

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра «Комп'ютерні системи та мережі»
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему СИСТЕМА ПІДТРИМКИ КЛІЄНТСЬКИХ ЗАПИТІВ ІЗ ВИКОРИСТАННЯМ
МОБІЛЬНОГО ТА ВЕБДОДАТКІВ

Виконав: студент 2 курсу, групи КНТ-513м
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерні системи та мережі»

БАЛАБУХА О.М.

(ПРИЗВИЩЕ та ініціали)

Керівник КИРИЧЕК Г.Г.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЄФИМЕНКО М.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти магістр

Спеціальність 123 «Комп'ютерна інженерія»

(код найменування)

Освітня програма (спеціалізація) «Комп'ютерні системи та мережі»

(нова назва освітньої програми(спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

«_____» _____ 2024 року

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БАЛАБУХА Олексій Миколайович

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Система підтримки клієнтських запитів з використанням мобільного та вебдодатків

керівник проекту (роботи) КИРИЧЕК Галина Григорівна, к.т.н., доцент

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

затверджені наказом закладу вищої освіти від «18» жовтня 2024 року № 149.

2. Строк подання студентом проекту (роботи) 10 грудня 2024 року

3. Вихідні дані до проекту (роботи) дані ресурсів мережі інтернет, літературні та електронні джерела, перелік використовуваних програмних засобів: JavaScript, TypeScript, React, Nest.js, MongoDB, Firebase, Sockets, Material ui

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

1)Предметна область та вимоги;

2)Вибір архітектури та компоненти системи;

3)Реалізація та впровадження системи;

4)Дослідження ефективності підтримки клієнтських запитів.

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

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	КИРИЧЕК Г. Г., к.т.н., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання «01» _____ вересня _____ 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Робота з літературою	20.10.2024	
2	Розробка робочого плану роботи за узгодженням з науковим керівником	25.10.2024	
3	Написання 1 розділу та представлення його керівнику	31.10.2024	
4	Написання 2 розділу та представлення його керівнику	10.11.2024	
5	Написання 3 розділу та представлення його керівнику	20.11.2024	
6	Написання 4 розділу та представлення його керівнику	29.11.2024	
7	Оформлення ПЗ	29.11.2024	
8			
9			

Студент(ка)

(підпис)

Олексій БАЛАБУХА

(прізвище та ініціали)

Керівник проекту (роботи)

(підпис)

Галина КИРИЧЕК

(прізвище та ініціали)

РЕФЕРАТ

ПЗ: 87с; 3 табл; 26 рис; 22 джерела.

ВЕБПЛАТФОРМА, ДОДАТОК, ВІ ЗАПИТ, ВІДПУСТКА, ЛІКАРНЯНИЙ, ВІДГУЛ, МАСШТАБОВАНІСТЬ, МОДУЛЬ

Об'єктом дослідження є процес реалізації системи підтримки клієнтських запитів з використанням мобільних та вебдодатків.

Предметом є моделі, методи, технології та програмні засоби для реалізації системи підтримки клієнтських запитів.

Мета дипломної роботи – проведення досліджень системи підтримки клієнтських запитів, аналіз та вибір технологій, методів та способів реалізації системи підтримки клієнтських запитів з використанням мобільних та вебдодатків.

У першому розділі проведено аналіз інформаційних систем для автоматизації клієнтських запитів, вивчено існуючі рішення, визначено вимоги до системи та напрямки дослідження.

Другий розділ присвячено розгляданню структури системи, її бізнес-логіку та сценарії роботи, а також обґрунтовано вибір сучасних технологій для забезпечення продуктивності та масштабованості.

У третьому - досліджено функціональну схему роботи, налаштування інтерфейсу веб- та мобільного додатків, а також механізми авторизованого доступу для безпечного використання системи.

В заключному, четвертому розділі досліджено аналіз функціональних можливостей системи, її конкурентних переваг, а також досліджено стабільність, швидкодію та механізми обробки запитів.

В результаті виконання дипломної роботи проаналізовано додаток, який не поступається іншим аналогам своїм простим і водночас корисним та зрозумілим функціоналом.

ABSTRACT

Explanatory note to the master's work: 87 p; 3 tables; 26 figures; 23 sources.

WEB PLATFORM, APPLICATION, CUSTOMER REQUEST, VACATION,
SICK LEAVE, DAY OFF, SCALABILITY, MODULE

The object of the research is the process of implementing a client request support system using mobile and web applications.

The subject of the research includes models, methods, technologies, and software tools for implementing a client request support system.

The aim of the thesis is to conduct research on the client request support system, analyze and select technologies, methods, and approaches for implementing such a system using mobile and web applications.

The first chapter provides an analysis of information systems for automating client requests, examines existing solutions, defines system requirements, and identifies research directions.

The second chapter focuses on the structure of the system, its business logic, and operational scenarios, as well as justifies the selection of modern technologies to ensure performance and scalability.

The third chapter explores the functional workflow, interface configuration for web and mobile applications, and mechanisms for authorized access to ensure the secure use of the system.

The final, fourth section examines the analysis of the system's functionality, its competitive advantages, and also examines stability, speed, and request processing mechanisms.

As a result of the thesis, an application was analyzed that rivals other alternatives in its simplicity while also offering useful and intuitive functionality.

ЗМІСТ

Вступ.....	8
1 Предметна область та вимоги.....	9
1.1 Інформаційні системи та автоматизація обробки клієнтських запитів.....	9
1.2 Аналіз існуючих рішень.....	12
1.3 Вимоги до системи і програмного забезпечення мобільного та вебдодатку....	22
1.4 Постановка завдань	28
1.5 Висновки до розділу 1	30
2 Вибір архітектури та компоненти системи	31
2.1 Структура та модель системи.....	31
2.2 Опис бізнес-логіки та основних сценаріїв	39
2.3 Вибір технологій.....	46
2.4 Висновки до розділу 2.....	49
3 Реалізація та впровадження системи	50
3.1 Реалізація веб- та мобільного додатків.....	50
3.2 Загальні налаштування та інтерфейс користувача	57
3.3 Функціональна схема роботи системи	61
3.4 Забезпечення авторизованого доступу	68
3.5 Висновки до розділу 3	70
4 Дослідження ефективності підтримки клієнтських запитів.....	71
4.1 Функціональні можливості додатків, порівняння з аналогами.....	71
4.2 Дослідження механізмів повідомлення про статус запитів.....	76
4.3 Аналіз швидкодії та стабільності системи	78

4.4 Висновки до розділу 4	82
Висновки.....	83
Перелік джерел посилання	85

ВСТУП

В сучасному бізнес-середовищі управління клієнтськими запитами є важливим аспектом ефективної діяльності компаній. Автоматизація цього процесу дозволяє знизити навантаження на працівників, зменшити витрати часу та підвищити якість обслуговування клієнтів. Одним з підходів до реалізації цієї задачі є створення інформаційної системи, яка забезпечує можливість оперативного реагування на потреби клієнтів та моніторинг усіх етапів обробки запитів.

Актуальність роботи зумовлена зростанням обсягів клієнтських запитів у сучасних організаціях і потребою в автоматизації їх обробки для підвищення ефективності бізнес-процесів. Використання інтегрованих мобільних та вебрішень дозволяє забезпечити доступність, зручність та високу швидкість обробки запитів у режимі реального часу, що є важливим фактором у конкурентному середовищі. Робота відкриває перспективи для подальшого розвитку систем управління клієнтськими запитами, зокрема шляхом інтеграції з іншими бізнес-системами.

Така система дозволить спростити процес управління робочими завданнями для персоналу, зокрема через функціонал відстеження робочого часу, створення запитів на відпустку чи лікарняні, а також генерацію статистики. Платформи, що включають мобільні та вебдодатки, забезпечують зручний доступ для користувачів і дозволяють взаємодіяти з системою незалежно від їхнього місця знаходження.

У роботі ми розглянемо предметну область і вимоги до інформаційної системи. Спочатку розглянемо основні поняття інформаційних систем, що забезпечують автоматизацію обробки запитів, їх вплив на підвищення ефективності бізнес-процесів та рівень задоволення клієнтів.

Таким чином, ця дипломна робота має на меті створення сучасного рішення для управління клієнтськими запитами, яке забезпечить ефективне використання ресурсів компанії, підвищить якість обслуговування клієнтів та дозволить отримувати аналітичні дані для оптимізації бізнес-процесів.

1 ПРЕДМЕТНА ОБЛАСТЬ ТА ВИМОГИ

1.1 Інформаційні системи та автоматизація обробки клієнтських запитів

Інформаційні системи стали невід'ємною частиною сучасного бізнесу, забезпечуючи автоматизацію численних процесів, включаючи обробку клієнтських запитів. За допомогою цих систем компанії можуть ефективно керувати інформаційними потоками, швидше реагувати на потреби клієнтів та надавати їм якісний сервіс.

Створення сучасних електро - обчислювальних машин (ЕОМ) дало можливість автоматизувати обробку даних у всіх галузях людської діяльності. Без систем обробки даних неможливо уявити промислові технології, управління економікою на всіх рівнях, наукові дослідження, освіту, роботу транспорту, видавничу справу тощо. Значно розширилися сфери застосування комп'ютерної техніки з появою персональних комп'ютерів і комп'ютерних мереж. Найбільш розповсюдженими системами обробки даних є інформаційні системи (ІС).

Надамо визначення терміну «інформаційна система». Інформаційна система - система підтримки й автоматизації інтелектуальних робіт - пошуку, адміністрування, експертиз й експертних оцінок або суджень, прийняття рішень, керування, розпізнавання, нагромадження знань, навчання. Це прикладний напрямок розвитку інформаційних технологій. Він спрямований на вирішення конкретних завдань керування системою і його інформаційного забезпечення. У широкому розумінні ІС - це будь-яка система обробки інформації.

Основні функції, які має виконувати інформаційна система для виконання призначених для них задач, пов'язані з підтримкою динамічної інформаційної моделі предметної області та забезпеченням інформаційних потреб її користувачів. До цих функцій відносяться [2]:

- обчислювальна - вчасно і якісно виконує обробку інформації в усіх аспектах, що цікавлять систему управління;

- сліdkувальна - відстежує і формує всю необхідну для управління зовнішню та внутрішню інформацію;

- запам'ятовувальна - забезпечує безупинне накопичення, систематизацію, збереження і відновлення всієї необхідної інформації;

- комунікаційна - забезпечує передачу потрібної інформації в задані пункти;

- інформаційна - реалізує швидкий доступ, пошук і видачу необхідної інформації;

- регулювальна - здійснює інформаційно-керуючий вплив на об'єкт управління і його підрозділ у випадку відхилення їх параметрів функціонування;

- документувальна - забезпечує формування всіх обліково-звітних, планово-розпорядчих, конструкторсько-технологічних та інших форм документів.

За цільовим призначенням інформаційні системи діляться на: керуючі, інформаційно-довідкові й допоміжні (рис. 1.1).

