужний механізм маршрутизації, що дозволяє ефективно керувати запитами від клієнта до відповідних обробників. Це забезпечує зручну і чітку організацію обробки різних HTTP-запитів (GET, POST, PUT, DELETE), що є важливим для побудови RESTful API або веб-застосунків.

3. Підтримка проміжних програм (middleware). Express.js дозволяє створювати проміжні програми, які обробляють запити до основних обробників. Проміжні програми можуть використовуватись для перевірки авторизації, логування запитів, обробки помилок, а також для реалізації інших необхідних функцій, що підвищує безпеку та зручність у роботі з додатком.

4. Масштабованість і висока продуктивність. Express.js є оптимальним рішенням для створення масштабованих серверних додатків. У поєднанні з Node.js, що забезпечує асинхронну обробку запитів, Express.js дозволяє створювати високопродуктивні системи, які можуть ефективно обробляти тисячі одночасних з'єднань.

5. Розширення функціональності через модулі. Express.js має велику екосистему модулів і пакетів, які дозволяють додавати додаткову функціональність до веб-застосунку, такі як обробка сесій, перевірка введених даних, підключення до баз даних тощо. Це значно прискорює процес розробки та дозволяє фокусуватись на специфічних вимогах проєкту.

6. Інтеграція з іншими технологіями. Оскільки Express.js є частиною стеку MERN, він безпосередньо інтегрується з MongoDB, React та Node.js. Це забезпечує єдність технологій у межах одного проєкту та дозволяє зменшити складність інтерфейсу взаємодії між компонентами.

Однією з ключових характеристик Express.js є його гнучкість у реалізації безпеки серверних додатків. Завдяки можливості інтеграції з різними модулями, такими як Helmet, можна легко додати заголовки HTTP, що підвищують захист від атак типу XSS, CSRF або clickjacking. Крім того, Express.js дозволяє налаштовувати політику CORS (Cross-Origin Resource Sharing), що особливо важливо для застосунків, які працюють із зовнішніми клієнтами чи API.

Express.js також підтримує сучасні веб-стандарти, включаючи WebSockets і Server-Sent Events (SSE), які забезпечують реальну багатокористувацьку взаємодію та передачу даних у режимі реального часу. Це робить його чудовим вибором для розробки чатів, стрімінгових платформ або систем моніторингу.

Ще одним важливим аспектом є можливості для роботи з асинхронними операціями. Express.js дозволяє легко інтегрувати асинхронні функції через підтримку promise-об’єктів і async/await синтаксису, що спрощує роботу з базами даних, сторонніми API або будь-якими іншими асинхронними джерелами даних.

Завдяки зазначеним перевагам, Express.js є потужним інструментом для розробки веб-сервісів, що потребують гнучкості, високої продуктивності та легкості в підтримці. Його використання в поєднанні з іншими технологіями, такими як Node.js, дозволяє досягти хороших результатів при створенні сучасних, масштабованих веб-додатків, що є важливим для реалізації інформаційних систем, які обробляють великі обсяги даних.

2.3 Бібліотека розробки користувацьких інтерфейсів React

React — це популярна бібліотека JavaScript, призначена для створення інтерфейсів користувача, розроблена і підтримувана компанією Meta [16]. Вона забезпечує ефективну розробку інтерактивних та динамічних веб-застосунків завдяки використанню декларативного підходу до побудови інтерфейсу користувача та компонентної архітектури. React дозволяє розробникам створювати компоненти, які зручно інтегруються між собою для реалізації складних інтерфейсів односторінкових веб-додатків.

Розглянемо переваги бібліотеки React:

– React створює віртуальний DOM, який налаштовується та працює швидше за традиційний, що значно підвищує ефективність роботи додатків і зменшує навантаження на браузер.

– Використання React спрощує написання скриптів, що робить технічне обслуговування легшим і збільшує ефективність розробки завдяки більш швидкому рендерингу.

– Однією з найбільших переваг React є наявність інструментів для створення мобільних додатків як для iOS, так і для Android.

– React базується на компонентній архітектурі, що дозволяє писати модульний код та повторно використовувати компоненти в різних частинах програми.

– React спрощує обмін та оновлення даних між компонентами, що робить їх взаємодію зручнішою і ефективнішою.

– За допомогою React можна створювати інтерфейси, які миттєво реагують на дії користувачів або зміни даних без необхідності перезавантаження сторінки.

– Підтримка серверного рендерингу в React покращує результати SEO, зменшує час завантаження сторінок і оптимізує загальну продуктивність.

– Велика спільнота розробників активно підтримує React, що сприяє його постійному розвитку і вдосконаленню.

React продовжує розвиватися та зростати завдяки своїй активній спільноті та численним інструментам для розробників. Останнім часом особливо популярним стало використання React для створення серверних додатків, оскільки бібліотека підтримує різноманітні оптимізації для рендерингу сторінок на сервері (SSR). Це дозволяє зменшити час першого рендеру та покращити результативність SEO для веб-ресурсів.

Крім того, важливою перевагою є вбудовані інструменти для тестування, такі як Jest і React Testing Library, які забезпечують зручне середовище для автоматизованих тестів. Вони дозволяють легко перевіряти компоненти на різних етапах їхнього життєвого циклу, що сприяє покращенню якості коду та зниженню ймовірності помилок.

Використання React дозволяє також інтегрувати інші популярні технології та інструменти, наприклад, Redux для управління станом програми. Завдяки цьому можна ефективно обробляти великі обсяги даних.

Redux дає можливість зберігати весь стан програми в одному місці, що спрощує процес розробки та тестування, а також дозволяє ефективно працювати з великою кількістю компонентів, які взаємодіють між собою через централізований стан. Redux також підтримує асинхронні операції за допомогою таких розширень, як redux-thunk або redux-saga, що дозволяє ефективно працювати з асинхронними запитами, обробляти побічні ефекти та інші складні сценарії.

Styled Components – це популярна бібліотека для стилізації компонентів у React, що дозволяє використовувати CSS безпосередньо в JavaScript. Завдяки Styled Components можна писати стилі, як частину компонента, що дозволяє значно спростити управління стилями в складних додатках. Ця бібліотека використовує технологію CSS-in-JS, що дозволяє компонуватизувати стилі разом із логікою компонентів, забезпечуючи кращу організацію та масштабованість коду.

Material-UI – популярна бібліотека компонентів для React, яка реалізує дизайн-систему Material Design від Google. Вона пропонує набір готових

компонентів, які можна використовувати для створення інтерфейсів користувача, що відповідають сучасним стандартам дизайну. MUI включає такі елементи, як кнопки, форми, таблиці, модальні вікна, а також механізми для управління темами та стилями.

Однією з основних переваг MUI є зручність використання та кастомізація компонентів. Всі компоненти є адаптивними, що дозволяє зручно працювати з різними розмірами екранів. Бібліотека має вбудовану підтримку зміни теми, що дозволяє швидко змінювати зовнішній вигляд додатку, зберігаючи єдину стилістичну концепцію.

React Router – це бібліотека для управління маршрутизацією в додатках на React. Вона дозволяє створювати односторінкові додатки (SPA), де зміна URL відбувається без перезавантаження сторінки, а контент змінюється динамічно в залежності від маршруту. React Router підтримує різні типи маршрутизації, як на клієнтському боці (client-side routing), так і на серверному боці (server-side routing), що робить його універсальним інструментом для різних типів додатків.

Основні можливості React Router включають:

1. Маршрутизація на основі URL. Це дозволяє відображати різні компоненти на різних маршрутах (шляхах) в URL. Наприклад, компоненти для головної сторінки, сторінки профілю, сторінки налаштувань можна відображати на різних шляхах.

2. Вкладені маршрути. React Router підтримує вкладені маршрути, що дозволяє створювати ієрархію сторінок з підсторінками.

3. Динамічні маршрути. Ви можете передавати параметри в URL і отримувати їх у компонентах, що дає змогу створювати динамічні сторінки, наприклад, для профілів користувачів або продуктів.

4. Перехід між сторінками без перезавантаження. React Router забезпечує переходи між сторінками без перезавантаження всього додатка.

5. Захист маршрутів. React Router дозволяє реалізувати захищені маршрути, де доступ до певних сторінок або компонентів може бути обмежений (наприклад, для користувачів, які не авторизовані).

React дозволяє ефективно реалізовувати архітектуру Single-Page Application (SPA), забезпечуючи інтерактивний і динамічний користувацький досвід. В основі SPA лежить ідея того, що вся веб-сторінка завантажується лише один раз, а подальші зміни в інтерфейсі відбуваються без необхідності перезавантаження сторінки. React, завдяки своїй компонентній архітектурі і можливості динамічно оновлювати лише необхідні частини інтерфейсу, є одним із основних інструментів для реалізації SPA [2].

У React для досягнення цієї мети використовуються такі концепції, як virtual DOM, що дозволяє оптимізувати процес оновлення інтерфейсу. Кожного разу, коли змінюється стан компонента, React порівнює віртуальний DOM з реальним і вносить тільки ті зміни, які необхідні. Це дозволяє значно зменшити навантаження на браузер, підвищуючи продуктивність додатка, що є критичним для SPA, де швидкість реакції на дії користувача має велике значення.

Таким чином, використання React для розробки інформаційної системи для управління відпустками є обґрунтованим завдяки високій ефективності, масштабованості та зручності інтеграції компонентів. React дозволяє реалізувати архітектуру Single-Page Application (SPA), що забезпечує безперервний користувацький досвід без необхідності перезавантаження сторінок. Завдяки цьому, система зберігає високу швидкість роботи, а зміни в інтерфейсі відбуваються миттєво, без переривань у взаємодії з користувачем.

Бібліотеки, такі як Material-UI, Styled Components та Redux, сприяють спрощенню створення адаптивних інтерфейсів, динамічного стилювання та централізованого керування станом, що дозволяє ефективно обробляти дані про відпустки. Це забезпечує зручність користування, швидкість рендерингу та легкість масштабування системи, що є важливим для складних веб-додатків з інтерактивними функціями та динамічними оновленнями.

React та його екосистема бібліотек дозволяють створити ефективну та масштабовану інформаційну систему для управління відпустками, забезпечуючи швидкий рендеринг, адаптивний інтерфейс, зручне керування даними та безперервний користувацький досвід у рамках SPA.

2.4 Node.js як платформа для серверної розробки

Node.js — це технологія для серверного програмування, побудована на основі JavaScript. Вона надає розробникам можливість використовувати JavaScript не тільки для клієнтської, але й для серверної частини додатків, що відкриває нові можливості для створення масштабованих та ефективних веб-систем. Node.js працює на двигуні V8, який був розроблений компанією Google для браузера Chrome [17]. Це забезпечує високу швидкість виконання JavaScript-коду, що робить Node.js одним з найбільш ефективних інструментів для серверної розробки.

Основною перевагою Node.js є його асинхронний, подієво-орієнтований підхід до обробки запитів. Це означає, що Node.js не блокує виконання програми під час очікування завершення вхідно-вихідних операцій, таких як звернення до бази даних або файлової системи. Замість цього, коли відбувається операція вводу/виводу, Node.js передає її в чергу, а основний потік продовжує виконувати інші операції, що дозволяє досягти більшої продуктивності при обробці великої кількості одночасних запитів.

Node.js використовує однопотокову модель обробки запитів, де всі операції виконуються в одному потоці за допомогою неблокуючих асинхронних функцій. Це значно зменшує накладні витрати, порівняно з традиційними багатопоточними системами, де кожен запит обробляється окремим потоком. Завдяки такій моделі Node.js здатний обробляти тисячі одночасних запитів із мінімальними затримками, що робить його ідеальним для реального часу, таких як чати, ігри, або сервісів із великим навантаженням на мережу.

Node.js використовує систему модулів, що дозволяє ефективно організовувати код і повторно використовувати вже створені функціональні блоки. Це забезпечує гнучкість і масштабованість додатків, дозволяючи розробникам включати лише необхідні компоненти, що зменшує накладні витрати на ресурс. Система модулів Node.js включає як вбудовані модулі (наприклад, для роботи з HTTP, файловою системою, потоками даних), так і

можливість підключення сторонніх пакетів, які можна знайти в репозиторії npm (Node Package Manager).

Npm є одним із найбільших і найактивніших сховищ пакетів, що дозволяє розробникам швидко інтегрувати додаткову функціональність у свої проекти. Завдяки npm, розробники можуть використовувати вже готові рішення для таких завдань, як управління базами даних, аутентифікація, робота з API, тестування тощо. Це дозволяє значно зменшити час на розробку і зосередитись на специфічних аспектах проекту.

Node.js також підтримує різноманітні протоколи та технології, що дає змогу створювати різноманітні типи серверних додатків. Вони можуть бути як простими веб-серверами для обробки HTTP-запитів, так і складними сервісами для обробки потокових даних або взаємодії з реальним часом, як у випадку з WebSocket. Завдяки цьому Node.js є популярним вибором для створення RESTful API, веб-додатків, чатових додатків, а також для роботи з мікросервісною архітектурою.

Ще однією важливою особливістю Node.js є його здатність працювати на різних платформах. Node.js підтримує Windows, Linux, macOS, що дозволяє створювати кросплатформенні додатки, зберігаючи однаковий код для всіх операційних систем. Це забезпечує високий рівень сумісності та дає змогу розробникам легко розгортати додатки на різних платформах без необхідності модифікації коду.

Node.js активно використовується в сучасних розподілених системах і хмарних інфраструктурах, де важливі швидкість обробки запитів та масштабованість. Інтеграція з контейнерами, такими як Docker, та інструментами для оркестрації, такими як Kubernetes, дозволяє легко масштабувати Node.js-додатки в залежності від навантаження, що робить його популярним у середовищах з високими вимогами до доступності та відмовостійкості.

На додаток до всіх цих переваг, Node.js має велику і активну спільноту розробників, що регулярно створюють нові бібліотеки та інструменти, покращуючи екосистему. Величезна кількість документації, форумів, ресурсів та

спільнот допомагає розробникам швидко освоювати Node.js і вирішувати будь-які проблеми, що виникають у процесі розробки.

На основі вищенаведеної інформації визначено особливості та переваги платформи Node.js:

1. Висока продуктивність. Node.js працює на движку V8, що забезпечує високу швидкість виконання JavaScript-коду, що робить його ефективним для серверного програмування.

2. Асинхронність. Node.js використовує неблокуючий підхід до обробки запитів, що дозволяє обробляти велику кількість запитів одночасно без затримок.

3. Однопотокова модель. Завдяки однопотоковій архітектурі Node.js знижує накладні витрати на керування потоками, що дозволяє обробляти тисячі одночасних запитів з мінімальними ресурсами.

4. Модульність та повторне використання коду. Система модулів Node.js дозволяє організовувати код та використовувати вже готові рішення з npm, що спрощує розробку.

5. Велика екосистема та підтримка npm. Репозиторій npm надає доступ до тисяч бібліотек і пакетів, що дозволяє швидко інтегрувати додаткові функціональності у проекти.

6. Кросплатформність. Node.js підтримує різні операційні системи (Windows, Linux, macOS), що дозволяє створювати кросплатформенні додатки.

7. Широке використання в мікросервісах і хмарних системах. Node.js популярний у розподілених і хмарних інфраструктурах завдяки своїй здатності швидко обробляти запити та масштабуватись.

8. Інтеграція з контейнерами і оркестрацією. Легко інтегрується з Docker та Kubernetes, що дозволяє масштабувати додатки в залежності від навантаження.

У результаті, Node.js став одним із найпопулярніших інструментів для серверної розробки завдяки своїй високій продуктивності, асинхронній природі, великій кількості інструментів і пакетів, а також гнучкості для створення різноманітних типів додатків.

2.5 Висновки до розділу

У даному розділі здійснено аналіз технологічних засобів MERN (MongoDB, Express.js, React, Node.js) для розробки інформаційної системи для управління відпустками. Вибір даних технологій для реалізації зазначеного програмного забезпечення є технічно обґрунтованим, оскільки MERN забезпечує інтеграцію усіх компонентів на основі JavaScript, що значно полегшує процес розробки, тестування та подальшої підтримки.

Використання документно-орієнтованої бази даних MongoDB забезпечує високу гнучкість при зберіганні та обробці даних. Express.js, як серверний фреймворк для Node.js, оптимізує процес розробки серверної частини, забезпечуючи ефективну маршрутизацію, обробку HTTP-запитів та взаємодію з базою даних. Компонентна архітектура React надає можливість створення динамічних, масштабованих і високопродуктивних інтерфейсів користувача. Використання Node.js, як основи серверної частини, забезпечує асинхронну обробку запитів, що дозволяє ефективно працювати з великими навантаженнями, мінімізуючи час затримки та оптимізуючи продуктивність системи.

Таким чином, застосування стеку технологій MERN для розробки інформаційної системи для управління відпустками є ефективним рішенням, що дозволяє забезпечити високу масштабованість, продуктивність і гнучкість системи.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для проектування і розробки інформаційної системи управління відпусками необхідно визначити її функціональні та нефункціональні вимоги. Розглянемо функціональні вимоги із врахуванням ролі користувача в системі – адміністратор та працівник (admin та user).

Функціональні вимоги системи для користувача із роллю адміністратора:

– Реєстрація працівників;

– Імпорт працівників з ERP-системи компанії за допомогою csv файла;

– Формування політик відсутностей;

– Перегляд звітності відпусток з можливістю фільтрації за обраним типом відсутностей, відділу та періоду;

– Перегляд нових запитів на відстуність з можливістю підтвердження або відхилення;

– Перегляд графіків відсутностей сформованих на основі запитів працівників.