Керуючі ІС - це системи керування, що мають, як правило, складну структуру і призначені для керування роботою системи.

Інформаційно-довідкові ІС - це системи, які забезпечують обслуговування певного кола спеціалістів, або певні керуючі системи. Вони можуть містити досить великі обсяги інформації. Розміщуються на одному комп'ютері або можуть бути розподілені на ряді комп'ютерів.

Допоміжні ІС - це системи, призначені для підтримки ухвалення рішення [1].



Рисунок 1.1 Типи інформаційних систем цільовим призначенням

У більш вузькому розумінні інформаційні системи - це сукупність апаратних і програмних засобів, які використовуються для рішення конкретного прикладного завдання.

Розглянемо поняття системи підтримки клієнтських запитів (СПКЗ). Системою підтримки клієнтських запитів називається комплекс, що включає обчислювальне і комунікаційне обладнання, програмне забезпечення, лінгвістичні засоби і інформаційні ресурси, а також системний персонал. Цей комплекс забезпечує підтримку динамічної інформаційної моделі деякої частини реального світу для забезпечення інформаційних потреб користувачів. Частина реального світу, яка моделюється інформаційною системою, називається її предметною областю [1].

Система підтримки - система, що реалізує інформаційні технології у сфері управління при сумісній роботі управлінського персоналу та комплексу технічних засобів.

Варто зауважити, що мета створення СПКЗ - підвищення ефективності управління бізнесом підприємства. СПКЗ реалізують типові функції управління: прогнозування, планування, облік, контроль, аналіз, регулювання. СПКЗ призначена для автоматизованого збирання, реєстрації, збереження, пошуку, оброблення та видачі інформації за запитом користувачів.

СПКЗ складається з сукупності взаємозв'язаних компонентів: технічних засобів обробки і передачі даних (засобів обчислювальної техніки і зв'язку), методів і алгоритмів обробки у вигляді відповідного програмного забезпечення, масивів (наборів, баз) даних на різних носіях, персоналу і користувачів, об'єднаних за організаційно-структурними, тематичними, технологічними чи іншими ознаками для виконання автоматизованої обробки даних з метою задоволення інформаційних потреб споживачів інформації. При цьому визначальним є принцип орієнтації розробленого технічного, програмного, інформаційного забезпечення на потреби конкретних користувачів. СПКЗ є людино-машинною системою, вона дає змогу підвищити якість управління завдяки оптимальному розподілу праці між людиною та комп'ютером на всіх стадіях управління. Уся сукупність операцій

оброблення інформації, що включає збирання, введення, запис, реєстрацію, перетворення, зчитування, збереження, знищення, коригування, обмін за каналами зв'язку, в СПКЗ здійснюється за допомогою технічних і програмних засобів.

Розглянемо поняття автоматизованої інформаційної системи. Автоматизована інформаційна система - це взаємозв'язана сукупність даних, обладнання, програмних засобів, персоналу, стандартних процедур, які призначені для збору, обробки, розподілу, зберігання, представлення інформації у відповідності з вимогами, які випливають з цілей організації.

Комплексна автоматизація підприємства має на увазі перекладення у площину комп'ютерних технологій всіх основних ділових процесів організації. Використання спеціальних програмних засобів, що забезпечують інформаційну підтримку бізнес-процесів, вважається найбільш виправданим та ефективним. Сучасні системи керування діловими процесами дозволяють інтегрувати навколо себе різне програмне забезпечення, формуючи єдину інформаційну систему.

Також керівництво одержує своєчасний доступ до достовірних даних про хід виробничого процесу, має засоби для оперативного прийняття і втілення у життя своїх рішень. І що найголовніше, отриманий автоматизований комплекс являє собою гнучку відкриту структуру, яку можна перебудовувати і доповнювати новими модулями або зовнішнім програмним забезпеченням [3].

Таким чином досліджено, що автоматизація процесів обробки запитів сприяє зниженню людських помилок, зменшенню витрат часу та забезпеченню більшої точності обробки інформації.

1.2 Аналіз існуючих рішень

Розглянемо існуючі інформаційні системи (ІС). Аналіз ІС важливим етапом у процесі впровадження технологій у бізнес, оскільки вибір відповідної системи може суттєво вплинути на ефективність роботи компанії, оптимізацію бізнес-

процесів та якість обслуговування клієнтів. Сьогодні інформаційні системи поділяються на різні типи залежно від їхнього призначення, особливостей і сфери використання.

Залежно від галузі та бізнес - цілей компанії вибір інформаційної системи може суттєво відрізнятись. Наприклад, для роздрібного бізнесу критично важливими є операційні системи обробки даних та CRM - системи, оскільки вони дозволяють оперативно обробляти замовлення та ефективно взаємодіяти з клієнтами. Водночас для виробничих компаній важливим фактором є наявність систем управління ланцюгами постачань (SCM) та систем підтримки прийняття рішень (DSS), які дозволяють оптимізувати виробничі процеси, уникати перебоїв у поставках та ефективно розподіляти ресурси.

Розглянемо формулювання дефініції HRM-системи, які пропонують провідні дослідники цього напрямку. Армстронг М. розглядає HRM-системи як такі, що включають в себе стратегії, процедури і практики, які визначають, як організація керуватиме своїми працівниками. Він наголошує на тому, що HRM-системи мають бути адаптивними до змін в організації і зовнішнього середовища.

Дуглас МакГрегор - автор теорій Х та Y - зазначає, що HRM системи повинні бути спрямовані на розвиток і задоволення потреб працівників, а не просто на контроль за ними [5]. Він також наголошує на важливості довіряти працівникам та надавати їм більше відповідальності в прийнятті рішень, що стосуються їх роботи. Це може позитивно вплинути на продуктивність та мотивацію працівників.

Петер Капеллі досліджує сучасні тенденції управління талантами та розвитку персоналу, звертаючи увагу на гнучкість та адаптабельність компаній до змін на ринку праці [13]. Джон Бордо розглядає HR як фактор, що впливає на стратегію та результативність компанії, та розвиває методики оцінювання та визначення ефективності HR-функції [14].

Для автоматизації управління персоналом на підприємствах використовуються інформаційні системи, що зазвичай позначаються аббревіатурою HRMS (Human Resources Management System - системи управління людськими ресурсами). У найповнішому варіанті такі системи охоплюють всі рівні управління

підприємством – операційний, тактичний та стратегічний, а також функціональний розріз - кадровий облік, розрахунки з персоналом та систему управління трудовими ресурсами. Система включає модулі для найму та підбору персоналу, оцінювання, навчання, розвитку та мотивації персоналу.

Сьогодні на ринку інформаційних систем, що спроможні покращити управління підприємством, є широкий вибір різноманітних ПЗ, серед яких слід зазначити «SAP ERP», «Bitrix24», «Oracle BPM Suite», «Comindware Business Application Platform», «HURMA System», «Scala HR» та ін. Разом із тим, недостатня обізнаність керівників підприємств у можливостях, що перед ними відкриваються, у разі застосування таких інформаційних систем, а також брак коштів на їх придбання, позбавляють суб'єкти господарювання отримати необхідні конкурентні переваги засобами сучасних ІТ-технологій. Вивчення функціональних можливостей поширених на ринку ПЗ, аналіз їх переваг та недоліків є вельми актуальним для покращення роботи сучасних підприємств [5].

Розглянемо можливості найбільш застосовуваних на вітчизняних підприємствах інформаційних систем [7, 8].

«1С: Підприємство». Цей інформаційний засіб призначений для автоматизації основних фінансових процесів. Дозволяє розрахувати базові бухгалтерські показники, Досить складне налаштування, проте доволі значний функціонал, а саме ведення бухгалтерського обліку, проведення фінансових операцій, нарахування заробітної плати та підрахунок податків. Також дана конфігурація має декілька версій: для малих фірм, складів, великих компаній.

«SAP ERP» [11] є багатокомпонентною програмою, що містить у собі основні бізнес-функції організації. Дане ПЗ містить 3 складові: клієнт, сервер та СУБД. Така ІС оптимізує процеси фінансового обліку, логістики, ведення обліку кадрів та взаємодію з партнерами. Кожна конфігурація включає більш розширені функції для певного типу організації.

«Oracle BPM Suite» - зручна CRM система, яка містить кілька ланок:

- взаємодія з контрагентами, клієнтами та постачальниками;
- внутрішні процеси;

- електронна комерція.

Фактично дана програма націлена на 3 напрямки, разом із тим, дозволяє управління персоналом, логістикою, фінансами та активами фірми.

«Bitrix24» - хмарний сервіс, який являє собою справжню корпоративну мережу, що містить власну CRM систему та месенджер. Містить великий спектр різних інструментів та шаблонів. Має такі функції, як налаштування логістичних процесів, аналіз залишків ресурсів.

«Comindware Business Application Platform» - комп'ютерна програма компанії «Comindware», яка у 2020 році, стала найкращим розробником ІТ-програм. Дане ПЗ допомагає автоматизувати управління персоналом, транспортну логістику, прибуття та відвантаження товарів, має свою власну CRM систему, є простим у використанні.

«Scala HR» - є продуктивною HRMS [9], що є повноцінною системою управління людськими ресурсами та охоплює багато функцій, таких як кадровий облік, відпустки, оплата праці, навчання та розвиток, оцінка працівників. Має гнучку конфігурацію, що дозволяє користувачам налаштовувати систему відповідно до своїх потреб та бізнес-процесів та гарний інтерфейс користувача, що дозволяє легко орієнтуватися в системі та швидко здійснювати потрібні дії. Проте варто зазначити, що «Scala HR» може бути складною для вивчення та налаштування для некваліфікованого персоналу, що може затримувати впровадження системи. Також може бути дорожчою, порівняно з іншими системами управління людськими ресурсами на ринку.

Проаналізуємо вітчизняну систему управління персоналом «HURMA System» [10]. Це міжнародна компанія, яка спеціалізується на розробленні системи для автоматизації процесів HR, рекрутингу та управління цілями компанії. Система дозволяє заощадити до 43% часу на рутинних процесах і 18% бюджету на рекрутингу. Компанія представлена в країнах східної Європи та СНД. Офіс знаходиться у м. Харків.

HURMA System - це комплексне рішення для HR, рекрутингу та OKR в одній системі. Дана система автоматизації включає функціонал, який закрийє всі завдання

від першого контакту з кандидатом, проходження всіх етапів рекрутингу до переведення його в співробітники, онбордингу, адаптації, утримання і навіть моніторингу настрою.

До основних можливостей цієї системи належать наявність таких функцій: онбординг і адаптація нових співробітників, ведення бази даних персоналу, журнал подій із нагадуваннями, опитування та оцінка ефективності, корпоративний календар, інструменти для відстеження співробітників, інтеграція чат-бота, HR-аналітика, а також сценарії для автоматизації процесів онбордингу та звільнення. Система також підтримує автоматизацію таких завдань, як облік відсутності працівників, переробки, відрядження та інші робочі події, а також ведення обліку переробок, робочого часу і розрахунку зарплат через Payroll.