Функціональні вимоги системи для користувача із роллю працівника:

– Перегляд залишку доступних вихідних днів різних типів;

– Створення запитів на відсутність;

– Перегляд історії власних запиті із відстеженням їхнього статусу;

– Перегляд інформації про заплановану відсутність колег у календарі;

До нефункціональних вимог системи відноситься:

– Адаптивність. Система повинна мати адаптивний інтерфейс, який забезпечує коректне відображення та зручність використання на різних пристроях;

– Кросбраузерність. Інтерфейс системи має коректно працювати у популярних браузерах (Chrome, Firefox, Safari);

– Безпека. Система повинна мати високий рівень безпеки, що включає шифрування даних захист паролів (хешування).

Спрощений алгоритм роботи інформаційної системи для управління відпустками наведено на рисунку 3.1, де детально зображено основні етапи процесів, що відбуваються в системі, починаючи від реєстрації користувачів і створення запитів на відсутність до їх затвердження та формування звітності.

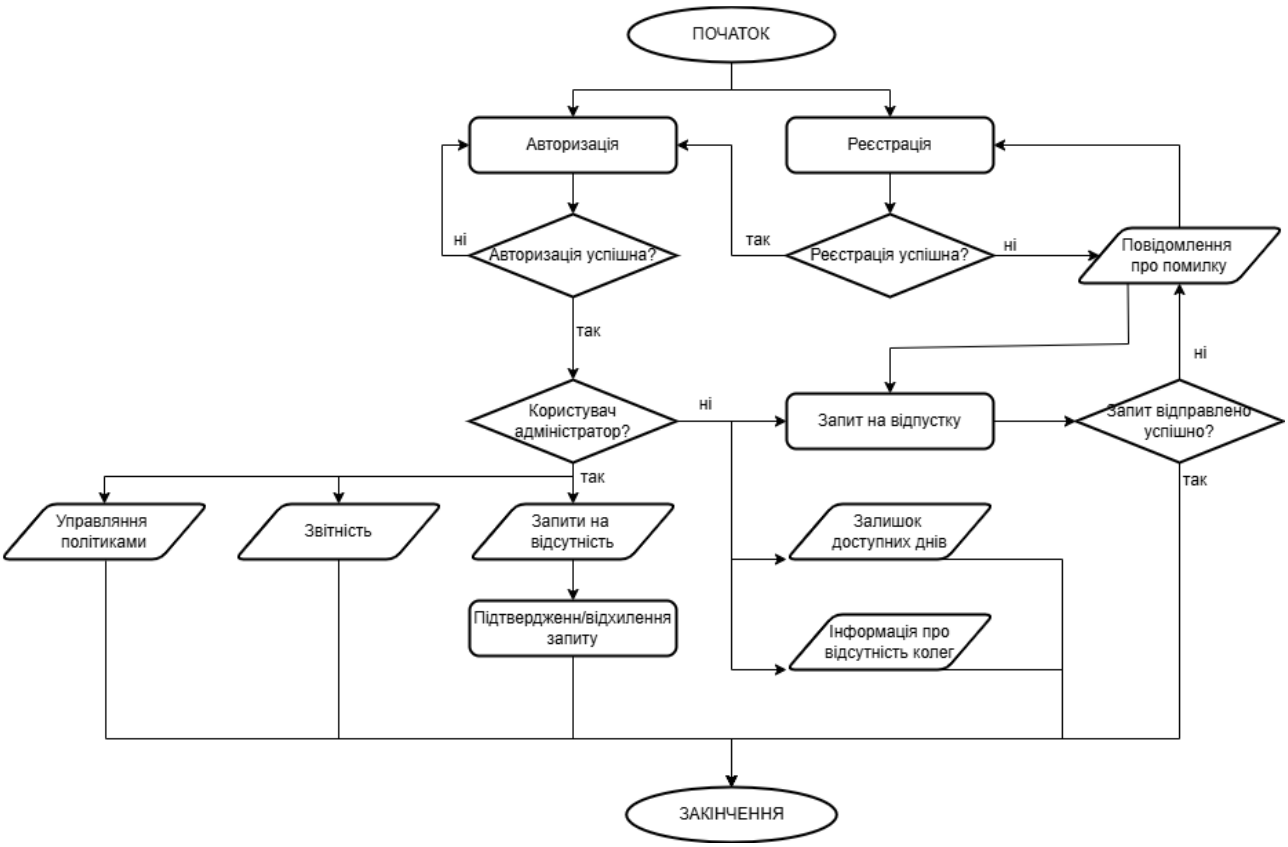

Рисунок 3.1 – Блок-схема алгоритму роботи системи

Використання UML-діаграм є важливим етапом проектування, що забезпечує візуалізацію роботи системи, а також слугує основою для подальшого впровадження та тестування.

UML-діаграми прецедентів, діяльності та класів наведено у додатку Г, де представлені основні аспекти проектування системи, зокрема взаємодії користувачів із системою, основні сценарії використання, процеси виконання дій, а також структура класів, які відображають ключові елементи системи та їхні взаємозв’язки.

3.1 Проектування бази даних

Важливим етапом розробки інформаційної системи для управління відпустками є проектування бази даних. У рамках даного підрозділу розроблено модель бази даних, описано її сутності та атрибути, а також обгрутновано зв’язки між сутностями.

Сутність – це основний концепт в моделюванні баз даних, який представляє об'єкт, про який система зберігає інформацію. Сутності описуються через набір атрибутів, які визначають їхні властивості.

Атрибут – це характеристика сутності, яка описує її окремі аспекти. Складовими атрибуту є його назва, тип даних та обмеження, наприклад вказівка на унікальність, заборона на порожнє значення чи позначення атрибуту як ключ сутності.

Зв'язки у базі даних визначають, як сутності пов'язані між собою. Вони встановлюються через первинні і зовнішні ключі. Існує три види зв’язків:

– Один до одного (1:1) – кожен запис однієї таблиці відповідає одному запису іншої.

– Один до багатьох (1:N) – один запис першої таблиці відповідає кільком записам другої.

– Багато до багатьох (M:N) – записи двох сутностей можуть співвідноситися багатьма способами, реалізується через проміжну таблицю.

Для проектування моделі бази даних розробленої системи визначено наступні сутності: користувач (user), токен (token), працівник (employee), роль (role), відділ (department), запит на відсутність (leaveRequest), тип відсутності (leaveType) та політика (policy), які забезпечують зберігання й управління ключовими даними системи. Розглянемо призначення та атрибути кожної сутності.

Сутність «користувач» (user) призначення для збереження реєстраційних даних працівника. Атрибути даної сутності наведено у таблиці 3.1.

Таблиця 3.1 – Опис атрибутів сутності «користувач»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
firstName	String	заборона на пусте значення	Ім’я співробітника
lastName	String	заборона на пусте значення	Прізвище співробітника
email	String	заборона на пусте значення	Електронна пошта
password	String	заборона на пусте значення	Захешований пароль
isActivated	Boolean	заборона на пусте значення	Підтвердження електронної пошти
activationLink	String	заборона на пусте значення	Посилання для активації електронної пошти

Сутність «токен» призначена для зберігання токенів оновлення для кожного користувача системи. Атрибути даної сутності наведено у таблиці 3.2.

Таблиця 3.2 – Опис атрибутів сутності «токен»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
user	ObjectId	Зовнішній ключ, заборона на пусте значення	Посилання на користувача-власника
refreshToken	String	заборона на пусте значення	Токен оновлення

Сутність «працівник» (employee) призначена для зберігання службової інформації про працівника орагізації, наприклад, приналежність до певного відділу; безпосереднього керівника чи менеджера, за яким закріплений працівник; історію запитів на відсутність; роль у системі; дату початку роботи; дату звільнення за необхідності та політику закріплену за працівником. Атрибути даної сутності наведено у таблиці 3.3.

Таблиця 3.3 – Опис атрибутів сутності «працівник»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
role	ObjectId	Зовнішній ключ	Роль у системі
policy	ObjectId	Зовнішній ключ	Політика
user	ObjectId	Зовнішній ключ	Реєстраційні дані
manager	ObjectId	Зовнішній ключ	Безпосередній керівник
history	ObjectId	відсутні	Історія запитів
startDate	Date	заборона на пусте значення	Дата найму
endDate	Date	відсутні	Дата звільнення
department	ObjectId	Зовнішній ключ	Відділ роботи

Сутність «роль» (role) призначення для зберігання інормації про можливі ролі користувачів у системі. Атрибути даної сутності наведено у таблиці 3.4.

Таблиця 3.4 – Опис атрибутів сутності «роль»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
name	String	заборона на пусте значення	Назва ролі

Сутність «відділ» (department) призначена для зберігання інформації про відділи в організації. Атрибути даної сутності наведено у таблиці 3.5.

Таблиця 3.5 – Опис атрибутів сутності «відділ»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
name	String	заборона на пусте значення	Назва відділу
lead	ObjectId	Зовнішній ключ	Керівник відділу

Сутність «запит на відсутність» призначена для зберігання інформації про створені запити кожним працівником. Атрибути даної сутності наведено у таблиці 3.6.

Таблиця 3.6 – Опис атрибутів сутності «запит на відсутність»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
leaveType	ObjectId	Зовнішній ключ	Тип відсутності
employee	ObjectId	Зовнішній ключ	Працівники
status	String	заборона на пусте значення	Статус запиту
startDate	Date	заборона на пусте значення	Дата початку
endDate	Date	заборона на пусте значення	Дата закінчення

Сутність «тип відсутності» призначена для зберігання інформації про можливі типи вісутностей у компанії, наприклад, відпустка, лікарняний, компенсаційний вихідний і т.д. Атрибути даної сутності наведено у таблиці 3.7.

Таблиця 3.7 – Опис атрибутів сутності «тип відсутності»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
name	String	заборона на пусте значення	Тип відсутності
description	String	відсутні	Опис
amount	Integer	заборона на пусте значення	Кількість днів

Сутність «політика» (policy) призначена для зберігання набору правил до обліку відсутностей та доступні до використання на поточний момент типи відсутностей. Організація може створити різні політики для різних відділів та працівників за потреби. Атрибути даної сутності наведено у таблиці 3.8.

Таблиця 3.8 – Опис атрибутів сутності «політика»

Назва	Тип даних	Обмеження	Призначення
id	ObjectId	Первинний ключ, заборона на пусте значення	Унікальний ідентифікатор запису
name	String	заборона на пусте значення	Назва політики
description	String	відсутні	Опис
leaveType	ObjectId	Зовнішній ключ	Доступні типи вихідних
owner	ObjectId	Зовнішній ключ	Власник

Для побудови моделі бази даних необхідно визначити зв’язки між сутностями та їхні типи. Розглянемо типи зв’язків у проектованій базі даних для інформаційної системи управління відпустками.

Сутність «токен» має зв’язок із сутністю «користувач» із типом «один до одного», тобто один користувач може мати лише один токен на оновлення токену доступу. На рисунку 3.2 представлено ER-діаграму сутностей «Токен – Користувач».

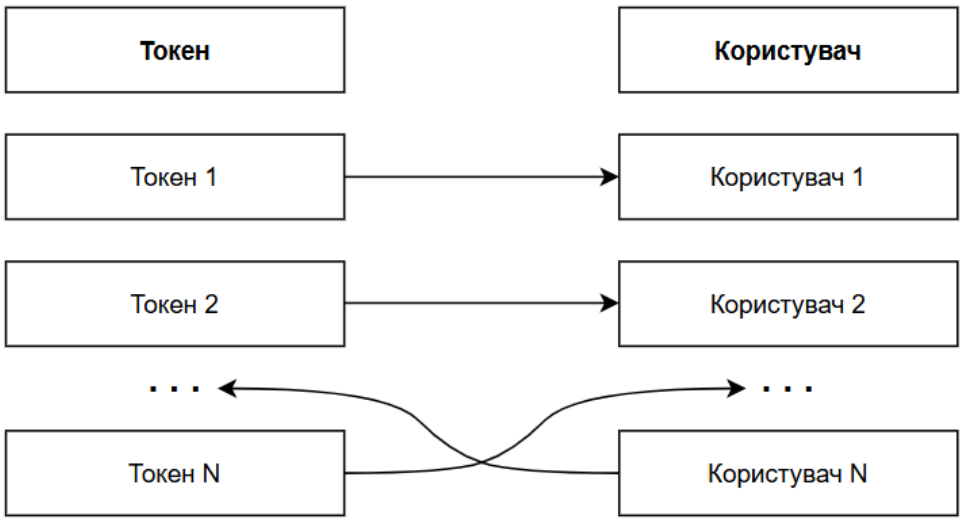

Рисунок 3.2 – ER-діаграма сутностей «Токен – Користувач»

Сутність «користувач» має зв’язок із стуністю «працівник» із типом «один до одного», так як за одним користувачем закріплений лише один працівник. На рисунку 3.3 представлено ER-діаграму сутностей «Користувач – Працівник».

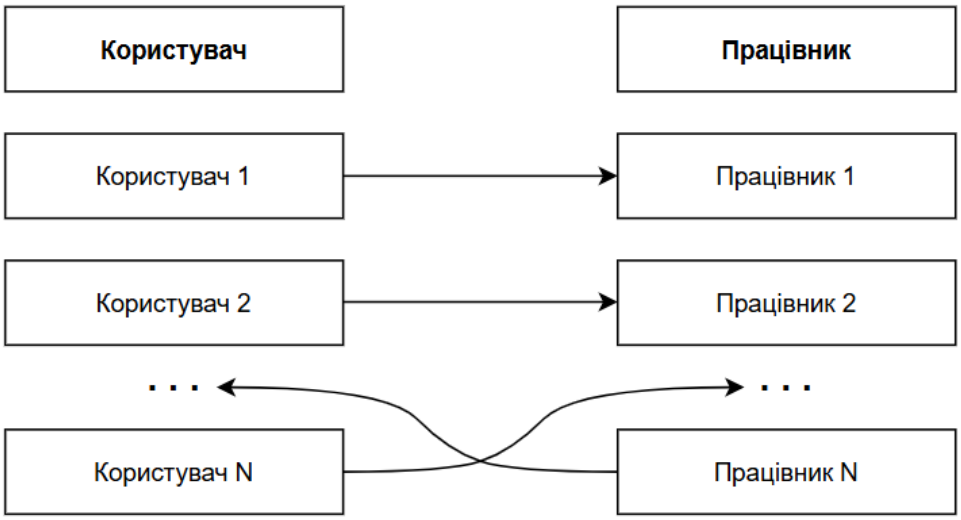

Рисунок 3.3 – ER-діаграма сутностей «Користувач – Працівник»

Сутність «роль» має зв’язок із сутністю «працівник» із типом «один до одного», тобто одному працівнику призначено лише одну роль, яка відповідатиме за права та розмежування доступів до модулів системи. На рисунку 3.4 представлено ER-діаграму сутностей «Працівник – Роль».

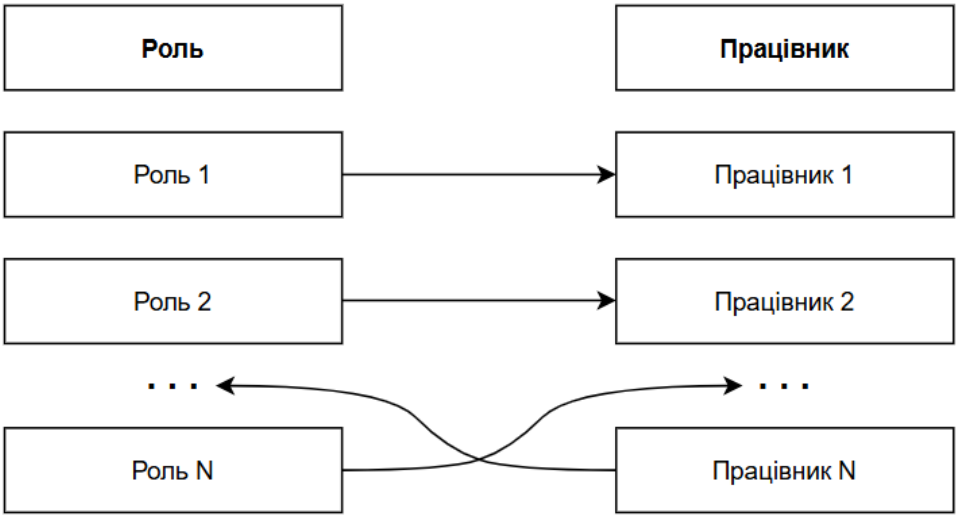

Рисунок 3.4 – ER-діаграма сутностей «Працівник – Роль»

Сутність «відділ» має зв’язок із сутністю «працівник» із типом «один до багатьох», так як за одним відділом може бути закріплено багато працівників. На рисунку 3.5 представлено ER-діаграму сутностей «Відділ – Працівник».

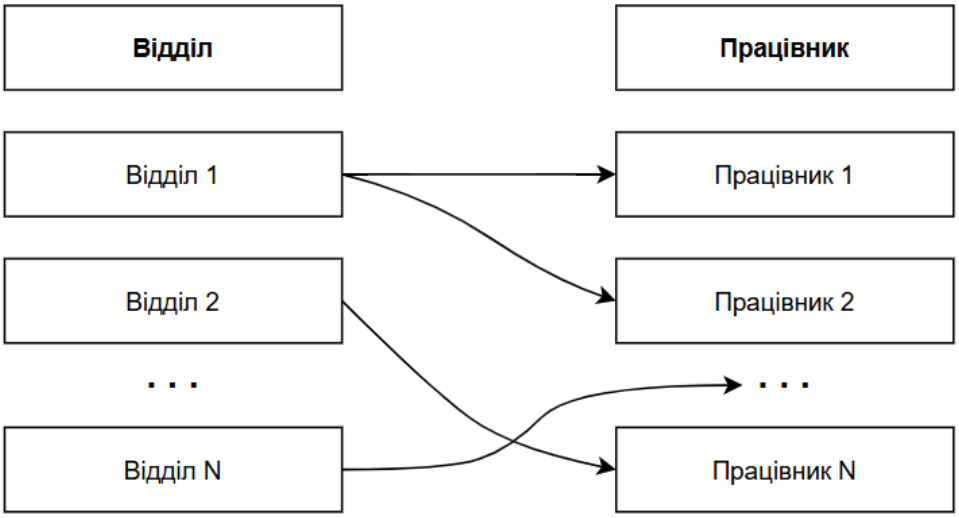

Рисунок 3.5 – ER-діаграма сутностей «Відділ – Працівник»

Сутність «політика» має зв’язок із сутністю «працівник» із типом «один до одного», тобто за одним працівником закріплена лише одна політика відсутностей. На рисунку 3.6 представлено ER-діаграму сутностей «Політика – Працівник».

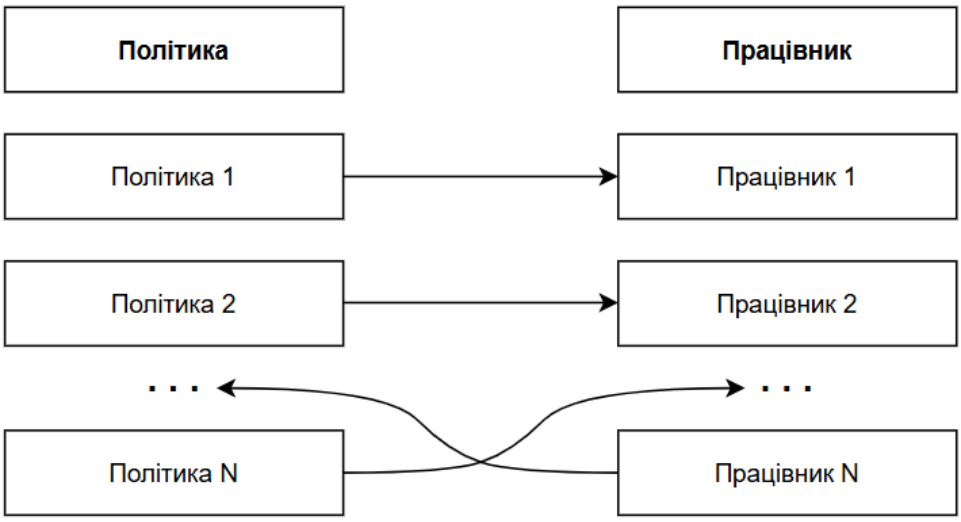

Рисунок 3.6 – ER-діаграма сутностей «Політика – Працівник»

Сутність «політика» має зв’язок із сутністю «тип відсутності» із типом «один до багатьох», так як одна політика може містити багато типів відсутностей. На рисунку 3.7 представлено ER-діаграму сутностей «Політика – Тип відсутності».

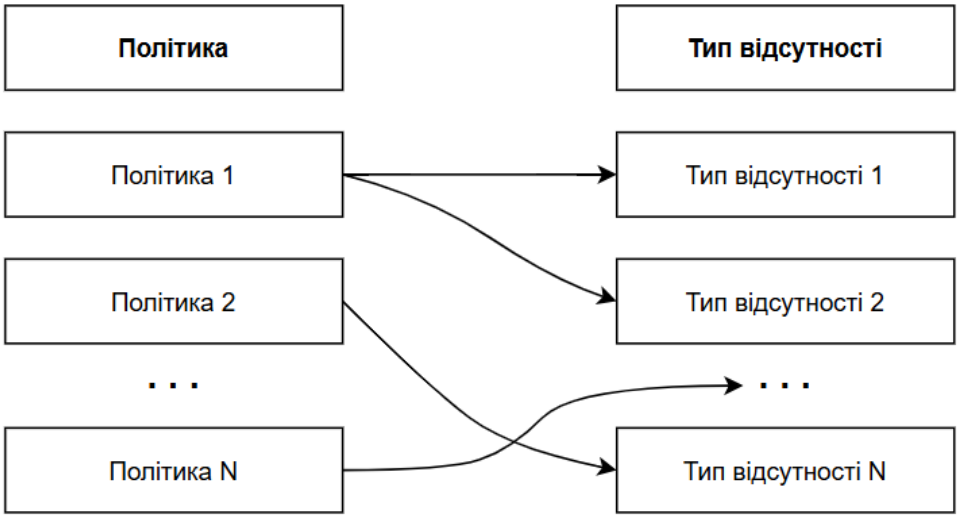

Рисунок 3.7 – ER-діаграма сутностей «Політика – Тип відсутності»

Сутність «Запит на відсутність» має зв’язок із сутністю «Тип відсутності» із зв’язком «один до одного», так як у одному запиті може міститися лише один тип відсутності. На рисунку 3.8 представлено ER-діаграму сутностей «Запит на відсутність – Тип відсутності».

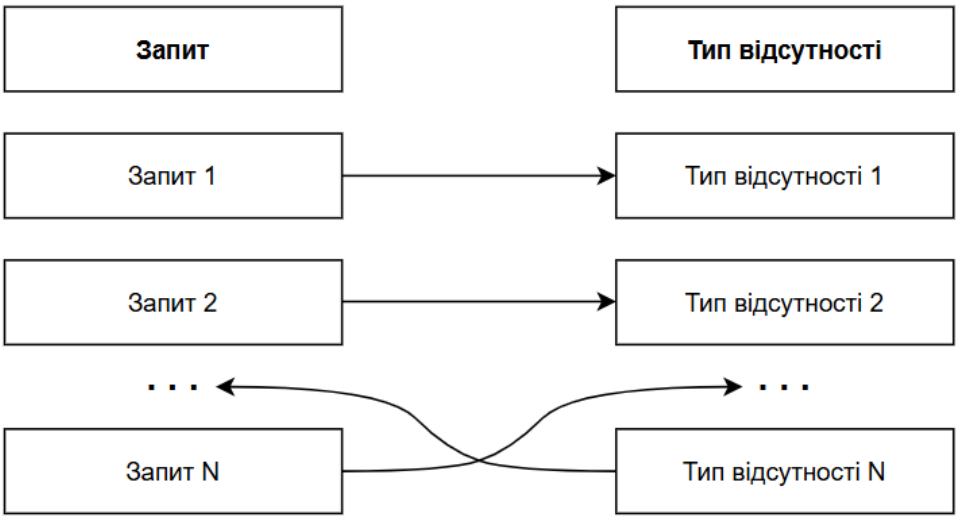

Рисунок 3.8 – ER-діаграма сутностей «Запит – Тип відсутності»

Сутність «Працівник» має зв’язок із сутністю «Запит на відсутність» із типом «один до багатьох», так як один працівник може відправляти кілька запитів на відсутність. На рисунку 3.9 представлено ER-діаграму сутностей «Працівник – Запит на відсутність».

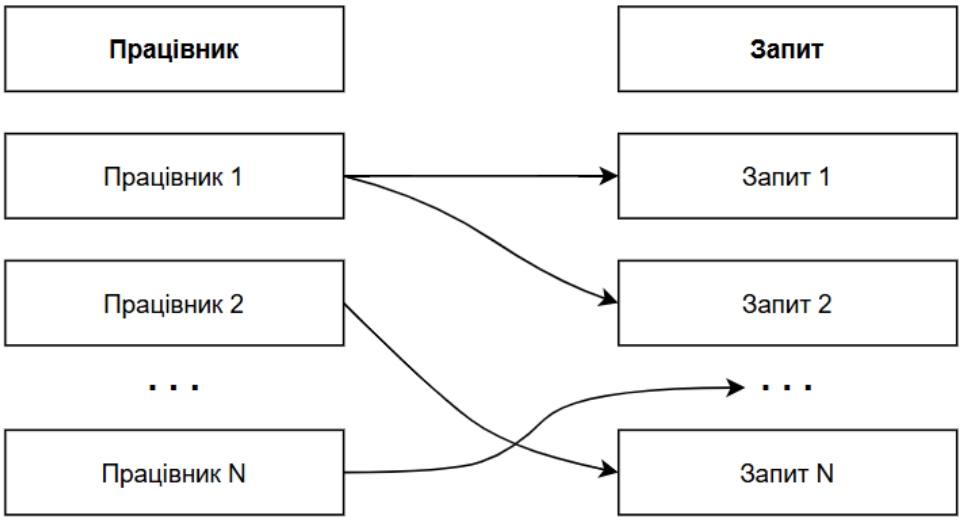

Рисунок 3.9 – ER-діаграма сутностей «Працівник – Запит на відсутність»

На основі створених ER-діаграм було розроблено одну результуючу діаграму, що відображає взаємозв'язки між сутностями. Результуюча ER-діаграма представлена на рисунку 3.10.

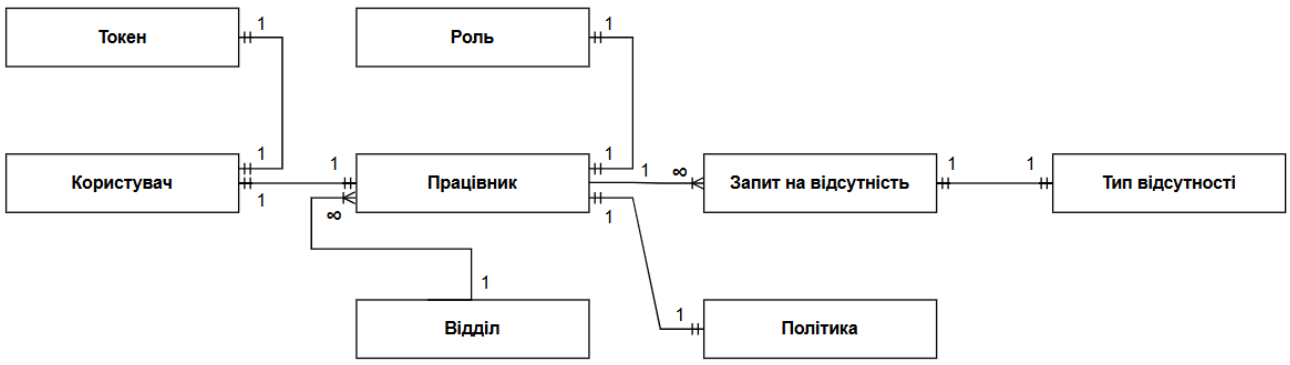

Рисунок 3.10 – Результуюча ER-діаграма системи

На основі результуючої ER-діаграми та визначених атрибутів було спроектовано модель бази даних, яка відповідає вимогам системи та забезпечує ефективне управління даними. На рисунку 3.11 представлено спроектовану базу даних для інформаційної системи управління відпустками, що включає всі сутності, їх атрибути та зв'язки між ними, необхідні для подальшої розробки системи.

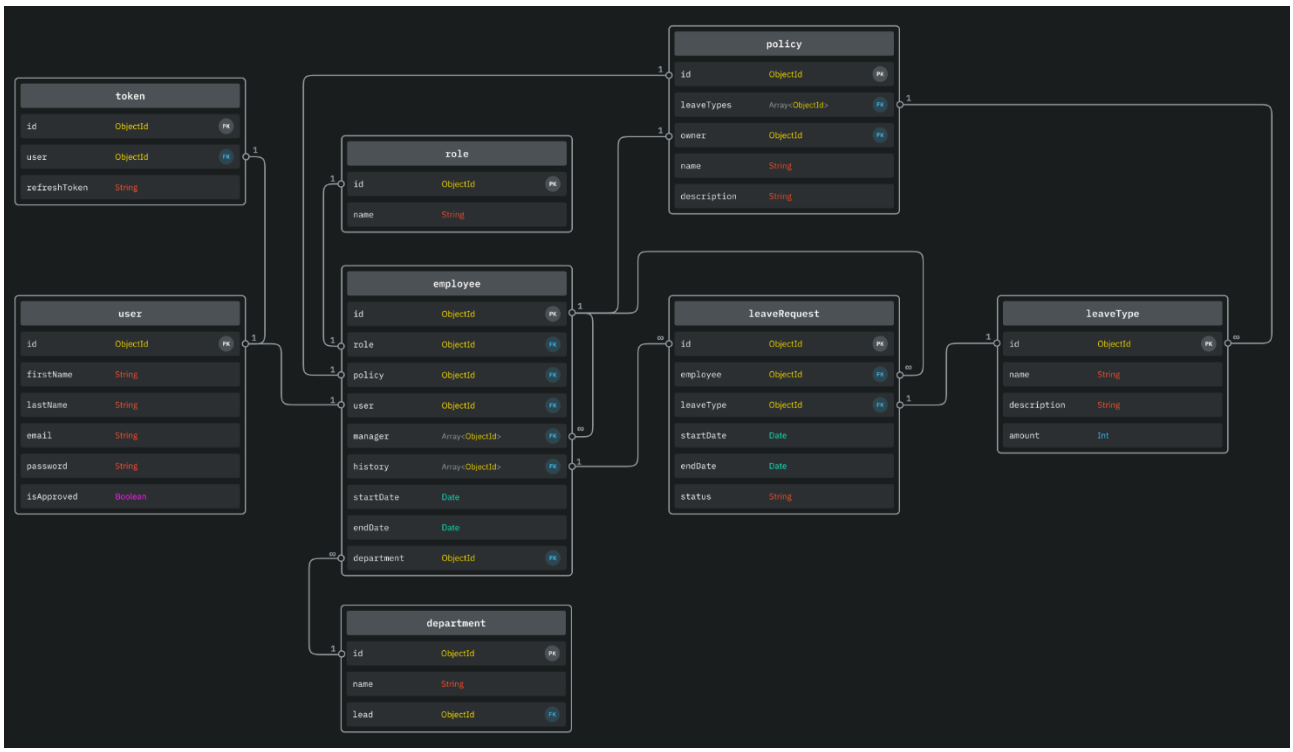

Рисунок 3.11 – Модель бази даних

Таким чином, спроектована бази даних для інформаційної системи управління відпустками дозволяє забезпечити ефективне зберігання та обробку даних, необхідних для функціонування системи.

Створена модель відповідає вимогам користувачів та організаційних процесів, забезпечуючи ефективне управління даними щодо відсутностей працівників, що в свою чергу підвищує ефективність роботи всієї організації.

3.2 Розробка серверної частини системи

Розробка серверної частини системи є важливим етапом, що забезпечує обробку запитів користувачів, управління даними та взаємодію з базою даних. Цей процес охоплює створення програмної логіки, налаштування API та впровадження механізмів безпеки для стабільної та ефективної роботи системи.

Розглянемо структуру серверної частини проекту інформаційної системи управління віпустками, предствлену на рисунку 3.12.

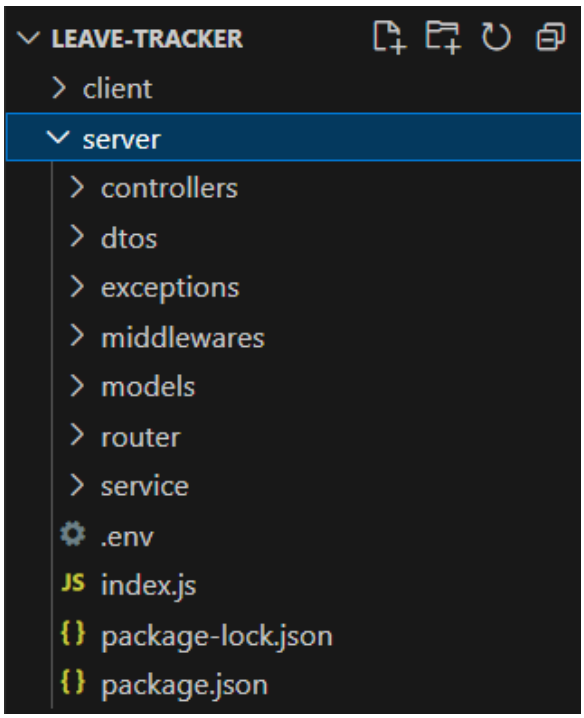

Рисунок 3.12 – Структура серверної частини проекту

Опис директорій серверної частини проекту:

1. *controllers* – відповідає за обробку HTTP-запитів і передачу відповідей клієнту, вмістом даної директорії є функції, які взаємодіють із сервісами та відправляють необхідні дані;

2. *dtos (Data Transfer Objects)* – призначена для визначення форматів даних, які передаються між клієнтом і сервером, допомагає уніфікувати структуру даних і спрощує валідацію;

3. *exceptions* – містить обробники або ж класи для роботи з винятковими ситуаціями, такими як помилки сервера;

4. *middlewares* – зберігає посередників, які виконуються між отриманням запиту сервером і його обробкою (наприклад, авторизація, логування, обробка CORS);

5. *models* – містить визначення моделей даних і схеми для роботи з базою даних, тобіж опис сутностей та їхні атрибути.