Проаналізувавши переваги та недоліки даної системи, можна зазначити, що до переваг відносяться: рекрутингові інструменти: база кандидатів і вакансій для рекрутера, інтеграція з локальними job-порталами і LinkedIn; автоматизація Welcome interview, Adaptation, Probation, Exit interview; зручне дерево компанії; статистика по HR і рекрутингу; календар подій, який синхронізується з Google-календарем і відправляє повідомлення про заходи в компанії; особистий профіль співробітника компанії з можливістю відправляти запити на відпустку, лікарняний.

До недоліків слід віднести недоступні графіки по філіях і Payroll знаходиться ще на стадії розробки.

Наведемо нижче переваги та недоліки деяких з вищеописаних інформаційних систем, що оптимізують роботу підприємства за різними критеріями (табл. 1. 1) [5].

Таблиця 1.1 - Аналіз ПЗ, що оптимізують роботу підприємства

Назва ПЗ	Переваги ПЗ	Недоліки ПЗ
1С: Підприємство	1) існують різні конфігурації для різних типів бізнесу; 2) потужний функціонал системи; 3) можливість працювати одночасно з великою кількістю користувачів;	1) потрібні незначні навички програмування; 2) високовартісні сервісні послуги на тривалий період.

Продовження таблиці 1.1

Назва ПЗ	Переваги ПЗ	Недоліки ПЗ
1С: Підприємство	4) контроль за діями користувачів.	
SAP ERP	1) інтеграція з іншими ПЗ; 2) індивідуальний інтерфейс; 3) для кожного користувача можливо налаштувати його певні функції, які йому доступні.	1) потрібні певні навички та знання в певних кодах та моделях; 2) для додавання додаткових функцій, необхідна доплата за кожну із них.
Oracle BPM Suite	1) різні шаблони для зручної роботи; 2) створення корпоративних порталів; 3) доступ до усіх ланок управління: від головного директора до рядового менеджера.	1) слабкий функціонал для великих підприємств; 2) відсутність роботи з різними функціями водночас.
Bitrix24	1) налаштування під усі види бізнесу; 2) вбудована IP-телефонія; 3) відслідковування усіх дій працівників та коментування їх роботи;	1) багато платних тарифів, без яких функціонал є слабким.
Comindware Business Application Platform	1) значний перелік наявних інструментів; 2) 84 готових шаблона, що не потребують багато часу для застосування на підприємстві; 3) проектування у графічному редакторі.	1) є пробна версія, проте після її закінчення досить велика ціна (для кожного клієнта індивідуально); 2) шаблони можуть і не підійти під певний тип бізнесу

Сучасні технології значно спрощують рутинні процеси в управлінні персоналом, пропонуючи зручне програмне забезпечення як для HR-менеджерів, так і для співробітників компаній. Завдяки автоматизації HR-функцій стає можливим ефективний розподіл завдань із управління інформацією через різні рівні систем управління персоналом.

Це сприяє:

- оптимізації управління заробітною платою, наймом і адаптацією нових працівників;
- збору, зберіганню та швидкому доступу до інформації про співробітників;
- точному відстеженню робочого часу і контролю прогулів;
- якісній оцінці продуктивності персоналу;
- виконанню адміністративних завдань;
- організації та контролю процесу навчання;
- наданню співробітникам інструментів для самообслуговування;
- плануванню завдань і графіків роботи;
- проведенню HR-аналітики для обґрунтованого прийняття рішень;
- підвищенню рівня безпеки HRMS.

Цей підхід робить управління кадрами більш прозорим і ефективним, одночасно покращуючи умови для продуктивної роботи всієї команди.

Нами розглянуто існуючі рішення, але невід'ємною частиною даних реалізацій є їх розробка. В наш час багато підходів щодо створення додатків. Наприклад використовувати фреймворк або бібліотеку в реалізації. Це два різних підходи до роботи з кодом, і вони мають свої особливості та застосування в розробці. Попри те, що ці терміни часто використовують разом, вони виконують різні функції та мають різний ступінь впливу на структуру вашого коду. Бібліотека - це набір готових функцій, класів або модулів, які можна використовувати для вирішення конкретних завдань у коді. При використанні бібліотеки, то розробник сам контролює, коли і як її частини включаються у проект, тобто викликає функції або методи бібліотеки, коли вони потрібні. У бібліотеці немає суворої структури чи контролю над програмою – бібліотека використовується як додатковий інструмент для полегшення роботи. Фреймворк працює інакше - це більш масштабний інструмент, який надає основну структуру та набір правил для побудови додатка. Використовуючи фреймворк, розробник підлаштовується під його структуру, адже фреймворк сам керує життєвим циклом додатка і викликає

код в певних місцях. Можна сказати, що у фреймворку розробник підлаштовується під його правила, тоді як з бібліотекою сам визначає, як її використовувати.

Розглянемо декілька прикладів бібліотек та фреймворків.

React – це одна з найпопулярніших JavaScript-бібліотек для створення інтерфейсів користувача. Вона дозволяє створювати компоненти, що можуть повторно використовуватись у різних частинах програми. React підходить як для розробки веб-додатків, так і для гібридних мобільних додатків у поєднанні з іншими технологіями[16]. Однією з найважливіших особливостей React є концепція "віртуального DOM" (Document Object Model). Віртуальний DOM дозволяє React оновлювати лише ті частини інтерфейсу, які змінилися, що значно підвищує продуктивність додатка, особливо для складних інтерфейсів. Крім того, React забезпечує підтримку компонентного підходу, що полегшує управління інтерфейсом, особливо коли додаток розширюється. Завдяки своїй популярності, React має величезну екосистему бібліотек та інструментів, що підтримують роботу з маршрутизацією, станом (як-от Redux), а також розширення можливостей, наприклад, для тестування..

Vue.js - це фреймворк, який працює на JavaScript, створений для розробки користувацьких інтерфейсів. Він працює на базі звичайного HTML, CSS та JavaScript, з можливостями декларативно програмувати користувацькі інтерфейси будь-якої складності на основі компонентів[17]. Vue добре підходить для побудови SPA-додатків (Single-Page Applications) і має низку розширень, що робить його гнучким для різних проектів. Vue також забезпечує двонаправлене зв'язування даних, що спрощує синхронізацію стану інтерфейсу з даними.

Angular – це універсальне інтерфейсне середовище розробки з шаблоном для проектування MVC, побудована на TypeScript з яким можна писати продуктивні додатки з мінімумом помилок[18]. Angular один JavaScript-фреймворк, що визначає сувору структуру проекту: контролери, компоненти, служби тощо. Він надає стандарти для організації файлів, методи для роботи з даними, маршрутизацію та інше. Angular вирішує, коли і які частини вашого коду

виконуються, а розробник повинен дотримуватися цих правил, щоб додаток працював належним чином.

React Native є фреймворком, побудованим на основі React для створення мобільних додатків. Використовуючи JavaScript, React Native дозволяє розробникам створювати додатки для Android та iOS з єдиною базою коду, зберігаючи відчуття нативності[19]. Це зменшує час і витрати на розробку, адже немає потреби писати окремий код для кожної платформи. React Native особливо популярний для швидкого створення мобільних додатків, які виглядають і працюють як нативні. Також фреймворк дозволяє підключати нативні модулі, що робить можливим роботу з апаратним забезпеченням телефону.

Критично важливим елементом будь-якого додатку є бази даних, які потрібні для зберігання, обробки та управління даними додатків. Вибір відповідної бази даних залежить від типу і масштабу додатка, а також вимог до продуктивності.

Firebase — це хмарна платформа від Google, яка пропонує базу даних реального часу, що базується на NoSQL-підході. Firebase часто використовується для мобільних додатків через легкість інтеграції та велику кількість інструментів, що входять до платформи. Вона підтримує синхронізацію даних у реальному часі, що дозволяє відображати зміни даних на інтерфейсі відразу після їх внесення. Firebase також пропонує модулі для аутентифікації, хостингу, аналітики та інші, що робить його популярним вибором для невеликих і середніх проектів. Однак, для великих додатків Firebase може бути дорогим та обмеженим у складних запитах[20].

Іншою поширеною NoSQL базою даних є MongoDB, яка зберігає дані в JSON-подібних документах, що робить її гнучкою для роботи з великими обсягами неструктурованих даних. MongoDB чудово підходить для веб-додатків, які мають змінні структури даних, оскільки її модель дозволяє легко додавати нові поля або змінювати структуру без потреби переналаштування. База даних підтримує горизонтальне масштабування, що робить її оптимальною для великих проектів[21].

Для зберігання структурованих даних та реалізації складних транзакцій підходить PostgreSQL — потужна реляційна база даних з підтримкою SQL-запитів. PostgreSQL популярна серед розробників через її стабільність, надійність і підтримку складних типів даних, таких як JSON, XML тощо. PostgreSQL гарантує збереження даних і надає інструменти для забезпечення цілісності даних, тому її часто використовують для додатків, які працюють з фінансами, бізнес-аналітикою та іншими критично важливими даними.

Для створення привабливих та адаптивних інтерфейсів використовують різні бібліотеки для стилізації, що дозволяють швидко і гнучко працювати з CSS. Material-UI (MUI) — це бібліотека компонентів для React, що базується на концепціях Material Design від Google. Вона забезпечує розробників широким набором готових компонентів, які можна легко інтегрувати у проект, налаштовуючи під власний стиль. Material-UI добре підходить для корпоративних веб-додатків, адже він забезпечує консистентність інтерфейсу та дотримується сучасних стандартів дизайну. Іншою популярною бібліотекою для стилізації є Styled-Components, яка дозволяє писати CSS безпосередньо у файлах JavaScript. Вона надає можливість динамічно змінювати стилі компонентів, що особливо зручно для побудови інтерфейсів з використанням умовних стилів. Styled-Components працює за допомогою JavaScript-темізації, що полегшує управління стилями в додатку. Для тих, хто хоче отримати більший контроль над CSS, чудовим варіантом є Tailwind CSS. Це утилітарна бібліотека, яка пропонує набір класів для створення адаптивних інтерфейсів. Tailwind дозволяє швидко стилізувати компоненти, не створюючи окремі CSS-файли, що пришвидшує процес розробки. Проте, для роботи з Tailwind потрібно звикнути до її структури та принципів використання утилітарних класів.

Таким чином, провівши аналіз існуючих рішень, варто зауважити, що незважаючи на очевидні переваги інформаційних систем, їх впровадження може супроводжуватись низкою викликів, включаючи високі витрати, складність інтеграції, а також потребу в спеціалізованих навичках та навчанні персоналу.

Варто зауважити, що перспективи розвитку інформаційних систем полягають у подальшій інтеграції з технологіями штучного інтелекту, автоматизації, великих даних та хмарних рішень. Хмарні інформаційні системи вже стали стандартом для багатьох компаній, оскільки вони забезпечують доступ до даних з будь-якої точки світу, високу масштабованість та економію на інфраструктурі. Наприклад, SAP, Oracle та інші провідні постачальники програмного забезпечення активно розвивають свої хмарні продукти, що дозволяє бізнесам швидше адаптуватися до змінних ринкових умов і ефективніше обробляти дані.

1.3 Вимоги до системи і програмного забезпечення мобільного та вебдодатку

У сучасному світі мобільні та вебдодатки стали незамінною частиною бізнесу та щоденного життя. Вони дозволяють компаніям забезпечувати клієнтів якісними послугами, збільшувати ефективність роботи співробітників, а також створювати нові можливості для взаємодії. Однак успішність і стабільна робота таких додатків залежать від чіткого визначення вимог до системи та програмного забезпечення на етапі розробки. Правильно сформульовані вимоги допомагають створити програмний продукт, який відповідає очікуванням користувачів, задовольняє бізнес-потреби та витримує технічні навантаження.

Варто зазначити, для того щоб почати користуватися системою візуального бронювання нам необхідний будь-який встановлений браузер (наша система кросбраузерна, тому підійде будь-який браузер). Система повинна легко і швидко працювати на різних платформах.

Мінімальні системні вимоги для користування веб-додатком можуть змінюватися в залежності від конкретного додатку та його функцій. Однак, зазвичай, основні системні вимоги для користування веб-додатком включають:

- операційна система: Веб-додатки можуть працювати на будь-якій сучасній операційній системі, такі як Windows, macOS, Linux, Android або iOS;
- веб-браузер: Веб-додатки зазвичай працюють в будь-якому сучасному веб-браузері, такому як Google Chrome, Mozilla Firefox, Safari, або Microsoft Edge;
- швидкість Інтернет-з'єднання: Веб-додатки можуть працювати з будь-якої швидкості Інтернет-з'єднання, але, зазвичай, мінімальною швидкістю рекомендується 1 Мбіт/с.

Загальні системні вимоги для використання нашого додатку дуже низькі, і більшість комп'ютерів, що працюють на сучасних операційних системах, зможуть запустити вебплатформу без проблем. Однак, є мінімальні вимоги для кращої продуктивності та користування функціями, які пропонує Google.

Наведемо нижче мінімальні системні вимоги для використання нашого додатку на Windows, Mac та Linux (табл. 1.2).

Таблиця 1.2- Вимоги до різних операційних систем

Операційна система	Процесор	Вільне місце на диску	Оперативна пам'ять
Windows 7 або новіше	Intel Pentium 4 або вище, AMD Athlon 64 або вище	2ГБ	2ГБ
Mac OS X 10.10 або новіше	Intel-процесор / M1 або новіше	2ГБ	2ГБ
Linux (будь-який дистрибутив)	Intel Pentium 4 або вище, AMD Athlon 64 або вище	2ГБ	2ГБ

Важливо відзначити, що це лише мінімальні вимоги. Якщо є більше вільного місця на диску та більше оперативної пам'яті, буде змога працювати з більшим комфортом та швидкістю. Також, наявність більш потужного процесора може покращити швидкість завантаження сторінок та відповіді на запити.

Щоб забезпечити безперебійну роботу мобільного додатка для системи підтримки клієнтських запитів, необхідно врахувати певні вимоги до операційної

системи пристроїв. Ці вимоги дозволять гарантувати сумісність додатка, його стабільну роботу, безпечне зберігання та обробку даних. Для стабільної роботи додатка важливо обрати версії ОС, що підтримують усі необхідні функції для обробки даних, мережевого з'єднання та інтеграції з іншими сервісами. Додаток повинен працювати на пристроях із сучасними процесорами, щоб забезпечити швидкість обробки даних та плавність інтерфейсу. Щоб система функціонувала без затримок і швидко перемикалася між екранами, бажано мати не менше 2 ГБ оперативної пам'яті для обох платформ. Для зберігання кешованих даних і локального зберігання користувацьких даних, додаток потребує вільного місця на пристрої.

Нижче, у таблиці 1.3, зазначені системні вимоги для використання мобільного додатку на платформах iOS та Android.

Таблиця 1.3 - Вимоги до iOS та Android

Параметр	Вимоги для iOS	Вимоги для Android
Версія ОС	iOS 13 або новіше	Android 9 (Pie)
Процесор	A9 (iPhone 6S і вище)	4 ядра (Snapdragon 400 або вище)
Оперативна пам'ять	2 ГБ або більше	2 ГБ або більше
Вільне місце для зберігання	100 МБ	100 МБ
Мережеве з'єднання	Wi-Fi, 4G і вище	Wi-Fi, 4G і вище
Доступ до служб пристрою	GPS, календар, Push-сповіщення, біометрія (Face ID, Touch ID)	GPS, календар, Push-сповіщення, біометрія (відбиток пальця)
Безпека	Шифрування, Face ID або Touch ID	Шифрування, біометрія (сканер відбитків)
Сумісність з SDK	Остання версія SDK iOS	Остання версія SDK Android

Розглянемо детально вимоги до системи і програмного забезпечення:

а) функціональні вимоги. Функціональні вимоги визначають конкретні функції та можливості, які повинні бути реалізовані в системі, щоб вона відповідала потребам користувачів. Основні функціональні вимоги для мобільних та вебдодатків можуть включати:

1) реєстрація та аутентифікація користувачів - додаток повинен підтримувати можливість створення облікових записів, вхід за допомогою логіна та пароля;

2) ролі доступу для різного типу користувачів – в кожній компанії є начальник, чи глава відділу, та підлеглі, тож в додатку повинно бути як мінімум дві ролі користувача (Admin/User), в одна роль має необмежені права доступу, в іншого урізані;

3) сповіщення - додаток має надсилати повідомлення про нові події або оновлення. Це можуть бути пуш-повідомлення або електронні листи, залежно від налаштувань користувача;

4) пошук і фільтрація - можливість швидкого пошуку та фільтрації даних за різними параметрами, щоб користувач міг легко знайти потрібну інформацію;

б) нефункціональні вимоги. Нефункціональні вимоги описують характеристики, які забезпечують належну якість роботи системи. Вони охоплюють наступне:

1) продуктивність - додаток повинен швидко завантажуватись і працювати без затримок, особливо при великому навантаженні. Оптимізація продуктивності особливо важлива для мобільних додатків, де ресурси обмежені;

2) безпека - важливо, щоб додаток був захищений від несанкціонованого доступу, включав шифрування даних, а також мав засоби захисту від кібератак (наприклад, захист від SQL-ін'єкцій і міжсайтового скриптингу для вебдодатків);

3) масштабованість - система повинна мати можливість розширюватись та підтримувати зростання кількості користувачів, даних та функцій;

4) сумісність - додаток повинен працювати на різних операційних системах (Android, iOS для мобільних додатків; усі сучасні браузері для вебдодатків);

в) технічні вимоги. Технічні вимоги описують інфраструктуру, на якій працює додаток, та інші технічні аспекти. Це включає:

1) вибір технологій - для вебдодатка це можуть бути фреймворки, такі як React, Angular чи Vue для фронтенду, та Node.js, Python, або Java для бекенду. Для мобільного додатка можуть використовуватись рідні мови програмування (Swift для iOS, Kotlin для Android) або кросплатформенні рішення (React Native, Flutter);

2) архітектура системи - зазвичай сучасні додатки мають багаторівневу архітектуру, яка включає клієнтську частину, серверну частину, базу даних та інші компоненти, як-от мікросервіси для розподіленої обробки даних;

3) база даних - вибір СУБД залежить від обсягу та типу даних. Для додатків, які потребують швидкого доступу до великих обсягів структурованих даних, підходить SQL (наприклад, PostgreSQL, MySQL); для неструктурованих – NoSQL рішення (MongoDB, Cassandra);

г) інтерфейс користувача та досвід користувача (UX/UI). Важливими аспектами будь-якого мобільного або вебдодатку є зручність та привабливість інтерфейсу. Основні вимоги до UX/UI включають:

1) адаптивний дизайн – цей підпункт стосується лише вебдодатку, але це не обов'язкова вимога, адже в більшості випадках додатки такого роду використовуються лише на комп'ютері, бо це набагато зручніше та швидше моніторити все через комп'ютер або ноутбук. Але, якщо інтерфейс підлаштовуватиметься також під планшети та мобільні девайси, додаток від цього отримає додатковий плюс;

2) простота навігації - забезпечення інтуїтивної навігації, де користувачі швидко знайдуть основні функції без зайвих зусиль;

3) використання візуальних елементів - додаток має використовувати зрозумілі візуальні елементи (іконки, кольорові схеми), що полегшує сприйняття інформації;

г) тестування та забезпечення якості. Якість програмного забезпечення має бути підтверджена шляхом проведення різних видів тестування:

1) функціональне тестування - перевірка, чи працюють усі функції системи відповідно до вимог;

2) навантажувальне тестування - оцінка продуктивності системи під великим навантаженням для запобігання збоєм у разі одночасного використання багатьма користувачами;

3) кросплатформене тестування - перевірка, чи правильно працює додаток на різних платформах і пристроях;

4) безпекове тестування - виявлення потенційних вразливостей у додатку, таких як SQL-ін'єкції чи міжсайтовий скриптинг;

д) підтримка та обслуговування. Після запуску додатка його необхідно підтримувати та оновлювати для забезпечення стабільної роботи. Це включає:

1) регулярні оновлення - додавання нових функцій, виправлення помилок, покращення безпеки;

2) технічна підтримка користувачів - наявність команди підтримки, яка може швидко реагувати на запити користувачів та допомагати у вирішенні проблем;

3) збір та аналіз відгуків користувачів - збір зворотного зв'язку для постійного покращення додатка та адаптації до потреб користувачів.

Таким чином, визначення вимог до системи та програмного забезпечення мобільних і вебдодатків є ключовим кроком у їхньому проектуванні та впровадженні. Чітко прописані функціональні та нефункціональні вимоги допомагають уникнути помилок під час розробки, підвищити зручність і ефективність додатка, а також забезпечити його безперебійну роботу та безпеку. Дотримання технічних вимог і тестування всіх компонентів гарантує, що продукт проявить себе стійким до навантажень та відповідатиме вимогам сучасного ринку.

Постійна підтримка та оновлення додатка сприяють його адаптації до змінних умов та потреб користувачів, що в цілому покращує його конкурентоспроможність та довговічність.