6. *Router* – відповідає за маршрутизацію, тобто визначає, який контролер викликати при зверненні за певною URL-адресою.

7. *Service* – містить бізнес-логіку програми. Сервіси взаємодіють із моделями, виконують необхідні операції з даними та повертають результат контролерам.

У файлі *.env* містяться змінні оточення, такі як ключі доступу, URL бази даних чи інші чутливі дані.

Файл *index.js* є головним файлом серверного додатка, який відповідає за ініціалізацію застосунку, підключення до бази даних та налаштування серверу.

У файлі *package.json* міститься метаінформація, перелік залежностей, скрипти для запуску застосунку та інші системні параметри.

Розглянемо принцип взаємодії елементів структури серверної частини проекту на прикладі розробленого модуля авторизації. Для забезпечення високого рівня надійності та безпеки системи обрано розробку модулю авторизації з використанням JWT.

JWT використовується для авторизації, забезпечуючи безпечну передачу даних між клієнтом і сервером. Після успішної автентифікації сервер генерує токен, який містить закодовані дані та підписується секретним ключем. Токен передається клієнту, зберігається у cookies та надсилається з кожним запитом у заголовку Authorization. Сервер верифікує токен, перевіряючи його підпис і термін дії, для надання доступу до захищених ресурсів.

На рисунку 3.12 представлено фрагмент контроллера користувача, що відповідає за реєстрацію.

```
const userService = require('../service/user-service');
const {validationResult} = require('express-validator');
const ApiError = require('../exceptions/api-error');

class UserController {
    async registration(req, res, next) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                return next(ApiError.BadRequest('Validation error', errors.array()))
            }
            const {email, password} = req.body;
            const userData = await userService.registration(email, password);
            res.cookie('refreshToken', userData.refreshToken, {maxAge: 30 * 24 * 60 * 60 * 1000, httpOnly: true})
            return res.json(userData);
        } catch (e) {
            next(e);
        }
    }
```

Рисунок 3.12 – Фрагмент котроллера користувача

Як бачимо, дана функція використовує обробник помилок, фрагмент коду якого наведено на рисунку 3.13.

```
module.exports = class ApiError extends Error {
    status;
    errors;

    constructor(status, message, errors = []) {
        super(message);
        this.status = status;
        this.errors = errors;
    }

    static UnauthorizedError() {
        return new ApiError(401, 'Unauthorized Error')
    }

    static BadRequest(message, errors = []) {
        return new ApiError(400, message, errors);
    }
}
```

Рисунок 3.13 – Фрагмент коду модулю обробки помилок

Також у контроллері користувача для реєстрації використано функцію реєстрації, реалізовану у сервісі користувача, фрагмент коду якої наведено на рисунку 3.14.

```
class UserService {
    async registration(email, password) {
        const candidate = await UserModel.findOne({email})
        if (candidate) {
            throw ApiError.BadRequest(`User with email ${email}  already exists`)
        }
        const hashPassword = await bcrypt.hash(password, 3);
        const activationLink = uuid.v4(); // v34fa-asfasf-142saf-sa-asf

        const user = await UserModel.create({email, password: hashPassword, activationLink})
        await mailService.sendActivationMail(email, `${process.env.API_URL}/api/activate/${activationLink}`);

        const userDto = new UserDto(user); // id, email, isActivated
        const tokens = tokenService.generateTokens({...userDto});
        await tokenService.saveToken(userDto.id, tokens.refreshToken);

        return {...tokens, user: userDto}
    }
```

Рисунок 3.14 – Фрагмент коду сервісу користувача

Спершу відбувається перевірка на наявність у базі даних вже зреєстрованого користувача за отриманим електронним адресом. У разі, коли даний користувач вже існує, викликається обробник помилок. Наступним кроком є хешування паролю за допомогою бібліотеки bcrypt. Після чого генерується посилання для активації електронної пошти, яке надсилається користувачеві за допомогою сервісу відправки повідомлень. Наступним етапом є генерація токенів та збереження токену оновлення для даного користувача в базі даних.

Розглянемо вміст файлу index.js, наведеного на рисунку 3.15. Ініціалізується сервер за допомогою Express, підключаються middleware для обробки JSON, cookie, CORS із вказаним клієнтським доменом, а також маршрутизація через /api і обробник помилок. Завантажуються змінні середовища з .env. У функції start встановлюється з'єднання з MongoDB через бібліотеку Mongoose, після чого сервер запускається на вказаному порту. У разі помилки з'єднання або запуску вона логується в консоль.

```
require('dotenv').config()
const express = require('express');
const cors = require('cors');
const cookieParser = require('cookie-parser')
const mongoose = require('mongoose');
const router = require('./router/index')
const errorMiddleware = require('./middlewares/error-middleware');

const PORT = process.env.PORT || 5000;
const app = express()

app.use(express.json());
app.use(cookieParser());
app.use(cors({
    credentials: true,
    origin: process.env.CLIENT_URL
}));
app.use('/api', router);
app.use(errorMiddleware);

const start = async () => {
    try {
        await mongoose.connect(process.env.DB_URL, {
            useNewUrlParser: true,
            useUnifiedTopology: true
        })
        app.listen(PORT, () => console.log(`Server started on PORT = ${PORT}`))
    } catch (e) {
        console.log(e);
    }
}

start()
```

Рисунок 3.15 – Вміст файлу index.js

Для серверної частини додатку були використані бібліотеки, що забезпечують безпеку, взаємодію з базою даних, обробку запитів, валідацію даних та відправку електронних листів:

1. Bcrypt – бібліотека для хешування паролів, яка використовується для забезпечення безпеки зберігання чутливих даних. Вона реалізує алгоритм хешування bcrypt, що додає випадкову "сіль" до пароля перед його хешуванням, щоб захистити дані від атак методом перебору та використання попередньо обчислених таблиць;

2. Mongoose – об'єктно-документний моделювальник (ODM) для роботи з MongoDB у Node.js, який спрощує взаємодію з базою даних. Він дозволяє створювати схеми для документів, забезпечує валідацію, обробку запитів і взаємодію з MongoDB у структурованому вигляді;

3. Nodemailer – бібліотека для Node.js, яка використовується для надсилання електронних листів через SMTP або інші транспортні механізми;

4. jsonwebtoken – бібліотека для Node.js, що використовується для створення, підписування та перевірки JSON Web Tokens (JWT). Вона дозволяє реалізувати механізми авторизації та автентифікації користувачів через токени.

5. express-validator – бібліотека для валідації та санітизації даних у запитах HTTP в рамках Express.js. Вона надає зручний набір методів для перевірки введених даних, таких як перевірка формату електронної пошти, мінімальна або максимальна довжина тексту, та інші умови, що дозволяють запобігти неправильним або небезпечним даним.

Таким чином, розробка серверної частини системи є важливим етапом, що забезпечує ефективну взаємодію між користувачем і базою даних, а також гарантує надійність і безпеку роботи системи. Всі елементи, від обробки HTTP-запитів до реалізації бізнес-логіки та валідації даних, інтегруються в єдину структуру, що дозволяє виконувати ключові функції інформаційної системи, такі як управління відпустками, реєстрація користувачів і захист даних. Використання сучасних бібліотек і технологій сприяє спрощенню розробки та підтримки цієї системи.

3.3 Розробка клієнтської частини системи

Розробка клієнтської частини проекту інформаційної системи для управління відпустками включає створення інтерфейсу, через який користувачі можуть подавати заявки на відпустку, переглядати їх статуси та управляти обліком відпусток. Важливим аспектом є інтеграція з серверною частиною для забезпечення ефективного обміну даними та забезпечення зручності в роботі з системою.

Розглянемо структуру клієнтської частини розроблюваної системи, наведену на рисунку 3.16.

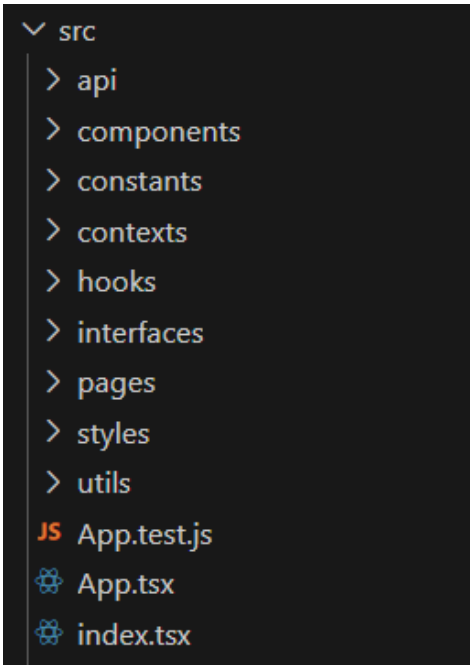

Рисунок 3.16 – Структура клієнтської частини проекту

Опис директорій клієнтської частини проекту:

1. *api* – містить функції для роботи з API, які здійснюють запити до серверної частини системи.

2. *components* – містить багаторазові компоненти інтерфейсу, що використовуються на різних сторінках.

3. *constants* – зберігає константи, такі як статичні значення елементів інтерфейсу чи розробки.

4. *contexts* – реалізує контексти для зберігання та управління станом додатку в рамках React.

5. *hooks* – містить кастомні хуки для покращення організації коду та повторного використання логіки.

6. *interfaces* – описує типи даних (інтерфейси) для TypeScript, що використовуються в додатку.

7. *pages* – містить компоненти, що відповідають за окремі сторінки в додатку.

8. *styles* – зберігає глобальні стилі для застосунку, використовуючи styled-components стилізацію.

9. *utils* – містить утиліти та допоміжні функції для спрощення коду.

Файли *App.test.js, App.tsx,* та *index.tsx* зазвичай використовуються для тестування, основного компонента додатку та для ініціалізації застосунку відповідно.

Для розробки клієнтської частини додатку було використано додаткові бібліотеки окрім React. Розглянемо основні з них:

1. *MUI (Material-UI)* – бібліотека компонентів інтерфейсу користувача для React, яка реалізує принципи Material Design. Вона надає набір готових адаптивних компонентів, що дозволяють розробникам швидко створювати естетично привабливі та адаптивні веб-додатки.

2. *react-router-dom* – бібліотека для реалізації маршрутизації в односторінкових веб-додатках на основі React. Вона дозволяє організовувати навігацію між різними компонентами через URL-адреси, забезпечуючи підтримку історії браузера, динамічних маршрутів та захищених маршрутів.

3. *styled-components* – бібліотека для стилізації компонентів в React, яка дозволяє використовувати синтаксис CSS у межах JavaScript через tagged template literals. Вона підтримує динамічну генерацію стилів та надає можливість створення стилізованих компонентів, що підвищує зручність та читаємість коду.

4. *react-hook-form* – бібліотека для ефективного управління формами в React. Вона спрощує інтеграцію валідації, обробки помилок та відправки даних з форм, знижуючи обсяг необхідного коду та покращуючи продуктивність за рахунок мінімізації перерендерів компонентів.

5. *axios* – бібліотека для виконання HTTP-запитів, яка забезпечує просту та ефективну роботу з асинхронними запитами, підтримує обробку промісів, обробку відповідей та помилок. Вона є популярним інструментом для взаємодії з RESTful API в JavaScript/TypeScript середовищах.

6. *eslint* – статичний аналізатор коду, який забезпечує дотримання стандартів кодування та виявлення помилок на етапі розробки. Бібліотека використовує правила лінтингу для автоматичної перевірки коду JavaScript/TypeScript, що дозволяє зменшити кількість помилок, покращити структуру коду та забезпечити його відповідність встановленим вимогам.

Для взаємодії з сервером у додатку виконуються HTTP-запити, для чого використовується бібліотека Axios, яка забезпечує зручний і ефективний механізм обробки запитів і отримання відповідей.

На рисунку 3.17 представлено приклад функції-запиту для отримання історії запитів.

```
import axios from 'axios';

const path = '/api/history';

export const getHistory = async (userID) => {
  try {
    const response = await axios.get(`${path}?user=${userID}`);
    return response.data;
  } catch (error) {
    console.error('Error fetching requests history', error.response ? error.response.data : error.message);
    throw error;
  }
};
```

Рисунок 3.17 – Приклад функції-запиту для отримання історії запитів

Маршрутизація у клієнтській частині веб-додатку забезпечує навігацію між різними сторінками або функціональними модулями без необхідності повного перезавантаження сторінки. Для реалізації цієї функціональності

використовується бібліотека react-router-dom, яка дозволяє організувати маршрути, управляти їхнім станом та забезпечувати зручний доступ до потрібних компонентів залежно від адреси в рядку браузера.

На рисунку 3.18 наведено фрагмент коду, що відповідає за маршрутизацію, у розробленій системі.

```
const App = () => {
  const { user, roles, isAuthChecking, loginMutation } = useUsersData();

  return (
    <UsersContext.Provider value={{ user, roles }}>
      <GlobalStyles />

      <ErrorBoundary>
        <Suspense fallback={<Spin className="page-loading-spinner-wrap" size="large" />}>
          <Switch>
            <PrivateRoute path="/" rolesForPage={ROLES_TO_PAGE.homePage} exact component={HomePage} />
            <Route path="/login" exact>
              <LoginPage isAuthChecking={isAuthChecking} login={loginMutation} />
            </Route>
            <PrivateRoute path="/report" rolesForPage={ROLES_TO_PAGE.reportPage} exact component={ReportPage} />
            <PrivateRoute path="/register" exact component={RegisterPage} />
            <Route path="/approveemail/thanks" exact component={ApprovedEmailThanksPage} />
            <Route path="*" component={NotFoundPage} />
          </Switch>
        </Suspense>
      </ErrorBoundary>
    </UsersContext.Provider>
  );
};
```

Рисунок 3.18 – Маршрутизація системи

У проекті використовується Error Boundary – механізм обробки помилок у React, який дозволяє запобігти падінню всього додатка при виникненні критичних помилок у рендерінгу, методах життєвого циклу або конструкторах дочірніх компонентів. Error Boundary працює як спеціальний компонент, що "перехоплює" помилки в межах своєї ієрархії дочірніх компонентів, відображаючи резервний інтерфейс або повідомлення про помилку. Error Boundary також дозволяє логувати деталі помилок або надсилати їх на сервер для подальшого аналізу, що сприяє швидкому виявленню та виправленню проблем. На рисунку 3.19 представлено код функції Error Boundary.

```
class ErrorBoundary extends Component {
  constructor() {
    super();
    this.state = { hasErrored: false };
  }
  static getDerivedStateFromError(error) {
    return { hasErrored: true };
  }
  componentDidCatch(error, info) {
    console.log(error);
  }
  render() {
    if (this.state.hasErrored) {
      return (
        <Result status="500" title="500" subTitle="Sorry, something went wrong."
          extra={
            <Button type="primary" href="/">
              Back Home
            </Button>
          }
        />
      );}
    return this.props.children;
  }
}
```

Рисунок 3.19 – Код функції Error Boundary

На рисунку 3.20 представлено приклад інтерфейсу системи у разі виненкнення невідловленої помилки (Error Boundary).

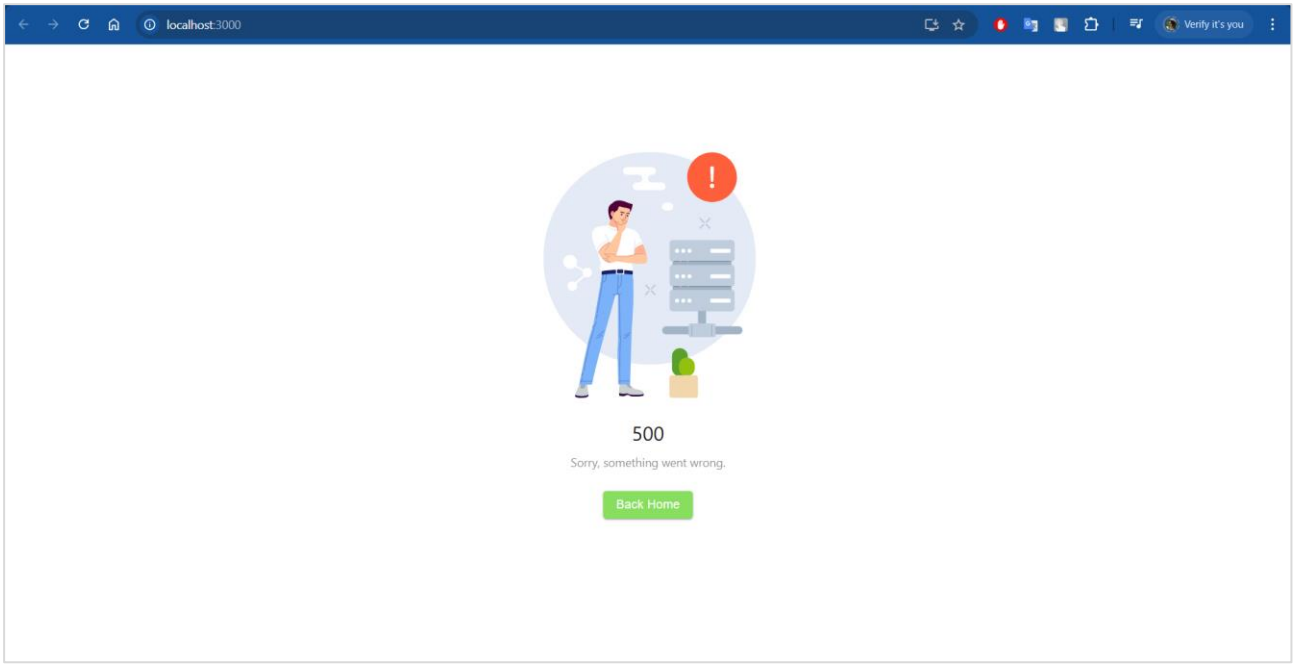

Рисунок 3.20 – Приклад інтерфейсу Error Boundary

Розглянемо приклад інтерфейсу розробленої системи для користувача з роллю «працівник», представлений на рисунку 3.21. Інтерфейс забезпечує зручний доступ до основних функцій управління відпустками, зокрема, у лівій частині екрана відображаються доступні види відпусток із зазначенням залишку днів. У центральній частині інтерфейсу знаходиться історія запитів користувача, що надає можливість відслідковувати поточний статус поданих заявок на відпустку. Функціонал відстеження статусу реалізовано за допомогою протоколу WebSocket, що забезпечує можливість оновлення інформації в реальному часі без необхідності перезавантаження сторінки.