1.4 Постановка завдань

Виходячи з мети роботи, проведеного аналізу предметної області та існуючих рішень, наведемо перелік завдань, які потрібно вирішити:

а) розглянути структуру та модель наших додатків - у цьому підрозділі розглянемо архітектуру системи та пояснимо як вона структурована. Наведемо схему архітектури, вкажемо основні компоненти системи: фронтенд (веб та мобільний додаток), бекенд, бази даних, системи автентифікації та авторизації, зовнішні сервіси та API. Важливо пояснити, як ці компоненти взаємодіють між собою і як забезпечується безперебійна робота системи загалом;

б) описати бізнес-логіку – тут описано основні сценарії взаємодії користувача із системою, які визначають її бізнес-логіку. Наприклад:

1) створення запиту: працівник надсилає запит через форму і система призначає відповідального оператора;

2) обробка запиту: запит автоматично розподіляється за категоріями, вирушає виконання, і адміністратор може відстежувати статус виконання;

3) закриття запиту: після завершення запиту працівник отримує повідомлення і запит закривається, але зберігається в базі даних для можливого подальшого аналізу або повторного звернення;

в) проаналізувати вибір технологій - у цій частині перерахуємо та опишемо технології за допомогою яких була створена система. Це є важливим фактором, адже в наш час є безліч різних технологій за допомогою яких можна розробити вебплатформи або мобільні додатки, тож ми розглянемо переваги обраних технологій в порівнянні з іншими аналогами, що доведе що наша система якісна та швидка платформа;

г) коротко розглянути реалізацію веб- та мобільного додатків - тут опишемо процес реалізації веб- та мобільних додатків, включаючи основні етапи розробки, використовувані підходи та адаптацію функціоналу для різних платформ

(наприклад, створення окремих компонентів для мобільних та веб-версій), а також принципи проектування (наприклад, мікросервісну або монолітну архітектуру);

д) розглянути базові налаштування інтерфейсу та параметри конфігурації, такі як мова, регіональні налаштування, адаптивність інтерфейсу, а також деталі роботи з візуальними компонентами. Тут можна розглянути, як в інтерфейсі організований доступ до ключових функцій і як вони забезпечують інтуїтивність роботи користувача;

е) побудувати функціональна схема роботи системи - послідовність кроків обробки запиту, ролі різних компонентів у процесі і схема, яка описує основні потоки даних у системі. Слід також описати, як система обробляє запити від користувачів та передає дані між фронтенд- та бекенд-частинами;

є) забезпечити авторизований доступ - у цьому підрозділі необхідно описати механізми автентифікації та авторизації користувачів, таких як використання JWT, OAuth або інших методів. Також важливо вказати, як система захищає дані користувачів та запобігає несанкціонованому доступу, які ролі існують і які права вони надають користувачам;

ж) розглянути функціональні можливості додатків – тут ми опишемо роботу ключових функцій веб- та мобільних додатків, детально розглянемо особливості кожної сторінки та скріна і покажемо переваги над іншими системами;

з) дослідити механізми повідомлення про статус запитів - у цьому розділі проаналізуємо механізми сповіщення користувачів про статус їх запитів. Тут опишемо методи (повідомлення, email-розсилки, push-повідомлення) та особливості їх реалізації. Важливо оцінити, наскільки дані сповіщення зручні для користувачів і чи надають вони своєчасну та актуальну інформацію;

і) провести аналіз швидкодії та стабільності системи – у останньому підрозділі ми опишемо продуктивність та стабільність системи, включаючи час завантаження, відгуку та надійність при високих навантаженнях. Тут проведені тести, щоб визначити максимальне навантаження, яке система здатна витримати, а також виявити слабкі місця у продуктивності.

Отже, ми визначили основні цілі та задачі, які повинні бути досягнуті в ході дипломної роботи. Основна задача полягає у створенні ефективної моделі взаємодії з користувачами, що забезпечує своєчасну обробку запитів, зручний інтерфейс і стабільну роботу на різних платформах. У рамках дослідження також визначено критерії для подальшої оцінки продуктивності та якості роботи системи, які дозволять виявити її сильні та слабкі сторони, а також напрямки для вдосконалення.

1.5 Висновки до розділу 1

Таким чином автоматизація процесів обробки запитів сприяє зниженню людських помилок, зменшенню витрат часу та забезпеченню більшої точності обробки інформації.

Незважаючи на очевидні переваги інформаційних систем, їх впровадження може супроводжуватись низкою викликів, включаючи високі витрати, складність інтеграції, а також потребу в спеціалізованих навичках та навчанні персоналу.

Визначення вимог до системи та програмного забезпечення мобільних і вебдодатків є ключовим кроком у їхньому проектуванні та впровадженні. Чітко прописані функціональні та нефункціональні вимоги допомагають уникнути помилок під час розробки, підвищити зручність і ефективність додатка, а також забезпечити його безперебійну роботу та безпеку. Дотримання технічних вимог і тестування всіх компонентів гарантує, що продукт проявить себе стійким стійким до навантажень та відповідатиме вимогам сучасного ринку.

Також було визначено основні цілі та задачі, які повинні бути досягнуті в ході дипломної роботи. Основна задача полягає у створенні ефективної моделі взаємодії з користувачами, що забезпечує своєчасну обробку запитів, зручний інтерфейс і стабільну роботу на різних платформах. У рамках дослідження було визначено критерії для подальшої оцінки продуктивності та якості роботи системи, які

дозволять виявити її сильні та слабкі сторони, а також напрямки для вдосконалення.

2 ВИБІР АРХІТЕКТУРИ ТА КОМПОНЕНТИ СИСТЕМИ

2.1 Структура та модель системи

Спираючись на технічні вимоги до системи, ми повинні обрати архітектурний підхід для нашого додатку. Вибір архітектурного підходу є одним із ключових етапів розробки програмного забезпечення, оскільки саме він визначає, як система буде функціонувати, масштабуватися і взаємодіяти з іншими компонентами та сервісами. Правильно підібрана архітектура дозволяє забезпечити стабільну роботу системи під навантаженням, гнучкість у розвитку та адаптацію до змін у бізнес-вимогах. Для системи підтримки клієнтських запитів, яка обробляє великий обсяг даних і вимагає високої швидкості обробки запитів у реальному часі, важливо обрати архітектуру, що забезпечить баланс між продуктивністю, безпекою та простотою технічного обслуговування. Під час дослідження я розглянув найпоширеніші архітектурні рішення: моноліт, мікросервіси та безсервесна архітектури.

Монолітна архітектура — це традиційна модель програмного забезпечення, що є єдиним блоком, який працює автономно та незалежно від інших програм. Монолітна архітектура передбачає створення цілісної програми, в якій усі функції та компоненти тісно інтегровані в одну структуру[22]. Усі елементи — інтерфейс, серверна логіка, бізнес-логіка, бази даних і навіть сторонні API — функціонують як єдина система, яка розгортається разом. Монолітна архітектура - це окрема велика обчислювальна мережа з єдиною кодовою базою, в якій об'єднані всі бізнес-завдання. Щоб зробити зміни в таку програму, потрібно оновити весь стек через кодову базу та створити та розгорнути оновлену версію інтерфейсу, яка знаходиться на стороні служби. Це обмежує роботу з оновленнями та займає багато

часу. Моноліти чудово використовувати на ранніх стадіях проектів, щоб полегшити розгортання та не вимагати надто багато розумових зусиль для керування кодом. Це дозволяє відразу звільнити все, що є в монолітному додатку. Ця архітектура має значні переваги в порівнянні з іншими аналогами, а саме:

- легка розробка і легкий запуск програми - монолітні програми простіші у побудові та розгортанні на ранніх етапах. Весь код зберігається в одному репозиторії, і при оновленні вимагає тільки одного розгортання;

- покращена продуктивність - якщо врахувати, що додатки були зібрані правильно, то додаток з монолітною архітектурою буде працювати продуктивніше, ніж, наприклад, з мікросервісами. Це, знову ж таки, забезпечується одним програмним кодом і працює з «одного» місця.

Нижче, на рисунку 2.1 показано принцип роботи монолітної архітектури.

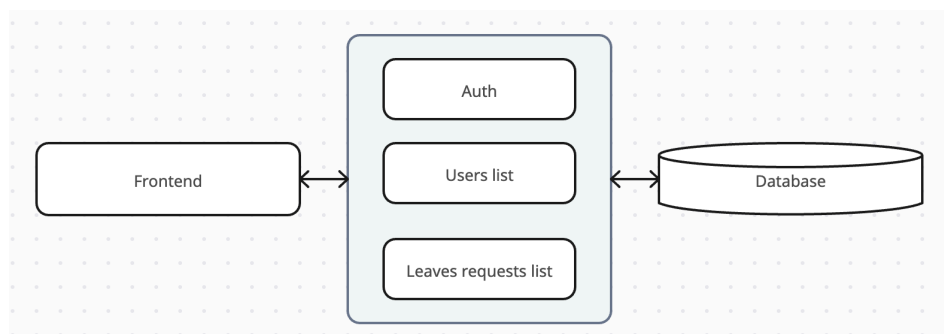


Рисунок 2.1 - Схема монолітної архітектури

Розглянемо наступний тип архітектури – мікросервісна архітектура. Мікросервісна архітектура це спосіб створення програмних продуктів, який передбачає розробку модулів, незалежних один від одного. Іншими словами, це звичайний підхід, коли додаток розбивається на невеликі самодостатні компоненти (мікросервіси) з чітко визначеними інтерфейсами[23]. Саме ця архітектура є типовою для хмарних додатків, що зараз популярні завдяки перевагам, які хмарні середовища відкривають для бізнесу. Кожна частина програми відповідає за певне завдання і може бути змінена або розширена без доповнень іншими, наприклад:

- сервіс управління користувачами - відповідає за створення, редагування, автентифікацію та авторизацію користувачів, коли користувач входить у систему,

цей мікросервіс перевіряє дані автентифікації та повертає токен для доступу до інших сервісів;

- сервіс обробки заявок на відпустку та лікарняні - обробляє заявки на відпустки та лікарняні, тобто коли працівник подає запит на відпустку, цей сервіс відправляє запит на розгляд менеджера, який, у свою чергу, може схвалити або відхилити його. Запис залишається в історії для звітності.

Це лише два приклади сервісів з яких може складатися мікросервісна архітектура, вони наведені виходячи з вимог до нашої системи. У той же час сервіси взаємодіють один з одним за допомогою обміну повідомленнями через API. Дана архітектура має значні переваги. Основною є масштабованість проекту, тобто завдяки мікросервісному рішенню можна внести певні зміни лише в один модуль, і вони жодним чином не зачеплять інші. Можна створювати незалежні міжфункціональні команди, націлені на вирішення конкретної бізнес-проблеми. Кожен елемент ІТ-системи, побудованої таким чином, виконує необхідну функцію, щоб її можна було передати на виробництво незалежно від інших служб. Також важливою перевагою є мультиплатформенність. За допомогою мікросервісів можна комбінувати різні технології, вибираючи найкращі можливі рішення. А використання стандартних протоколів взаємодії дозволяє писати мікросервіси різними мовами програмування та використовувати різні технології зберігання даних. На додаток до вищезазначеного, мікросервіси можуть працювати на будь-якому пристрої, у хмарному середовищі та локально. Але незважаючи на такі переваги, мікросервіси мають дуже значні недоліки:

- складність у комунікації - оскільки кожен мікросервіс працює незалежно, потрібні ефективні механізми для комунікації між сервісами (наприклад, REST або GraphQL);

- збільшення складності інфраструктури - управління великою кількістю мікросервісів вимагає розвиненої інфраструктури, включно з контейнеризацією (Docker, Kubernetes) та сервісами моніторингу;

- розподілене тестування - через розподіленість системи тестування стає складнішим, оскільки всі мікросервіси можуть взаємодіяти один з одним.