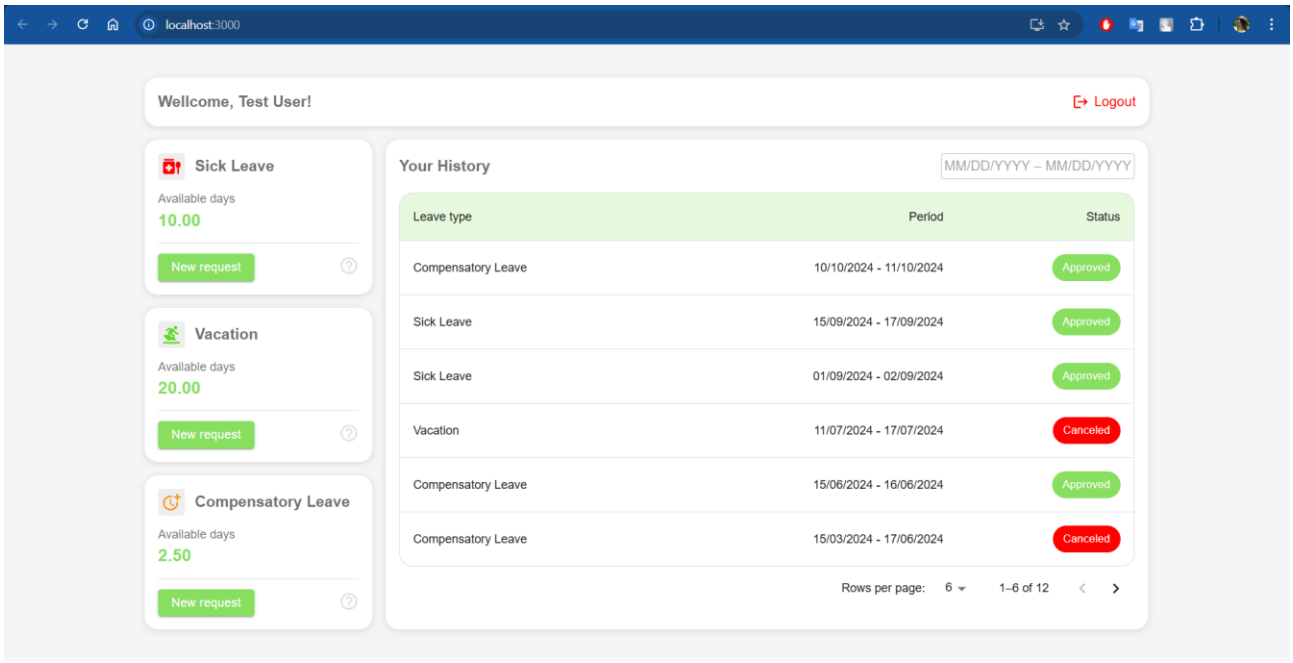

Рисунок 3.21 – Приклад головної сторінки інтерфейсу користувача

На рисунку 3.22 представлено форму відправки запиту на відсутність, забезпечує користувача інтуїтивно зрозумілим інтерфейсом для введення необхідних даних, таких як тип відпустки, бажані дати початку та завершення. Форма включає механізми валідації, які запобігають помилкам під час заповнення полів. Валідація реалізована за допомогою бібліотеки react-hook-form у поєднанні з Yup, що дозволяє налаштовувати правила та зручний зворотний зв'язок для користувача у випадку помилок.

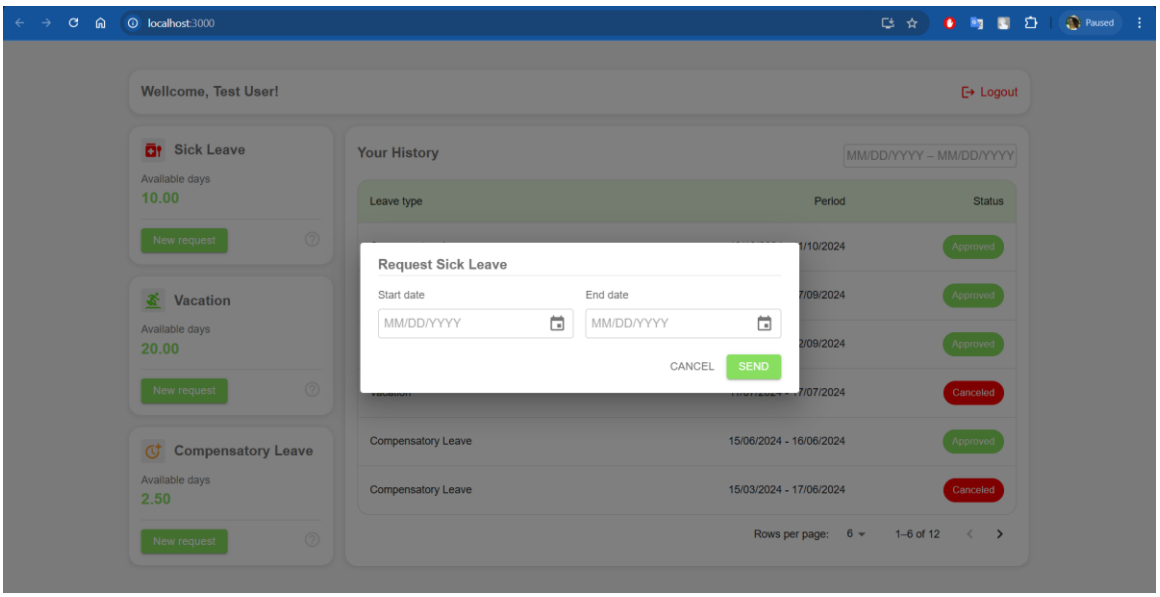

Рисунок 3.22 – Форма відправки запиту на відсутність

На рисунку 3.23 наведено приклад інтерфейсу для управління політиками компанії. Як бачимо адміністратор має можливість гнучкого створення політик для окремих відділів та працівників компанії. Наявна можливість зміни як доступних типів відсутностей, так і редагування кількості днів доступних до використання за заданими типами відсутностей. Фільтрація працівників та відділів за обраною політикою дозволяє швидко та зручно коригувати закріплених співробітників за певною політикою.

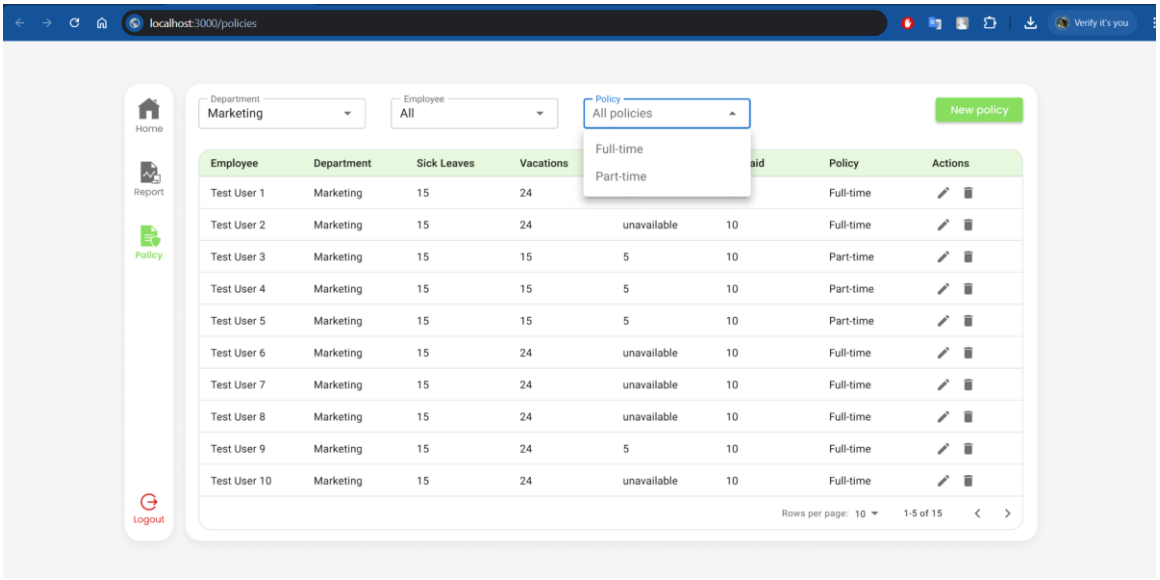

Рисунок 3.23 – Управління політиками компанії

3.4 Тестування інформаційної системи для управління відпустками

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, що сприяє забезпеченню високої якості продукту, його надійності та безпеки. Тестування дозволяє виявити дефекти на різних етапах розробки, а також підтвердити, що система відповідає вимогам користувачів і забезпечує необхідний рівень функціональності.

Основною метою тестування є перевірка того, чи виконує програмне забезпечення свої функції згідно з вимогами, чи немає в ньому помилок та чи є можливість його подальшої експлуатації в реальних умовах. Воно допомагає мінімізувати ризики, пов'язані з використанням продукту, і підвищити ефективність робочих процесів.

Види тестування програмного забезпечення:

1. Функціональне тестування — перевірка коректності виконання функцій, зазначених у вимогах. Це включає тестування інтерфейсу користувача, бізнес-логіки та взаємодії з іншими системами.

2. Нефункціональне тестування — оцінка таких аспектів, як продуктивність, зручність користування, безпека, сумісність тощо.

3. Тестування безпеки — оцінка системи на вразливості, перевірка захисту даних та ідентифікації користувачів.

4. Регресійне тестування — перевірка, чи не виникли нові дефекти після внесення змін у програмний код.

Процес тестування зазвичай включає кілька етапів:

- Підготовка тестових сценаріїв (test cases) на основі вимог.
- Виконання тестів та реєстрація результатів.
- Аналіз результатів тестування і коригування системи у разі виявлення помилок.

Для кожної функціональності визначаються тест-кейси, які описують кроки, умови та очікувані результати тестування. Це дозволяє створити чітку структуру тестування та забезпечити його відтворюваність.

Для тестування системи було розроблено на тест-кейси – детальний опис кроків, умов і очікуваного результату, що використовуються для перевірки роботи певної функції або системи. Тест-кейси для ролі адміністратора наведені у таблиці 3.9.

Таблиця 3.9 – Тест-кейси для перевірки ролі адміністратора

№	Тест-кейс
1	2
1	*Реєстрація нового працівника* *Мета:* Перевірити можливість реєстрації нового працівника в системі. *Передумови:* Адміністратор авторизований у системі. *Кроки:* 1. Перейти до розділу "Реєстрація працівників". 2. Ввести коректні дані працівника. 3. Натиснути "Зареєструвати". *Очікуваний результат:* Працівника успішно зареєстровано, дані з'являються у списку працівників. Виводиться повідомлення про успішну реєстрацію.
2	*Імпорт працівників із CSV-файла* *Мета:* Перевірити, чи можна імпортувати працівників через CSV-файл. *Передумови:* Адміністратор авторизований у системі. Підготовлений валідний CSV-файл із даними працівників. *Кроки:* 1. Перейти до розділу "Імпорт працівників". 2. Завантажити файл CSV із коректними даними. 3. Натиснути "Імпортувати". *Очікуваний результат:* Працівники з файлу додаються в систему, і відображається повідомлення про успішний імпорт. Якщо файл має некоректні дані (наприклад, відсутній email), виводиться відповідне повідомлення про помилку.

Продовження таблиці 3.9

1	2
3	*Формування політики відсутностей* *Мета:* Перевірити можливість створення політики відсутностей. *Передумови:* Адміністратор авторизований у системі. *Кроки:* 1. Перейти до розділу "Політики відсутностей". 2. Натиснути "Створити нову політику". 3. Вказати параметри політики (тип відсутності, ліміти тощо). 4. Натиснути "Зберегти". *Очікуваний результат:* Політика створена та відображається в списку.
4	*Перегляд та обробка запитів на відсутність* *Мета:* Перевірити можливість керування запитами на відсутність. *Передумови:* Адміністратор авторизований. У системі є нові запити. *Кроки:* 1. Перейти до розділу "Нові запити на відсутність". 2. Вибрати запит із переліку. 3. Натиснути "Підтвердити" або "Відхилити". *Очікуваний результат:* Запит успішно підтверджується або відхиляється, статус запиту оновлюється відповідно до дії адміністратора.
5	*Фільтрація звітів по відсутностях* *Мета:* Перевірити, чи працює фільтрація звітів за відсутностями. *Передумови:* Адміністратор авторизований. У системі є звіти по відсутностях працівників. *Кроки:* 1. Перейти до розділу "Звіти по відсутностях". 2. Вибрати тип відсутності (наприклад, відпустка). 3. Вибрати певний відділ і період. *Очікуваний результат:* Список звітів оновлюється згідно з вибраними фільтрами.

Аналогічно розроблено тест-кейси для перевірки системи у ролі користувача, які охоплюють основні сценарії взаємодії користувача з системою та наведені у таблиці 3.10.

Таблиця 3.10 – Тест-кейси для перевірки ролі користувача

№	Тест-кейс
1	2
1	*Перегляд залишку доступних вихідних днів* *Мета:* Перевірити, чи може працівник переглянути залишок доступних вихідних днів різних типів. *Передумови:* Користувач із роллю працівника авторизований у системі. *Кроки:* 1. Авторизуватися в системі. 2. Перейти до розділу доступних днів. 3. Переглянути залишок вихідних днів для всіх доступних типів (оплачувана відпустка, лікарняний, відгули). *Очікуваний результат:* Відображаються коректні значення залишків днів для кожного типу вихідних.
2	*Створення запиту на відсутність* *Мета:* Перевірити можливість створення запиту на відсутність. *Передумови:* Користувач із роллю працівника авторизований у системі. У профілі є залишок днів для вибраного типу відсутності. *Кроки:* 1. Авторизуватися в системі. 2. Обрати тип відсутності із доступних користувачеві. 3. Натиснути "Створити запит". 4. Вказати дати відсутності. 5. Натиснути "Надіслати". *Очікуваний результат:* Запит успішно створений і відображається у списку запитів із відповідним статусом.

Продовження таблиці 3.10

1	2
3	*Перегляд історії власних запитів* *Мета:* Перевірити можливість перегляду історії запитів працівника та відстеження статусу кожного запиту. *Передумови:* У системі є створені запити на відсутність (із різними статусами). *Кроки:* 1. Авторизуватися в системі. 2. Переглянути статуси запитів ("Очікує підтвердження", "Підтверджено", "Відхилено"). *Очікуваний результат:* Історія запитів відображається коректно із зазначенням дат, типів відсутності та статусів кожного запиту.
4	*Перевірка помилок при створенні запиту* *Мета:* Перевірити поведінку системи при створенні запиту з некоректними даними. *Передумови:* Користувач із роллю працівника авторизований у системі. *Кроки:* 1. Авторизуватися в системі. 2. Натиснути "Створити запит". 3. Вибрати невалідний період дат. 4. Натиснути "Надіслати". *Очікуваний результат:* Система виводить повідомлення про помилку, запит не створюється.

Результати проведеного тестування свідчать про коректну роботу основного функціоналу системи відповідно до визначених вимог. Усі критичні тест-кейси були успішно виконані, що підтверджує стабільність і надійність програмного забезпечення. Загальний стан системи оцінюється як готовий до впровадження.

3.5 Висновки до розділу

У даному розділі детально описано процес розробки інформаційної системи. Визначено функціональні та нефункціональні вимоги, що слугували основою для формування архітектури системи та визначення критеріїв її якості. Проведено проектування бази даних, яка забезпечує раціональне зберігання, доступність та цілісність даних.

Реалізовано клієнтську частину системи, орієнтовану на забезпечення зручності користувачів, та серверну частину, що відповідає за обробку запитів і підтримку стабільної роботи серверної логіки. Виконане тестування підтвердило коректність реалізації функціональності, відповідність вимогам та стійкість системи до можливих помилок.

Результат розробки підтверджує готовність інформаційної системи для управління відпустками до впровадження та подальшого використання.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи, для визначення відповідності вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки.

4.1 Проведення комерційного та технологічного аудиту

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка інформаційної системи для управління відпустками» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями [18] (таблиця 4.1).

Таблиця 4.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	3	3	3
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	3	3	3
5. Ринкові переваги (експлуатаційні витрати)	2	2	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	3	4	4
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	4	3
Сума балів	36	39	38
Середньоарифметична сума балів $СБ_c$	37,7		

За результатами розрахунків, наведених в таблиці 4.1, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в [18].

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка інформаційної системи для управління відпустками» становить 37,7 бала, що, відповідно до [18], свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

4.2 Розрахунок узагальненого коефіцієнта якості розробки

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [19]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \,, \tag{4.1}$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *і-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Результати порівняння зведемо до таблиці 4.2.

Таблиця 4.2 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований продукт	Відношення параметрів нової розробки до аналога	Питома вага показника
Кількість одночасних користувачів	шт	100	250	2,5	0,3
Середній час обробки запитів	сек	2	1,5	1,33	0,2
Середній час завантаження сторінки	сек	1,5	1	1,5	0,15
Обсяг резервної копії	Гігабайти	2	6	3	0,25
Кількість підтримуваних типів звітів	шт	2	3	1,5	0,1

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 2{,}5 \cdot 0{,}3 + 1{,}33 \cdot 0{,}2 + 1{,}5 \cdot 0{,}15 + 3 \cdot 0{,}25 + 1{,}5 \cdot 0{,}1 = 2{,}14.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 2,14 рази.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка інформаційної системи для управління відпустками», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.3.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [18]:

$$З_о = \sum_{i=1}^{k} \frac{М_{ni} \cdot t_i}{Т_р}, \quad (4.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{ni}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=21 дні.

$$З_о = 18000{,}00 \cdot 21 / 21 = 18000{,}00 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 4.3.

Таблиця 4.3 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Проектний менеджер (координатор)	18000,00	857,14	21	18000,00
Інженер-програміст	19200,00	914,29	15	13714,29
Консультант (HR-фахівець)	18000,00	857,14	4	3428,57
Технік 1-ї категорії	8100,00	385,71	18	6942,86
Всього				42085,71

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Розробка інформаційної системи для управління відпустками» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} С_i \cdot t_i, \quad (4.3)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (4.4)$$

де M_M – розмір мінімальної місячної заробітної плати, приймемо $Мм$=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення [18, таблиця Б.2];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок;

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$$C_1 = 8000{,}00 \cdot 1{,}10 \cdot 1{,}15 / (21 \cdot 8) = 60{,}24 \text{ грн.}$$

$$З_{р1} = 60{,}24 \cdot 8{,}20 = 493{,}95 \text{ грн.}$$

Величина витрат на основну заробітну плату робітників наведемо в таблиці 4.4.

Таблиця 4.4 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1	2	3	4	5	6
Встановлення обладнання для моделювання та проектування інформаційної системи	8,20	2	1,10	60,24	493,95
Підготовка робочого місця розробника програмного забезпечення	4,70	3	1,35	73,93	347,46

Продовження таблиці 4.4.

1	2	3	4	5	6
Підготовка серверного обладнання	3,20	4	1,50	82,14	262,86
Інсталяція програмного забезпечення для моделювання та розробки аналітично-інформаційної системи	6,22	4	1,50	82,14	510,93
Компіляція програмних блоків	8,00	5	1,70	93,10	744,76
Налагодження програмних блоків	4,60	4	1,50	82,14	377,86
Тестування системи	10,00	2	1,10	60,24	602,38
Всього					3340,20

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод}$ = (42085,71 + 3340,20) · 10 / 100% = 4542,59 грн.

4.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (42085,71 + 3340,20 + 4542,59) · 22 / 100% = 10993,07 грн.

4.3.3 Сировина та матеріали

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}\ , \qquad (4.7)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$М_1$ = 3,0 · 215,00 · 1,02 - 0 · 0 = 657,90 грн.

Проведені розрахунки зведемо до таблиці 4.5.

Таблиця 4.5 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
1	2	3	4	5	6
Офісний папір Cristal A4 500	215,00	3,0	0	0	657,90
Папір для записів Cristal LightPapers 65 A5	100,00	5,0	0	0	510,00
Органайзер офісний Cristal	175,00	2,0	0	0	357,00
Набір офісний Cristal Base	190,00	3,0	0	0	581,40
Картридж для принтера Canon LBP5000	780,00	1,0	0	0	795,60

Продовження таблиці 4.5.

1	2	3	4	5	6
Диск оптичний Vybir CD-R	23,25	3,0	0	0	71,15
Flesh-пам'ять Kingston 32 GB	169,00	1,0	0	0	172,38
Тека для паперів	89,00	4,0	0	0	363,12
Інші матеріали	200,00	1,0	0	0	204,00
Всього					3712,55

4.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка інформаційної системи для управління відпустками», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \qquad (4.8)$$

де $Н_j$ – кількість комплектуючих *j*-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих *j*-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*).

$К_в = 1 \cdot 3820,00 \cdot 1,02 = 3896,40$ грн.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
RouterBOARD TP-Link230	1	3820,00	3896,40
База даних (тестувальна) DataBase 64-A10	1	1750,00	1785,00
Всього			5681,40

4.3.5 Спецустаткування для наукових (експериментальних) робіт

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \quad (4.9)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10...1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 44599,00 \cdot 1 \cdot 1,02 = 45490,98$ *грн.*

Отримані результати зведемо до таблиці 4.7.

Таблиця 4.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Серверне обладнання на основі ЕОМ ASUS AJ13-A71BC	1	44599,00	45490,98
Всього			45490,98

4.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \quad (4.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг}$ = *6820,00 · 1 · 1,02 = 6956,40 грн.*

Отримані результати зведемо до таблиці 4.8.

Таблиця 4.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	К-сть, шт	Ціна за шт, грн	Вартість
Емулятор серверу для моделювання поведінки інформаційної системи	1	6820,00	6956,40
Середовище розробки Visual Studio 2019	1	9320,00	9506,40
Платформа .NET Framework 4.8	1	5280,00	5385,60
Абонентна плата доступу	1	352,00	359,04
Всього			22207,44

4.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл}$ = *(32999,00 · 1) / (2 · 12) = 1374,96 грн.*

Проведені розрахунки зведемо до таблиці 4.9.

Таблиця 4.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер розробника ПЗ	32999,00	2	1	1374,96
Персональний комп'ютер розробника аналітично-інформаційних систем	24999,00	2	1	1041,63
Робоче місце інженера-програміста	8700,00	5	1	145,00
Робоче місце аналітика	8700,00	5	1	145,00
Пристрої передачі даних	6800,00	4	1	141,67
Оргтехніка	8100,00	4	1	168,75
Приміщення лабораторії розробки ІС	415000,00	25	1	1383,33
OS Windows 11	6800,00	2	1	283,33
Прикладний пакет Microsoft Office	6700,00	2	1	279,17
Всього				4962,83

4.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \qquad (4.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; приймемо Ц$_e$ = 10,98 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$$B_e = 0,42 \cdot 160,0 \cdot 10,98 \cdot 0,95 / 0,97 = 737,86 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 4.10

Таблиця 4.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер розробника ПЗ	0,42	160,0	737,86
Персональний комп'ютер розробника аналітично-інформаційних систем	0,32	160,0	562,18
Робоче місце інженера-програміста	0,12	120,0	158,11
Робоче місце аналітика	0,10	20,0	21,96
Пристрої передачі даних	0,05	120,0	65,88
Оргтехніка	0,45	1,3	6,18
Серверне обладнання на основі ЕОМ ASUS AJ13-A71BC	0,10	120,0	131,76
RouterBOARD TP-Link230	0,02	160,0	35,14
Всього			1719,06

4.3.9 Службові відрядження

Витрати за статтею «Службові відрядження» відсутні.

4.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.13)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$$B_{сп} = (42085,71 + 3340,20) \cdot 30 / 100\% = 13627,78 \text{ грн.}$$

4.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.14)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в = (42085,71 + 3340,20) \cdot 50 / 100\% = 22712,96$ грн.

4.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}\ , \qquad (4.15)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (42085,71 + 3340,20) · 100 / 100% = 45425,92 грн.

Витрати на проведення науково-дослідної роботи на тему «Розробка інформаційної системи для управління відпустками» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \qquad (4.16)$$

$В_{заг}$= 42085,71 +3340,20 +4542,59 +10993,07 +3712,55 +5681,40 +45490,98 + 22207,44 + 4962,83 +1719,06 +0,00 +13627,78 +22712,96 +45425,92 = 226502,48 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \qquad (4.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 226502,48 / 0,9 = 251669,43 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Результати дослідження проведені за темою «Розробка інформаційної системи для управління відпустками» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються (таблиця 4.11).

Таблиця 4.11 – Показники збільшення кількості споживачів

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	550	900	1500	1300

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 75000 осіб;

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 70,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо 29,87 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [18]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.18)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 42\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (29{,}87 \cdot 75000{,}00 + 99{,}87 \cdot 550) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 656081{,}36$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (29{,}87 \cdot 75000{,}00 + 99{,}87 \cdot 1450) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 681774{,}60$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (29{,}87 \cdot 75000{,}00 + 99{,}87 \cdot 2950) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 724596{,}66$ грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (29{,}87 \cdot 75000{,}00 + 99{,}87 \cdot 4250) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 761709{,}11$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{4.19}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}25$;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП = 656081{,}36/(1+0{,}25)^1 + 681774{,}60/(1+0{,}25)^2 + 724596{,}66/(1+0{,}25)^3 +$

$+ 761709{,}11/(1+0{,}25)^4 = 524865{,}09 + 436335{,}74 + 370993{,}49 + 311996{,}05 =$

$= 1644190{,}38$ грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{4.20}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 251669,43 грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 251669{,}43 = 503338{,}86$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{4.21}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 1644190,38 грн;

PV – теперішня вартість початкових інвестицій, 503338,86 грн.

$$E_{абс} = ПП - PV = 1644190{,}38 - 503338{,}86 = 1140851{,}52 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.22}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 1140851,52 грн;

PV – теперішня вартість початкових інвестицій, 503338,86 грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_{в} = \sqrt[Тж]{1+\frac{E_{абс}}{PV}} - 1 = (1+1140851,52/503338,86)^{1/4} = 0,34.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{4.23}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,15.

$\tau_{мін}$ = 0,11+0,15 = 0,26 < 0,34 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка інформаційної системи для управління відпустками» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_{в}}, \tag{4.24}$$

де $E_{в}$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,34 = 2,90 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка інформаційної системи для управління відпустками» становить 37,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 2,14 рази.

Також термін окупності становить 2,90 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка інформаційної системи для управління відпустками».

ВИСНОВКИ

Розробка системи для управління відпустками є важливим кроком у підвищенні ефективності управлінських процесів, спрощенні обліку та забезпеченні прозорості процесів у компаніях. Зокрема, автоматизація процесів обробки запитів, а також інтеграція з ERP-системамою для фінансових обрахунків, забезпечує синхронізацію даних і зменшує кількість помилок, що виникають через людський фактор.

У процесі розробки інформаційної системи для управління відпустками було успішно вирішено ряд актуальних завдань автоматизації та оптимізації кадрових процесів. Проведений аналіз предметної області та існуючих систем-аналогів дозволив створити рішення, яке враховує специфічні потреби підприємств та забезпечує гнучкість у налаштуванні політик відпусток відповідно до різних умов праці, таких як тип контракту, зайнятості та особливі домовленості з працівниками.

Вибір технологічного стеку MERN сприяв створенню продуктивного та масштабованого програмного забезпечення, яке забезпечує стабільну роботу як на серверній, так і на клієнтській частинах. Проектування бази даних і реалізація бізнес-логіки дозволили створити систему, здатну ефективно обробляти запити та забезпечувати зручний інтерфейс для користувачів.

Тестування системи підтвердило її функціональність і відповідність вимогам, що свідчить про надійність і ефективність розробленого рішення. Система здатна ефективно працювати з базою даних обсягом до 50 млн документів у колекціях MongoDB, що відповідає близько 200 ГБ даних, залежно від структури документів і рівня індексації. Це дозволяє забезпечувати тривале використання системи без необхідності негайного масштабування. Завдяки використанню базового кластеру MongoDB та розміщенню на сервері з 8 ГБ оперативної пам’яті, середній час обробки запиту становить близько 100-150 мс за умов стандартного навантаження.

Економічне обґрунтування показало, що проект має високий потенціал комерціалізації та є рентабельним у контексті реального впровадження. Згідно проведених економічних розрахунків, рівень комерційного потенціалу розробки становить 37,7 бала, що свідчить про її високу цінність. За технічними параметрами система переважає існуючі аналоги приблизно в 2,14 рази. Термін окупності становить 2,90 року, що підтверджує її комерційну привабливість.

Усі задачі, поставлені в межах магістерської дипломної роботи, виконано в повному обсязі. Створена інформаційна система забезпечує оптимізацію процесів управління відпустками, відповідає сучасним вимогам ефективності та масштабованості, а також готова до практичного використання. Отримані результати підтверджують відповідність розробки визначеним цілям та її економічну доцільність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Голод С. В., Богач І. В. Розробка інформаційної системи для управління відпустками. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/index (дата звернення 28.11.2024).

2. Голод С.В. Навчальна платформа для підготовки до екзаменів з української мови. Матеріали LII науково-технічної конференції підрозділів Вінницького національного технічного університету. Вінниця: ВНТУ, 2023. URL: https://press.vntu.edu.ua/index.php/vntu/catalog/view/788/1373/2632-1 (дата звернення 28.11.2024)

3. Види відпусток та загальні засади їх надання. *Південне міжрегіональне управління Державної служби з питань праці* : веб-сайт. URL: https://pd.dsp.gov.ua/news/vydy-vidpustok-ta-zahalni-zasady-ikh-n/ (дата звернення 13.10.2024)

4. Конвенція Про щорічні оплачувані відпустки N52. URL: https://zakon.rada.gov.ua/laws/show/993_003#Text (дата звернення 13.10.2024)

5. Види відпусток. *Factor* : веб-сайт. URL: https://i.factor.ua/ukr/journals/nibu/2015/april/issue-33/article-7458.html (дата звернення 15.10.2024)

6. Безлімітні відпустки в IT-компаніях: як це працює і чи справді вони необмежені. *DOU* : веб-сайт. URL: https://dou.ua/lenta/articles/unlimited-vacations-in-it-companies/ (дата звернення 15.10.2024)

7. Результати дослідження «Відпустки в компаніях». *Бізнес-школа УКУ* : веб-сайт. URL: https://lvbs.com.ua/news/rezultaty-doslidzhennya-vidpustky-v-kompaniyah-infografika/ (дата звернення 16.10.2024)

8. Система для керування відсутністю співробітників. *Hurma* : веб-сайт. URL: https://hurma.work/ (дата звернення 16.10.2024)

9. BambooHR. *BambooHR* : веб-сайт. URL: https://www.bamboohr.com/k1 (дата звернення 16.10.2024)

10. Облік вихідних та відпусток. *PeopleForce* : веб-сайт. URL: https://peopleforce.io/uk/products/peoplehr/leave-tracking (дата звернення 18.10.2024)

11. MERN Stack Explained. *MongoDB* : веб-сайт. URL: https://www.mongodb.com/resources/languages/mern-stack (дата звернення 20.10.2024)

12. Introduction to MongoDB. *MongoDB* : веб-сайт. URL: https://www.mongodb.com/docs/manual/introduction/ (дата звернення 20.10.2024)

13. MongoDB. Data Types. *MongoDB* : веб-сайт. URL: https://www.mongodb.com/docs/mongodb-shell/reference/data-types/ (дата звернення 20.10.2024)

14. Developer ecosystem (Databases). *JetBrains* : веб-сайт. URL: https://www.jetbrains.com/lp/devecosystem-2022/databases/ (дата звернення 21.10.2024)

15. Fast, unopinionated, minimalist web framework for Node.js. *Express* : веб-сайт. URL: https://expressjs.com/ (дата звернення 25.10.2024)

16. JavaScript-бібліотека для створення користувацьких інтерфейсів. *React* : веб-сайт. URL: https://uk.legacy.reactjs.org/ (дата звернення 30.10.2024)

17. Node.js v23.3.0 documentation. *Node.js* : веб-сайт URL: https://nodejs.org/docs/latest/api/documentation.html (дата звернення 30.10.2024)

18. Козловський В. О., Лесько О.Й., Кавецький В.В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2021. 42 с.

19. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

20. Randy Connolly, Ricardo Hoar. Fundamentals of Web Development. 3rd Edition. Boston: Pearson, 2021. 1031 p.

21. Ethan Brown. Web Development with Node and Express: Leveraging the JavaScript Stack 2nd Edition. Sebastopol: O'Reilly, 2019. 340 p.

22. Молчанов В. П., Пандорін О. К. Технології розробки WEB-ресурсів: навчальний посібник. Харків: ХНЕУ, 2019. 130 с.

23. Wandschneider M. Learning Node.js. MA: Addison-Wesley, 2013. 396с.

24. Ambler S, Sadalage P. Refactoring Databases: Evolutionary Database Design. MA: Addison-Wesley Professional, 2006. 384c.

25. Simpson K. You Don't Know JS. CA : O'Reilly Media, 2014. 98c.

26. Холмс С. Стек MEAN. Mongo, Express, Angular: навч. посіб. Київ: Шкільний світ, 2017. 496 с.

27. Клієнт-серверна архітектура. *QATestLab* : веб-сайт. URL: https://training.qatestlab.com/blog/ technical-articles/client-server-architecture/ (дата звернення 02.10.2024).