На рисунку 2.2 проілюстрована схема роботи даної архітектури.

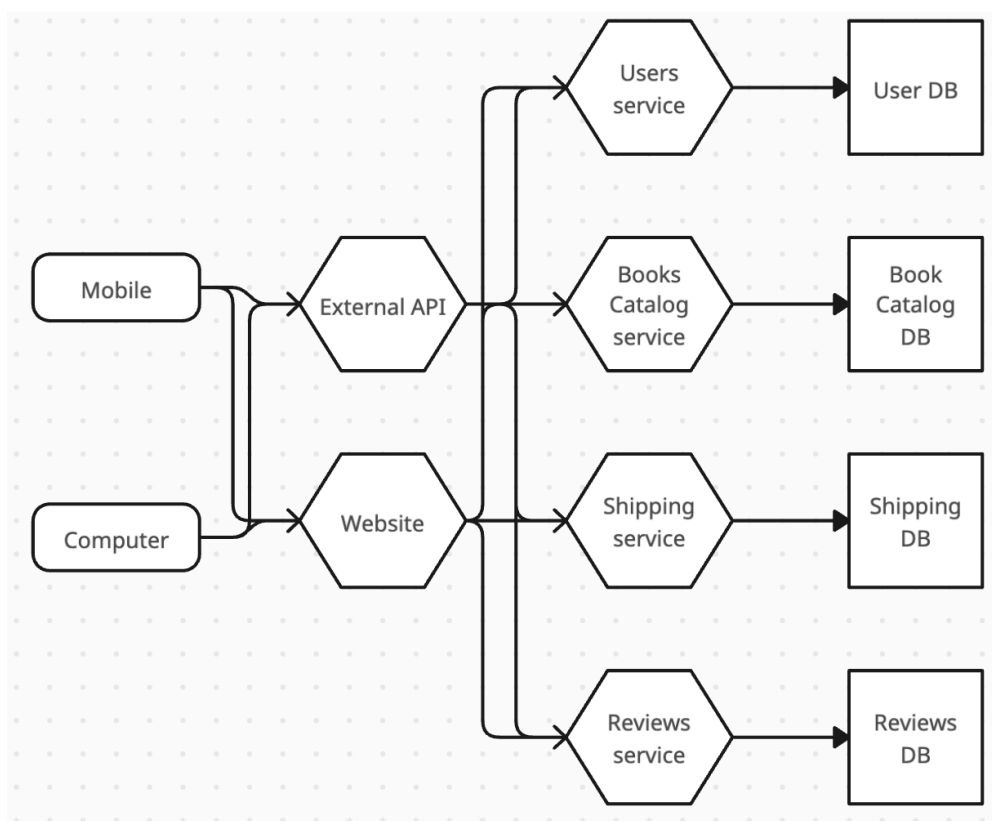


Рисунок 2.2 – Схема мікросервісної архітектури

Останній тип - безсерверна архітектура. Безсерверна архітектура дозволяє розробникам писати код, не турбуючись про управління серверними ресурсами. У цьому підході код виконується на серверах провайдера хмарних обчислень: AWS Lambda, Google Cloud Functions. Ресурс виділяється тільки в момент виконання коду, що дозволяє суттєво знизити витрати. У безсерверній архітектурі зовнішній хмарний постачальник відповідає за керування сервером. Розробникам просто потрібно потурбуватися про код, постачальник послуг подбає про оновлення безпеки, балансування навантаження, керування потужністю, масштабованість, журналювання та моніторинг[24]. За допомогою безсерверної архітектури можна запускати весь додаток або лише його частину. Щойно код програми запускається, сервер виділяє йому ресурси та звільняє їх, коли програма більше не використовується, тому це потрібно лише тоді, коли програма активно використовується(рис 2.3). Безсерверна архітектура має перевагу в швидкості

розгортання, тобто розробники можуть зосередитись на написанні бізнес-логіки, не турбуючись про налаштування серверів або їх підтримку. Але, величезним недоліком є обмеження продуктивності. Безсерверні функції мають обмеження щодо пам'яті та часу виконання, що робить їх непридатними для довготривалих задач.

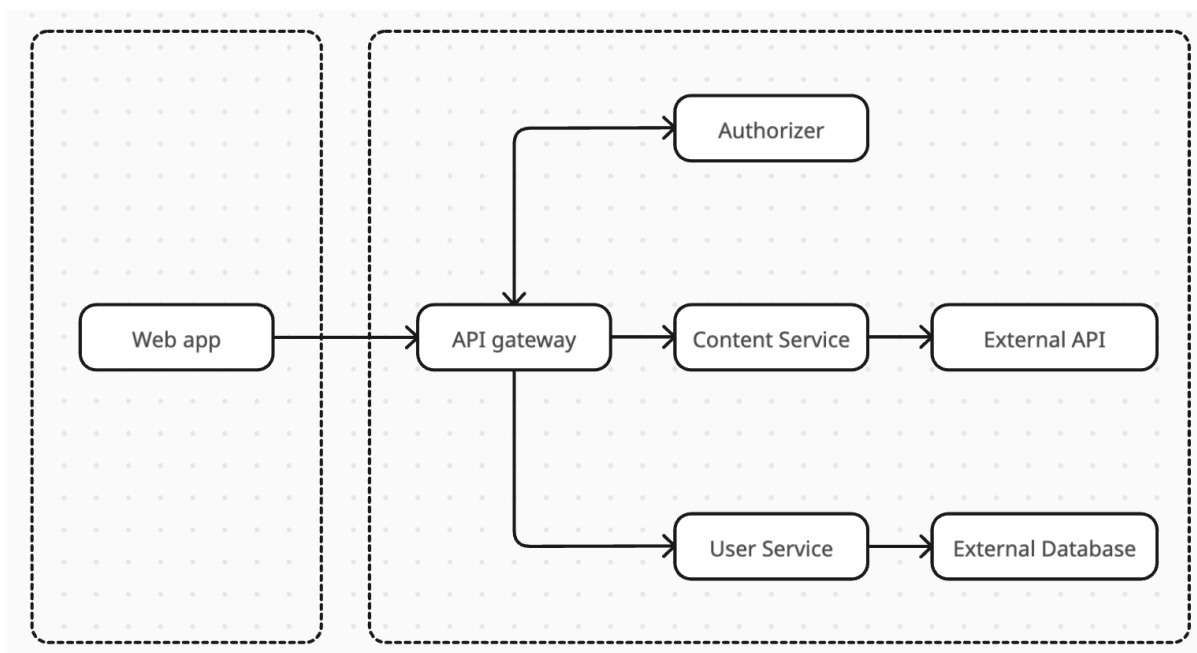


Рисунок 2.3 – Схема безсерверної архітектури

Монолітна архітектура є оптимальним вибором для цього проекту через її простоту та ефективність на початкових етапах розробки. Оскільки вся система працює як єдиний додаток, це значно полегшує управління кодом і налагодження, зменшує потребу в складних інтеграційних процесах, а також спрощує розгортання, адже всі компоненти системи працюють в одному середовищі. Такий підхід дозволяє уникнути багатьох труднощів, характерних для мікросервісної або безсерверної архітектури, які потребують додаткових зусиль для підтримки зв'язків між окремими сервісами та складнішого управління їхньою інфраструктурою. Обравши монолітну архітектуру, ми зменшуємо технічні ризики та скорочуємо час розробки, оскільки уникнемо необхідності будувати окремі інтерфейси зв'язку, системи балансування навантаження та підтримки безперервного обміну даними. Крім того, моноліт підходить, коли система має

обмежений обсяг функціоналу й команда невелика - це дозволяє ефективніше впоратися з поставленими задачами без ускладнень, характерних для розподілених систем.

Перейдемо до основних компонентів системи. Вони включають вебдодаток і мобільний додаток, кожен з яких виконує свою унікальну роль, але обидва працюють як частина фронтенду, взаємодіючи із сервером через API. Детальний опис цих двох компонентів дозволяє зрозуміти їхню окрему функціональність та взаємодію із системою.

Вебдодаток є основним інструментом для менеджерів та адміністраторів, які працюють зі складним функціоналом системи. Цей компонент орієнтований на використання з настільних пристроїв і надає широкий набір можливостей, зокрема: управління працівниками, моніторинг робочого часу, перегляд запитів на відпустки та лікарняні, створення звітів і аналіз статистики. Інтерфейс вебдодатка розробляється з урахуванням вимог до зручності використання, з акцентом на велику кількість даних, які потрібно ефективно візуалізувати. Завдяки інтеграції із сторонніми сервісами, такими як календарі та поштові клієнти, вебдодаток стає важливим інструментом для управління процесами в організації.

Мобільний додаток, на відміну від вебдодатка, призначений переважно для працівників, які мають потребу швидко отримувати доступ до своїх даних і виконувати базові операції. Основні функції мобільного додатка включають перегляд робочих змін, створення запитів на відпустки чи лікарняні, отримання сповіщень про зміну статусу запитів і перевірку залишку відпусток. Завдяки компактному інтерфейсу мобільний додаток забезпечує інтуїтивно зрозуміле управління для користувачів з мінімальною потребою у навчанні. Він також інтегрується з системою сповіщень на рівні пристрою (PUSH-повідомлення), забезпечуючи оперативне інформування працівників про важливі події.

Ці два компоненти - вебдодаток і мобільний додаток - доповнюють один одного, забезпечуючи покриття всіх потреб користувачів системи. Обидва компоненти взаємодіють із бекендом через API, що гарантує синхронність даних між усіма частинами системи. API обробляє всі запити, що надходять від веб- та

мобільного додатків, забезпечуючи логіку роботи та доступ до даних. Бекенд також відповідає за реалізацію бізнес-логіки, включаючи розрахунок робочих годин, перевірку доступності відпусток, обробку запитів працівників та затвердження їх менеджерами. Однією з ключових функцій серверної частини є надання API, через яке клієнтські додатки отримують доступ до необхідної інформації. API забезпечує стандартизовану взаємодію між клієнтами та сервером, що робить систему гнучкою та придатною до подальшого масштабування. Бекенд також керує доступом до бази даних, виконуючи операції створення, читання, оновлення та видалення даних. База даних є центральним сховищем даних системи. Вибір бази даних є критично важливим етапом при розробці проекту, оскільки від цього залежить ефективність обробки даних, масштабованість системи та її здатність адаптуватися до зростаючого навантаження. Тому важливо обрати БД, яка найкраще відповідає вимогам проекту. Розглянемо два типи БД, які можна обрати для потреб нашого проекту: реляційна та нереляційна БД.