28. Advantages and disadvantages of TypeScript over JavaScript. *GeeksforGeeks* : веб-сайт. URL: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-typescript-over-javascript/ (дата звернення: 24.10.2024).

29. Створення веб-додатку: основні аспекти ефективної розробки веб-сайту. *Web Book* : веб-сайт. URL: https://webbookstudio.com/ua/articles/building-a-web-application-key-considerations-for-efferctive-website-development/ (дата звернення 15.10.2024)

30. Shama Hoque. Full-Stack React Projects. Second Edition. Birmingham: Packt Publishing Ltd, 2020. 706 p.

31. Kristina Chodorow, Eoin Brazil, Shannon Bradshaw. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition. Sebastopol: O'Reilly, 2020. 512 p.

32. Alex Banks. Learning React: Functional Web Development with React and Redux 1st Edition. Sebastopol: O'Reilly, 2017. 320 p.

33. Philip Ackermann. Full Stack Web Development: The Comprehensive Guide. Bonn: SAP Press, 2023. 800 p.

Додатки

Додаток А (обов'язковий)

Технічне завдання

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«__» _____2024 року

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ ВІДПУСТКАМИ»
08-31.МКР.002.02.000 ТЗ

Керівник роботи:
к.т.н., доц. каф. АІІТ
Ілона БОГАЧ
«__» _____ 2024 р.

Виконавець:
ст. гр. 1ІСТ-23м
Світлана ГОЛОД
«__» _____ 2024 р.

Вінниця ВНТУ – 2024

1. Назва та галузь застосування.

Розробка інформаційної системи для управління відпустками. Інформаційні системи та технології.

2. Підстава для розробки.

Розробку системи здійснювати на підставі наказу по університету № 310 від 17 вересня 2024 та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій»

3. Мета та призначення розробки.

Метою роботи є розробка інформаційної системи для управління відпустками, яка забезпечить автоматизацію процесів подання та обробки заявок, спростить процес управління відпустками, а також створить зручні інструменти для аналізу та моніторингу використання відпусток і лікарняних на підприємстві.

4. Джерела розробки:

1. Randy Connolly, Ricardo Hoar. Fundamentals of Web Development. 3rd Edition. Boston: Pearson, 2021. 1031 p.

2. Ethan Brown. Web Development with Node and Express: Leveraging the JavaScript Stack 2nd Edition. Sebastopol: O'Reilly, 2019. 340 p.

3. Молчанов В. П., Пандорін О. К. Технології розробки WEB-ресурсів: навчальний посібник. Харків: ХНЕУ, 2019. 130 с.

4. Kristina Chodorow, Eoin Brazil, Shannon Bradshaw. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition. Sebastopol: O'Reilly, 2020. 512 p.

5. Alex Banks. Learning React: Functional Web Development with React and Redux 1st Edition. Sebastopol: O'Reilly, 2017. 320 p.

6. Philip Ackermann. Full Stack Web Development: The Comprehensive Guide. Bonn: SAP Press, 2023. 800 p.

5. Показники призначення

Основні технічні вимоги та мінімальні системні вимоги до програми:

- Оперативна пам’ять: 8 GB оперативної;
- ОС: Сумісність із Windows 10 та вище, macOS 10.15 та вище;
- Процесор: тактова частота не менше 2 GHz
- Node.js v18.12.0, React v18.0

Вхідні дані:

Інформація про співробітників, стек технологій для розробки продукту MERN, нормативні дані, API для інтеграції.

Результати роботи програми:

Співробітники можуть подавати заявки на відпустки через інтерфейс системи; менеджери отримують сповіщення про нові заявки на електронну пошту та в системі; інформація про затверджені або відхилені заявки синхронізується з корпоративним календарем; генерація звітів.

6. Економічні показники

До економічних показників входять:

- витрати на розробку – до 20 тис. грн.
- мінімальна дохідність – не менше 60 тис. грн.
- термін окупності – не більше 1 року

7. Стадії розробки:

1. Розділ 1 «Аналіз предметної області та аналогів системи» має бути виконаний до 02.10.2024.

2. Розділ 2 «Огляд програмних засобів для реалізації поставленої задачі» має бути виконаний до 10.10.2024.

3. Розділ 3 «Розробка та тестування програмного забезпечення» має бути виконаний до 15.11.2024.

4. Розділ 4 «Економічний розділ» має бути виконаний до 28.11.2024.

8. Порядок контролю та приймання

1. Рубіжний контроль провести до 01.11.2024.

2. Попередній захист магістерської кваліфікаційної роботи провести до 01.12.2024.

3. Захист магістерської кваліфікаційної роботи провести 19.12.2024.

Додаток Б (обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ ВІДПУСТКАМИ

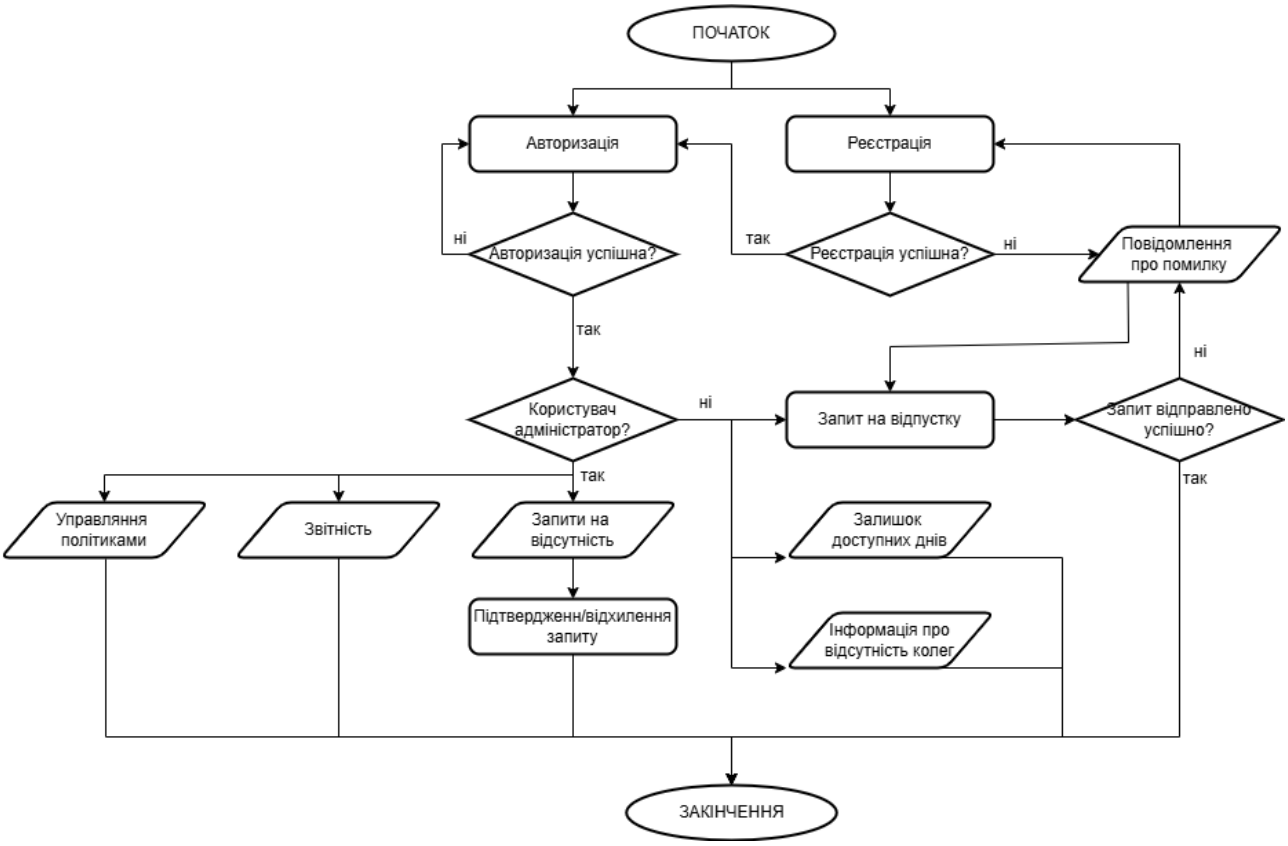

Рисунок Б.1 - Схема алгоритму роботи інформаційної системи для управління відпусток

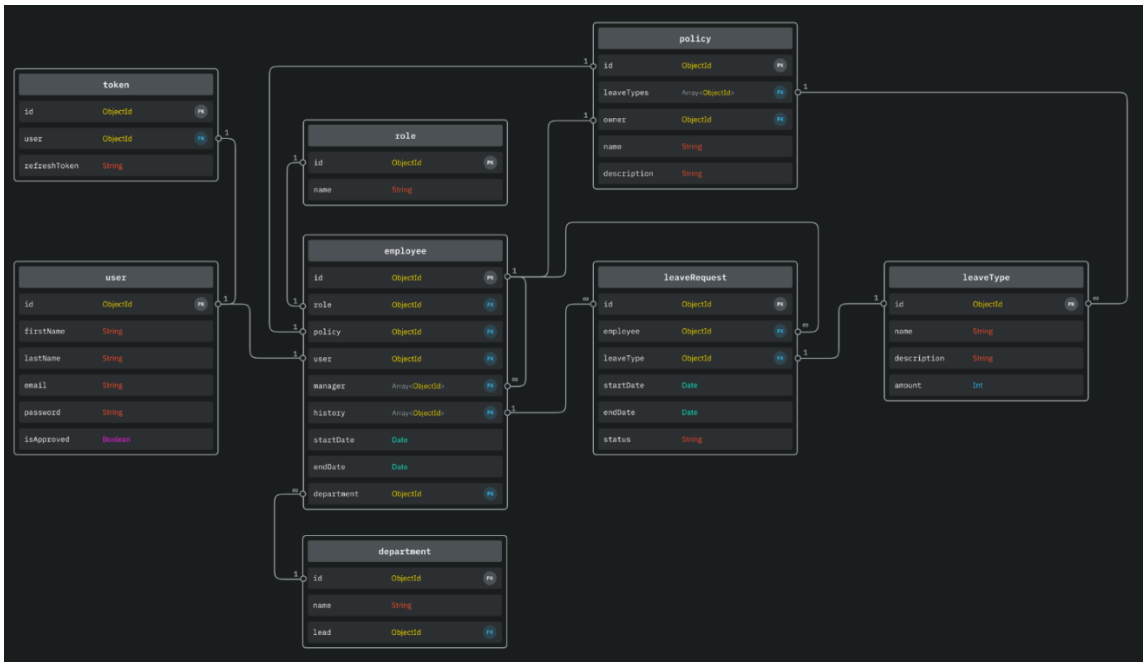

Рисунок Б.2 – Модель бази даних інформаційної системи для управління відпусток

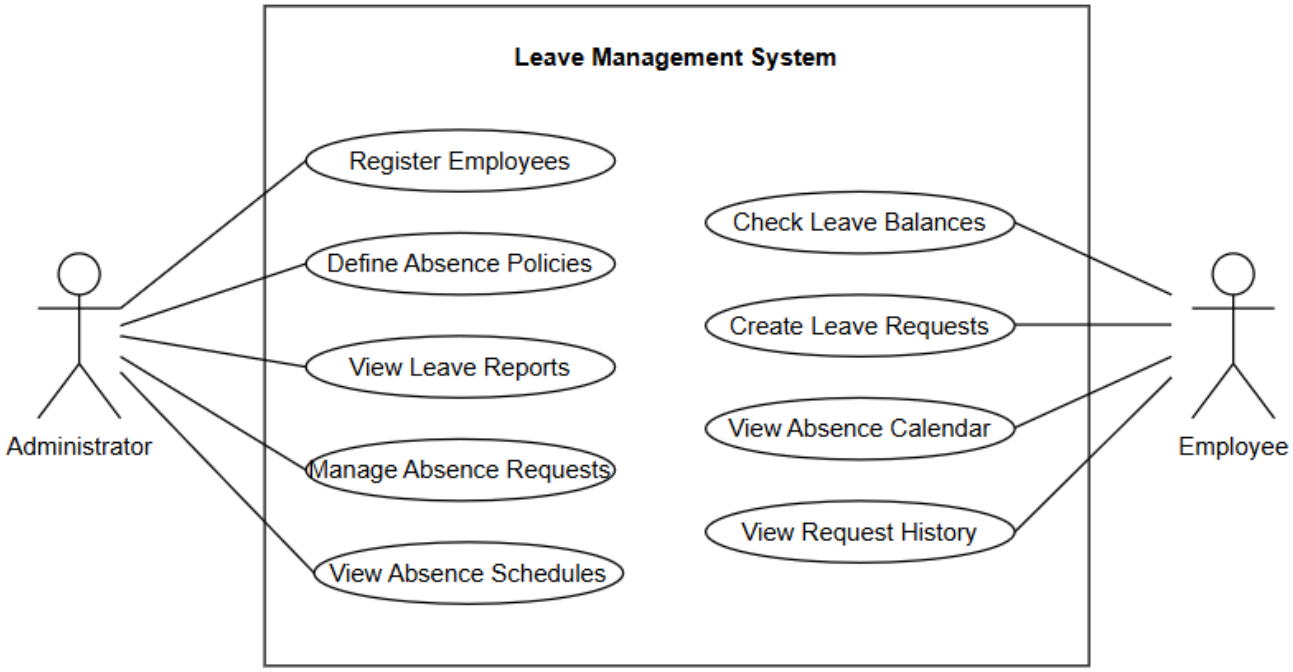

Рисунок Б.3 – UML-діаграма прецедентів інформаційної системи для управління відпусток