Реляційна база даних (РБД) - це набір інформації, яка організовує дані в попередньо визначених зв'язках, де дані зберігаються в одній або кількох таблицях (або «зв'язках») стовпців і рядків, що дозволяє легко побачити та зрозуміти, як різні структури даних пов'язані одна з одною[25]. Кожна таблиця складається з рядків і стовпців, де кожен рядок є записом, а кожен стовець — атрибутом цього запису. Всі таблиці реляційної БД взаємопов'язані за допомогою зовнішніх ключів, що дозволяє створювати зв'язки між даними з різних таблиць. Реляційна модель передбачає використання мови запитів SQL (Structured Query Language), яка дозволяє здійснювати операції над даними, такі як додавання, оновлення, видалення та пошук інформації за допомогою складних запитів.

Нереляційна база даних (NoSQL) - це база даних, яка не використовує табличну схему рядків і стовпців, яка є в більшості традиційних систем баз даних. Натомість нереляційні бази даних використовують модель зберігання, оптимізовану для конкретних вимог типу даних, що зберігаються[26]. Дані зберігаються в інших формах, таких як документи, графи, пари ключ-значення або колонки (рис. 2.4). Нереляційні бази даних зазвичай використовуються для

зберігання великих обсягів даних, які не потребують жорсткої структури і можуть змінюватися з часом.

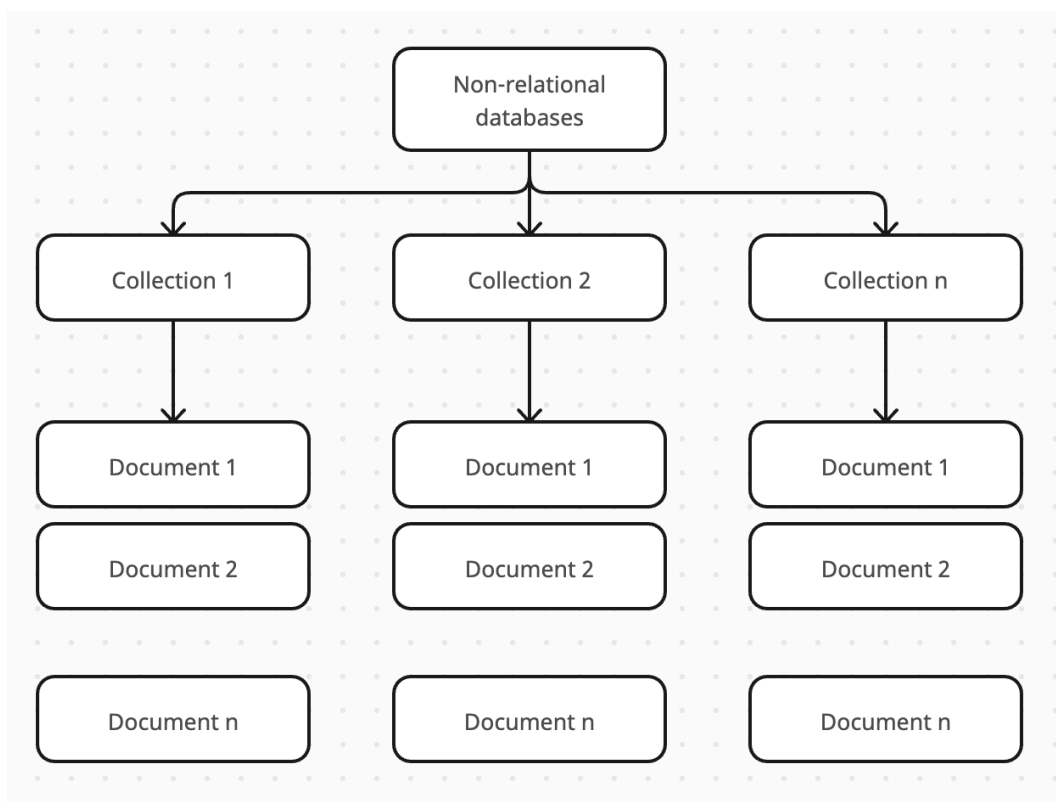


Рисунок 2.4 – Схема роботи нереляційної бази даних

Для реалізації цього проекту обрано нереляційну базу даних (NoSQL), оскільки вона краще підходить для динамічних і гнучких вимог системи. Основна перевага нереляційних баз даних полягає у їхній здатності працювати зі слабо структурованими або змінними даними, що дозволяє уникнути жорстких схем. Це особливо корисно в проектах, де структура даних може змінюватися в процесі розвитку системи або є потреба в швидкому масштабуванні. Основні переваги нереляційної бази даних у цьому проекті:

- гнучкість структури - можливість легко додавати нові поля або змінювати схему даних без необхідності модифікації існуючих записів;
- масштабованість - нереляційні бази даних зазвичай розподіляються між серверами, що забезпечує високу продуктивність і можливість роботи з великими обсягами даних;

- швидкий доступ до даних - завдяки оптимізації під читання і запис, такі бази чудово підходять для обробки великої кількості запитів у реальному часі.

Таким чином, у результаті аналізу можна зробити висновок, що правильний вибір архітектури та технологій для розробки системи є основою її ефективності та масштабованості. Кожен компонент, від фронтенду до бекенду та бази даних, відіграє важливу роль у забезпеченні зручності використання, швидкості обробки даних та безпеки. Вибір нереляційної бази даних для цього проекту, дозволяє оптимально вирішити завдання зберігання та обробки даних, забезпечуючи гнучкість та високу продуктивність при роботі з великими обсягами даних і частими змінами в структурі. Вибір монолітної архітектури є виправданим, оскільки він дозволяє зосередитися на основній функціональності, спрощує управління системою та забезпечує достатню продуктивність для створення стабільного й зручного рішення для підтримки клієнтських запитів.

2.2 Опис бізнес-логіки та основних сценаріїв

Спираючись на функціональні вимоги та цілі системи, описані у попередніх розділах, розробка бізнес-логіки також відіграє важливу роль у забезпеченні коректної роботи всіх компонентів. Бізнес-логіка визначає правила взаємодії між користувачами, ролями та процесами, забезпечуючи виконання завдань відповідно до заданих сценаріїв. Враховуючи специфіку системи, яка орієнтована на управління запитами, відстеження робочих годин та створення звітів, бізнес-логіка повинна бути чітко структурованою та враховувати ролі користувачів, механізми авторизації, обробки даних і тд.

Розпочнемо з процесу аутентифікації. Процес аутентифікації в системі організований таким чином, щоб забезпечити ефективний доступ користувачів до різних функцій залежно від їх ролі. У нашій системі немає необхідності в реєстрації нових користувачів через інтерфейс. Перший користувач компанії (Admin),

створюється за в Супер адмін панелі. Цей користувач має можливість створювати нових користувачів з різними ролями. Це забезпечує центральне управління доступом та дозволяє Admin визначати, хто має доступ до системи та з якими правами. Адміністратор після налаштування системи є відповідальним за управління доступом, створення та конфігурацію ролей у системі. Він може створювати нових користувачів і призначати їм відповідні ролі:

- admin – Admin має можливість створювати інших користувачів з роллю Admin, він також має повний доступ до всієї системи. Admin може не тільки керувати іншими користувачами та їх ролями, але й має доступ до всіх функцій як в мобільному, так і в веб-додатку;

- operator - працівники, які мають обмежений доступ лише до мобільного додатку. Оператори можуть створювати запити на відпустку, лікарняні тощо, але не мають доступу до жодної іншої функціональності вебдодатку;

- backoffice - особи, які обробляють запити на відпустку та лікарняні. Вони також мають доступ до мобільного додатку для подачі запитів, але їх головня роль полягає в прийнятті рішень щодо запитів. Вони мають певні адміністративні права, але їх доступ обмежений лише відповідними функціями.

Після того, як Admin створює нового користувача в системі, цей користувач отримує електронний лист з паролем для входу. Процес починається з того, що адміністратор вводить необхідні дані нового користувача: ім'я, прізвище, електронну адресу, роль, обирає відповідного користувача з роллю Backoffice, який буде відповідальний за цього користувача, а також можлива кількість доступних днів для відпустки, лікарняних, відгулів і кількість днів які можуть накопичуватись щомісяця, до цього перейдемо пізніше. Після збереження цих даних система автоматично генерує пароль, який надсилається на вказану електронну адресу користувача(рис. 2.5).

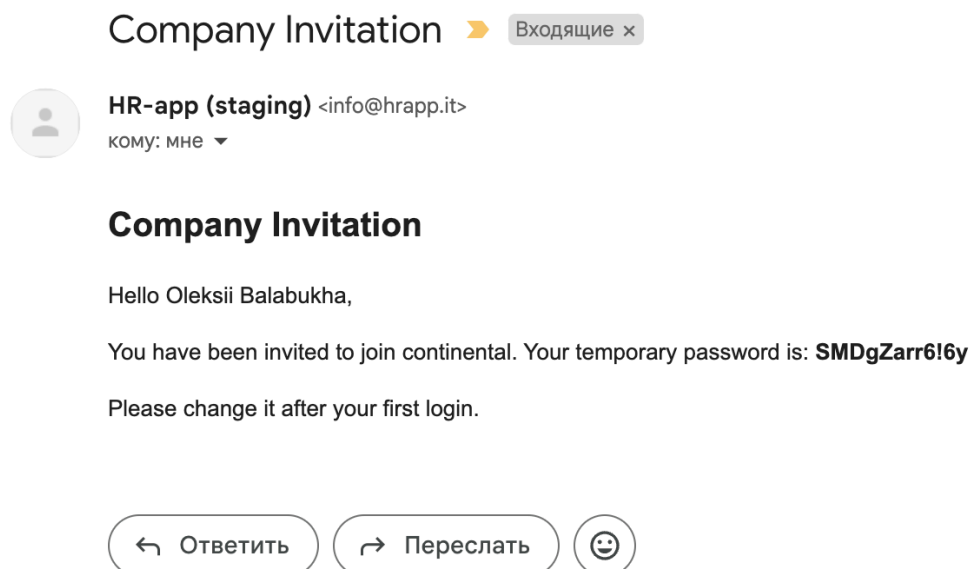


Рисунок 2.5 – Email із згенерованим паролем до системи

Користувач переходить до інтерфейсу входу в систему, де вводить свій email як логін та наданий пароль. Після успішної авторизації він отримує доступ до основних функцій системи. Після входу, користувач має можливість змінити свій пароль через налаштування профілю, щоб встановити більш безпечний пароль на свій розсуд. Цей процес забезпечує безпеку при первинній реєстрації та вході в систему, оскільки тимчасовий пароль гарантує, що лише користувач, якому надіслано листа, зможе отримати доступ до свого облікового запису.