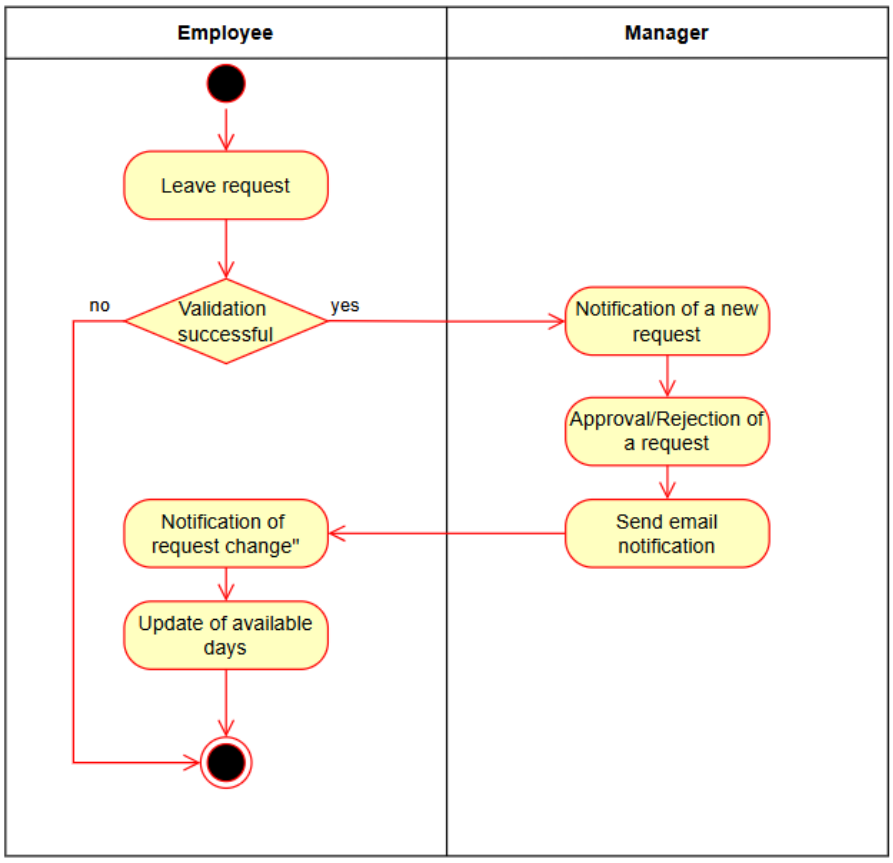

Рисунок Б.4 – UML-діаграма діяльності процесу створення запиту на відсутність

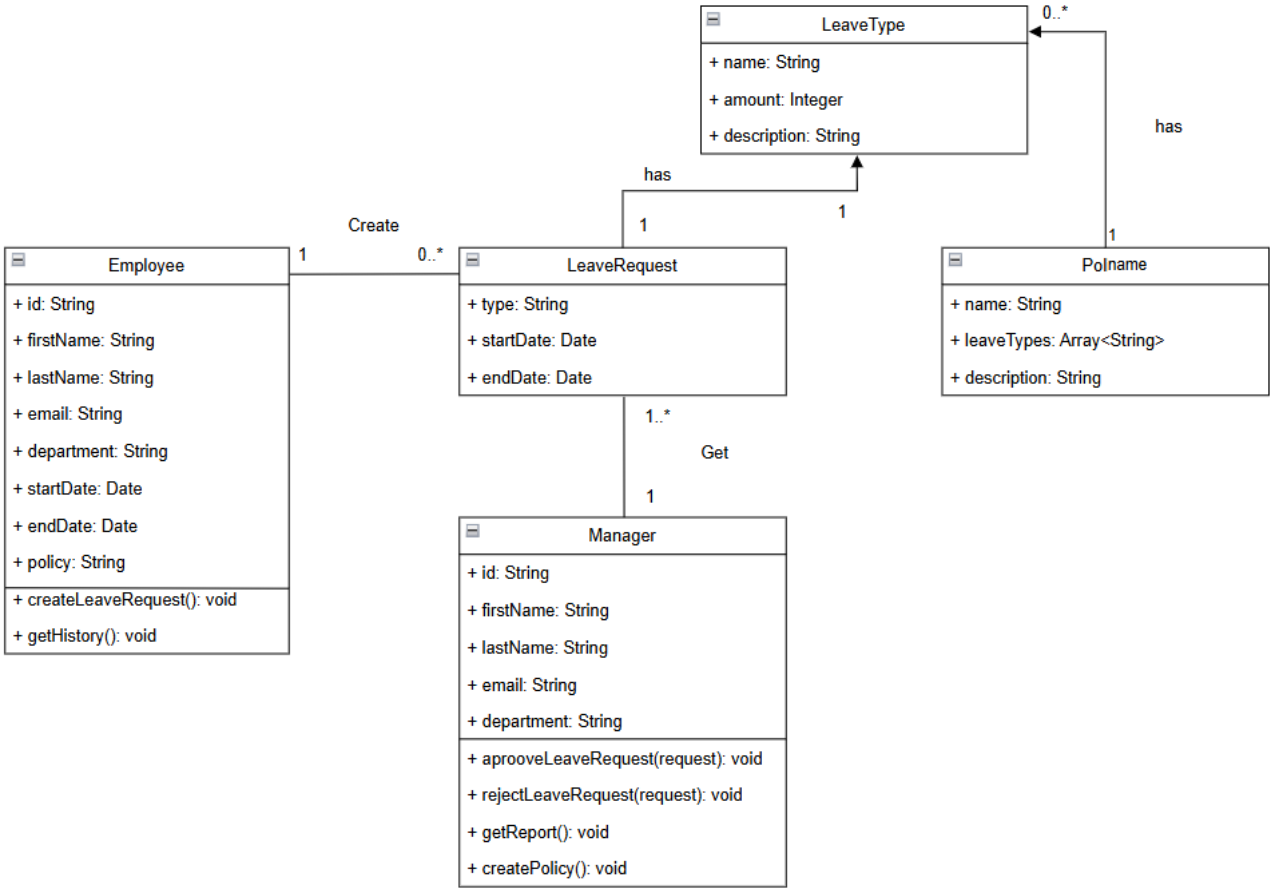

Рисунок Б.5 – UML-діаграма класів інформаційної системи для управління відпусток

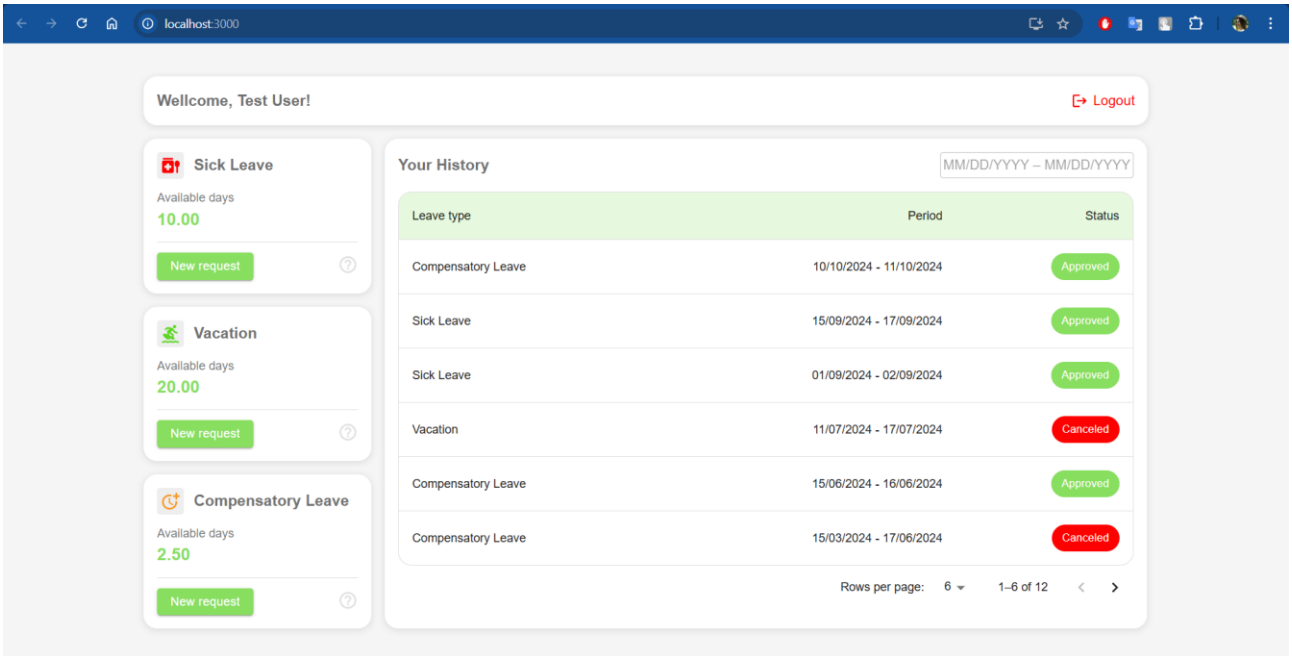

Рисунок Б.6 – Приклад інтерфейсу розробленої системи

Додаток В (обов'язковий)

Лістинг основних функцій

Фрагмент коду сервісу користувача (UserServise)

```
const UserModel = require('../models/user-model');
const bcrypt = require('bcrypt');
const uuid = require('uuid');
const mailService = require('./mail-service');
const tokenService = require('./token-service');
const UserDto = require('../dtos/user-dto');
const ApiError = require('../exceptions/api-error');

class UserService {
    async registration(email, password) {
        const candidate = await UserModel.findOne({email})
        if (candidate) {
            throw ApiError.BadRequest(`User with email ${email} already exists`)
        }
        const hashPassword = await bcrypt.hash(password, 3);
        const activationLink = uuid.v4();
        const user = await UserModel.create({email, password: hashPassword, activationLink})
        await mailService.sendActivationMail(email,
`${process.env.API_URL}/api/activate/${activationLink}`);
        const userDto = new UserDto(user);
        const tokens = tokenService.generateTokens({...userDto});
        await tokenService.saveToken(userDto.id, tokens.refreshToken);
        return {...tokens, user: userDto}
    }
    async activate(activationLink) {
        const user = await UserModel.findOne({activationLink})
        if (!user) {
            throw ApiError.BadRequest('Wrong activation link')
        }
        user.isActivated = true;
```

```
        await user.save();
    }
    async login(email, password) {
        const user = await UserModel.findOne({email})
        if (!user) {
            throw ApiError.BadRequest('Can not find user with this email')
        }
        const isPassEquals = await bcrypt.compare(password, user.password);
        if (!isPassEquals) {
            throw ApiError.BadRequest('Wrong password');
        }
        const userDto = new UserDto(user);
        const tokens = tokenService.generateTokens({...userDto});

        await tokenService.saveToken(userDto.id, tokens.refreshToken);
        return {...tokens, user: userDto}
    }
    async logout(refreshToken) {
        const token = await tokenService.removeToken(refreshToken);
        return token;
    }
    async refresh(refreshToken) {
        if (!refreshToken) {
            throw ApiError.UnauthorizedError();
        }
        const userData = tokenService.validateRefreshToken(refreshToken);
        const tokenFromDb = await tokenService.findToken(refreshToken);
        if (!userData || !tokenFromDb) {
            throw ApiError.UnauthorizedError();
        }
        const user = await UserModel.findById(userData.id);
        const userDto = new UserDto(user);
        const tokens = tokenService.generateTokens({...userDto});

        await tokenService.saveToken(userDto.id, tokens.refreshToken);
```

```
        return {...tokens, user: userDto}
    }
    async getAllUsers() {
        const users = await UserModel.find();
        return users;
    }
}
module.exports = new UserService();
```

Фрагмент коду сервісу повідомлень (MailServise)

```
const nodemailer = require('nodemailer');
class MailService {
    constructor() {
        this.transporter = nodemailer.createTransport({
            host: process.env.SMTP_HOST,
            port: process.env.SMTP_PORT,
            secure: false,
            auth: {
                user: process.env.SMTP_USER,
                pass: process.env.SMTP_PASSWORD
            }
        })
    }
    async sendActivationMail(to, link) {
        await this.transporter.sendMail({
            from: process.env.SMTP_USER,
            to,
            subject: 'Account activation' + process.env.API_URL,
            text: '',
            html:
                ` <div>
                    <h1>Activate your account by this link:</h1>
                    <a href="${link}">${link}</a>
                  </div>`
        })}
```

```
  async sendLeaveRequestMail(employeeName, managerEmail, leaveDetails) {
    const { type, startDate, endDate, description } = leaveDetails;
    await this.transporter.sendMail({
      from: process.env.SMTP_USER,
      to: managerEmail,
      subject: `Leave Request from ${employeeName}`,
      text: '',
      html:
        ` <div>
            <h1>New Leave Request</h1>
            <p><strong>Employee:</strong> ${employeeName}</p>
            <p><strong>Type:</strong> ${type}</p>
            <p><strong>Start Date:</strong> ${startDate}</p>
            <p><strong>End Date:</strong> ${endDate}</p>
            <p><strong>Description:</strong> ${description}</p>
          </div>`
    });
  }
  async sendStatusChangeMail(employeeEmail, requestDetails) {
    const { type, status } = requestDetails;
    await this.transporter.sendMail({
      from: process.env.SMTP_USER,
      to: employeeEmail,
      subject: `Your Leave Request Status Update`,
      text: '',
      html:
        `<div>
            <h1>Status Update for Your Leave Request</h1>
            <p><strong>Request Type:</strong> ${type}</p>
            <p><strong>New Status:</strong> ${status}</p>
          </div>`
    });
  }
}
module.exports = new MailService();
```

Фрагмент коду сервісу працівника (EmployeeServise)

```
const LeaveRequest = require('../models/LeaveRequest');
const Employee = require('../models/Employee');
const mailService = require('./MailService');

class EmployeeService {
  async createLeaveRequest(employeeId, leaveRequestData) {
    const { leaveType, startDate, endDate, status } = leaveRequestData;
    const employee = await Employee.findById(employeeId).populate('manager');
    if (!employee) {
      throw new Error('Employee not found');
    }
    const leaveRequest = new LeaveRequest({
      employee: employeeId,
      leaveType,
      startDate,
      endDate,
      status
    });
    await leaveRequest.save();
    if (employee.manager && employee.manager.length > 0) {
      const leaveDetails = {
        type: leaveType,
        startDate,
        endDate,
        reason: `Leave requested by ${employee.user}`,
      };
      for (const managerId of employee.manager) {
        const manager = await Employee.findById(managerId).populate('user');
        if (manager && manager.user.email) {
          await mailService.sendLeaveRequestMail(
            `${employee.user}`,
            manager.user.email,
            leaveDetails
          );
```

```
            }
        }
    }
    return leaveRequest;
  }
  async getLeaveHistory(employeeId) {
    const employee = await Employee.findById(employeeId);
    if (!employee) {
      throw new Error('Employee not found');
    }
    const leaveRequests = await LeaveRequest.find({ employee: employeeId })
      .populate('leaveType')
      .sort({ startDate: -1 });
    return leaveRequests;
  }
}
module.exports = new EmployeeService();
```

Фрагмент коду сервісу працівника (ManagerServise)

```
const Employee = require('../models/Employee');
const LeaveRequest = require('../models/LeaveRequest');
const UserService = require('./UserService');
const mailService = require('./MailService');
class ManagerService {
  async registerEmployee(managerId, employeeData) {
    const { email, firstName, lastName, role, department, startDate, endDate } = employeeData;
    const manager = await Employee.findById(managerId);
    if (!manager) {
      throw new Error('Manager not found');
    }
    const password = uuid.v4().slice(0, 8);
    const user = await UserService.registration(email, password);
    const employee = new Employee({
      user: user.user.id,
      role,
```

```
        department,
        startDate,
        endDate,
        firstName,
        lastName,
        manager: [managerId],
    });
    await employee.save();
    await mailService.sendActivationMail(
        email,
        `${process.env.API_URL}/api/activate/${user.user.activationLink}`
    );
    return employee;
}
async handleLeaveRequest(managerId, leaveRequestId, status) {
    if (!['Approved', 'Rejected'].includes(status)) {
        throw new Error('Invalid status. Use "Approved" or "Rejected".');
    }
    const leaveRequest = await LeaveRequest.findById(leaveRequestId).populate('employee');
    if (!leaveRequest) {
        throw new Error('Leave request not found');
    }
    const manager = await Employee.findById(managerId);
    if (!manager || !leaveRequest.employee.manager.includes(managerId)) {
        throw new Error('Manager is not authorized to handle this request');
    }
    leaveRequest.status = status;
    await leaveRequest.save();
    const employee = await Employee.findById(leaveRequest.employee._id).populate('user');
    if (employee.user.email) {
        await mailService.sendLeaveStatusUpdateMail(
            employee.user.email,
            status,
            {
                startDate: leaveRequest.startDate,
```

```
                endDate: leaveRequest.endDate,
                leaveType: leaveRequest.leaveType,
            }
        );
    }
    return leaveRequest;
  }
  async getReport(managerId) {
    const manager = await Employee.findById(managerId);
    if (!manager) {
      throw new Error('Manager not found');
    }
    const employees = await Employee.find({ manager: managerId }).select('_id');
    const leaveRequests = await LeaveRequest.find({ employee: { $in: employees } })
      .populate('employee', 'user')
      .populate('leaveType')
      .sort({ startDate: -1 });
    const report = leaveRequests.map((request) => ({
      employeeName: `${request.employee.user.firstName} ${request.employee.user.lastName}`,
      leaveType: request.leaveType.name,
      startDate: request.startDate,
      endDate: request.endDate,
      status: request.status,
    }));
    return report;
  }
}
module.exports = new ManagerService();
```

Фрагмент коду інтерфейсу користувача (RequestDialog.jsx)

```
import React from "react";
import { Dialog, DialogTitle, DialogContent,
DialogActions, Button, TextField, Box } from "@mui/material";

function RequestDialog({ open, onClose, type }) {
```

```
 return (
   <Dialog open={open} onClose={onClose}>
     <DialogTitle>Request {type}</DialogTitle>
     <DialogContent>
       <Box display="flex" flexDirection="column" gap={2}>
         <TextField label="Start Date" type="date" InputLabelProps={{ shrink: true }} />
         <TextField label="End Date" type="date" InputLabelProps={{ shrink: true }} />
       </Box>
     </DialogContent>
     <DialogActions>
       <Button onClick={onClose} color="secondary">
         Cancel
       </Button>
       <Button onClick={onClose} variant="contained" color="primary">
         Send
       </Button>
     </DialogActions>
   </Dialog>
 );
}

export default RequestDialog;
```

Фрагмент коду інтерфейсу користувача (LeaveHistory.jsx)

```
import React from "react";
import { Table, TableBody, TableCell, TableContainer, TableHead, TableRow, Paper } from
"@mui/material";

function LeaveHistory({ history }) {
 return (
   <TableContainer component={Paper}>
     <Table>
       <TableHead>
         <TableRow>
           <TableCell>Leave Type</TableCell>
```

```
          <TableCell>Period</TableCell>
          <TableCell>Status</TableCell>
        </TableRow>
      </TableHead>
      <TableBody>
        {history.map((item, index) => (
          <TableRow key={index}>
            <TableCell>{item.name}</TableCell>
            <TableCell>{item.period}</TableCell>
            <TableCell>{item.status}</TableCell>
          </TableRow>
        ))}
      </TableBody>
    </Table>
  </TableContainer>
 );
}

export default LeaveHistory;
```

Фрагмент коду інтерфейсу користувача (LeaveTypeCard.jsx)

```
import React from "react";
import { Card, CardContent, Typography, Button, Box, Icon } from "@mui/material";
import BeachAccessIcon from "@mui/icons-material/BeachAccess";
import HealingIcon from "@mui/icons-material/Healing";
import AvTimerIcon from "@mui/icons-material/AvTimer";

function LeaveTypeCard({ type, availableDays, color, onRequestClick }) {
 const getIcon = (leaveType) => {
  switch (leaveType) {
    case "Vacation":
      return <BeachAccessIcon color="primary" />;
    case "Sick Leave":
      return <HealingIcon color="error" />;
    case "Compensatory Leave":
```

```
      return <AvTimerIcon color="warning" />;
    default:
      return null;
  }
 };

 return (
  <Card variant="outlined">
   <CardContent>
    <Box display="flex" alignItems="center" gap={2}>
     {getIcon(type)}
     <Typography variant="h6">{type}</Typography>
    </Box>
    <Typography color={color}>Available days: {availableDays}</Typography>
    <Box marginTop={2}>
     <Button variant="contained" color="primary" onClick={onRequestClick}>
      New request
     </Button>
    </Box>
   </CardContent>
  </Card>
 );
}

export default LeaveTypeCard;
```

Додаток Г (обов'язковий)

Протокол перевірки магістерської кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: «Розробка інформаційної системи для управління відпустками»
Тип роботи: магістерська кваліфікаційна робота
Підрозділ: кафедра АІІТ

Показники звіту подібності Turnitin

Оригінальність 95,0 % Схожість 5,0 %

Аналіз звіту подібності (відмітити потрібне):

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.
□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ
(підпис)

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи _____ Світлана ГОЛОД
(підпис)

Керівник роботи _____ Ілона БОГАЧ
(підпис)