Як зазначено вище, функціонал створення нового користувача Адміністратором, включає можливість налаштування днів для відпустки, лікарняного, відгулу та призначення відповідальних осіб із роллю Vackoffice. Цей процес забезпечує персоналізований підхід до кожного працівника, дозволяючи адаптувати систему під реальні потреби організації. Адміністратор може визначити початкову кількість доступних днів для кожної категорії: відпустки, лікарняного чи відгулу. Ці дні зберігаються в базі даних і використовуються системою для перевірки залишків під час створення запитів працівниками. Крім того, є можливість налаштувати щомісячне автоматичне додавання днів до певної категорії, наприклад, по 2 дні до відпустки або по 1 дню до лікарняного щомісяця. Це дозволяє уникнути ручного оновлення залишків, забезпечуючи зручність і

точність у роботі з балансом відпусток. Ще однією важливою частиною процесу є вибір рецензента із роллю Vackoffice. Адміністратор може призначити одного або декількох користувачів, які відповідатимуть за розгляд запитів нового працівника. Ці користувачі отримуватимуть email сповіщення про нові запити, зможуть переглядати залишки днів працівника та приймати рішення про затвердження чи відхилення запиту, додаючи коментарі за потреби. Призначення рецензентів дозволяє ефективно розподілити навантаження між членами команди, роблячи процес обробки запитів оперативним і організованим. У результаті ця функція інтегрує ключові процеси управління персоналом, сприяючи підвищенню ефективності роботи організації в цілому.

На практиці цей функціонал дозволяє оптимізувати всі процеси, пов'язані з управлінням днями відпусток, лікарняних чи відгулів. Система автоматично враховує зміни в балансі днів кожного працівника, що значно зменшує ризик помилок і ручної роботи. Працівники через мобільний додаток можуть у будь-який момент перевірити свій залишок днів, а рецензенти отримують чітку та актуальну інформацію для прийняття рішень. Прозорість і автоматизація таких процесів роблять систему зручною та зрозумілою для всіх її користувачів.

Процес створення та розгляду запитів на відпустку, відгул або лікарняний є важливою частиною бізнес-логіки системи, що забезпечує ефективне управління персоналом і дозволяє оперативно реагувати на запити працівників. Основні сценарії подання та обробки цих запитів розроблені з урахуванням гнучкості, автоматизації та прозорості процесу.

Для запитів на відпустку та відгули працівники з роллю Operator використовують мобільний додаток. У додатку працівник обирає відповідну категорію (відпустка або відгул), вказує дати та надсилає запит. Після цього користувач з роллю Vackoffice, у якого є можливість переглядати надсилаючи запити, отримує автоматичне повідомлення на електронну пошту, яке інформує про новий запит. У вебдодатку користувачі Vackoffice можуть перейти до списку запитів, знайти потрібний запит і переглянути його деталі: ім'я працівника який надіслав запит, дата надсилання, дати початку та кінця відпустки або відгулу, та

статус запиту. Під час розгляду запиту Vaskoffice має дві основні дії: підтвердити або відхилити запит. Якщо запит підтверджується, система автоматично перевіряє залишки доступних днів. У випадку, якщо доступних днів для відпустки чи відгулу немає, але запит все ж підтверджено, працівник проводить ці дні за свій рахунок. У цьому разі система записує до статистики негативне значення для відповідної категорії (наприклад, -2 дні), що дозволяє аналізувати ситуації, коли працівники перевищують ліміти.

Запит на лікарняний працює за схожим принципом, проте з певними унікальними особливостями. Після створення запиту через мобільний додаток працівник вказує дати, причину, і цей запит також надходить до Vaskoffice для розгляду. Коли користувач Vaskoffice відкриває запит на лікарняний, система автоматично перевіряє баланс доступних днів лікарняного. Якщо залишків немає, у користувача Vaskoffice з'являється два можливі варіанти дій(рис. 2.6):

- конвертувати лікарняний у відпустку або відгул, за умови, що у цих категоріях є доступний час. Система дозволяє обрати категорію, в яку буде переведено дні, і автоматично списує потрібну кількість із залишків відповідної категорії. Цей підхід дає гнучкість у використанні доступних ресурсів працівника;
- підтвердити лікарняний із записом негативного значення у статистику, якщо немає залишків днів у жодній із категорій. У цьому випадку працівник проводить лікарняний за свій рахунок, а у статистиці буде відображене від'ємне значення, що фіксує перевищення ліміту.

Уся історія запитів зберігається у системі для подальшого аналізу та звітності. Користувачі Vaskoffice можуть переглядати як поточні, так і закриті запити, включаючи їхній статус, залишки днів на момент розгляду. Для працівників у мобільному додатку передбачено відображення статусу запиту: очікує розгляду, підтверджено, відхилено. Цей функціонал забезпечує прозорість і автоматизацію процесів, дозволяє ефективно управляти ресурсами компанії та створює зручні умови для працівників і менеджерів. Гнучкість у розгляді запитів, особливо лікарняних, дозволяє адаптувати процеси під реальні потреби компанії, водночас

зберігаючи контроль за перевищеннями лімітів і надаючи можливість аналізувати статистику для подальшого планування.

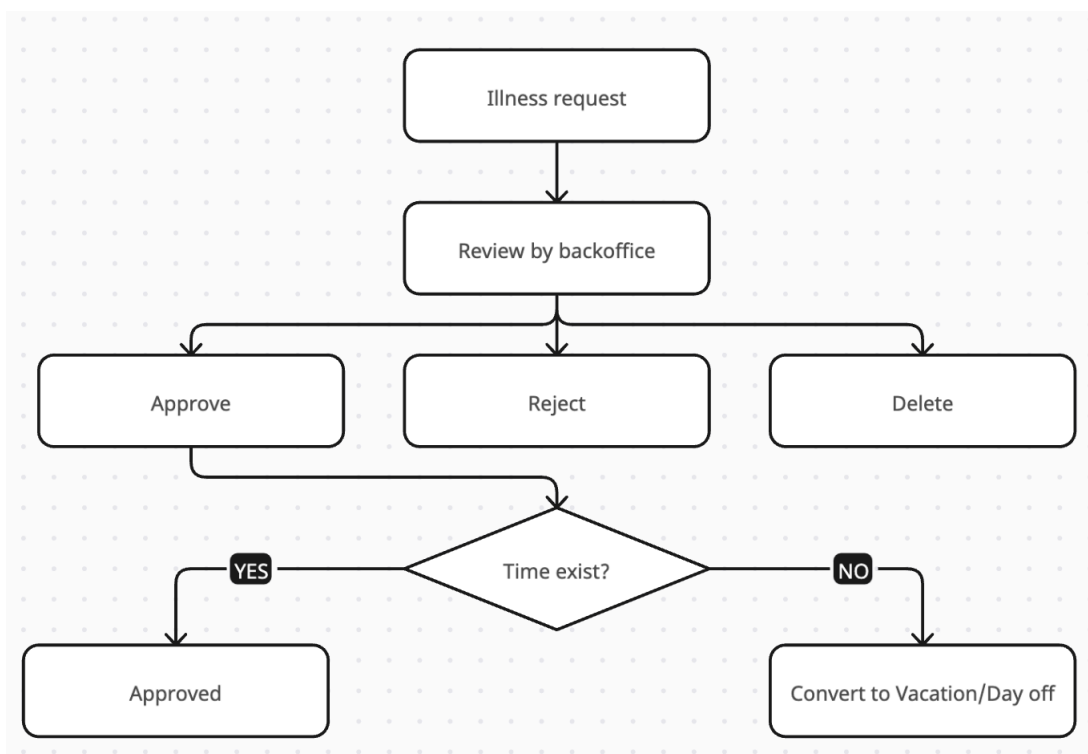


Рисунок 2.6 – Схема роботи запитів на лікарняний

Для забезпечення організованої та ефективної роботи система надає функціонал створення робочих змін, департаментів і графіків працівників, який дозволяє стандартизувати процеси, забезпечити гнучкість і підвищити прозорість управління. Створення робочих змін починається з введення їх назви, яка має бути унікальною, наприклад, "Ранкова зміна" або "Денна зміна". Далі задаються часи роботи, включаючи час початку і завершення зміни, а також дні тижня, у які ця зміна діятиме. Наприклад, зміна може охоплювати стандартний робочий тиждень з понеділка по п'ятницю або специфічні дні: понеділок, середа і п'ятниця. Після заповнення цих параметрів зміна зберігається у системі для подальшого використання.

Створення департаментів є простішим процесом і включає лише введення назви, яка також має бути унікальною, наприклад, "ІТ-відділ" або "Відділ кадрів".

Ці департаменти потім можуть бути прив'язані до працівників через графіки, що забезпечує чітку структурування та звітність.

Формування графіка працівника починається з вибору конкретного працівника із доступного списку. Далі Admin або Vackoffice додає одну робочу зміну, які відповідатимуть графіку цього працівника. Для кожного графіка обов'язково обирається департамент, до якого належить працівник, що дозволяє пов'язати його діяльність з конкретною структурною одиницею компанії. Також задається період, протягом якого діятиме графік, включаючи дату початку і, за необхідності, дату завершення. Наприклад, графік може бути активним з 1 грудня до 31 грудня 2024 року або залишатися відкритим до подальших змін.

Після заповнення всіх полів графік зберігається в системі, і його можна використовувати для обліку робочого часу, планування змін і формування звітності. Такий підхід забезпечує стандартизацію процесів, гнучкість у розподілі обов'язків, прозорість обліку робочого часу та оптимізацію роботи департаментів. Прив'язка графіків до департаментів дозволяє ефективно розподіляти ресурси, аналізувати продуктивність підрозділів і забезпечувати справедливий розподіл навантаження серед працівників. Таким чином, цей функціонал є важливою складовою для ефективного управління робочими процесами та ресурсами компанії.

Бізнес-логіка додатку ретельно спроектована для забезпечення ефективного управління персоналом, робочими процесами та запитами в межах компанії. Всі основні аспекти роботи системи спрямовані на автоматизацію та спрощення складних адміністративних завдань. Використання сповіщень і автоматичних обчислень підвищує ефективність роботи працівників з роллю Vackoffice, мінімізуючи ризик помилок і покращуючи комунікацію між працівниками. Таким чином, бізнес-логіка додатку не лише забезпечує високу ефективність та автоматизацію внутрішніх процесів, але й сприяє підвищенню продуктивності та задоволеності працівників, створюючи зручне та гнучке робоче середовище.

2.3 Вибір технологій

Розглянемо питання пов'язані з вибором технологій, за допомогою який працює наша система. Вибір технологій для розробки системи є важливим етапом, який визначає успішність реалізації проекту, його функціональність, масштабованість та ефективність. Правильно підібрані інструменти дозволяють не лише забезпечити необхідну продуктивність і стабільність роботи додатку, а й сприяють оптимізації процесу розробки, полегшують інтеграцію з іншими сервісами та забезпечують комфортну роботу для розробників. Крім того, вибір технологій впливає на витрати часу та ресурсів, можливості майбутньої підтримки та модернізації системи, а також зручність для кінцевих користувачів. Саме тому важливо ретельно проаналізувати доступні варіанти, враховуючи особливості проекту, його цілі та специфіку функціонування.

Для реалізації фронтенд-частини системи я обрав React, TypeScript, Material-UI та Zustand, враховуючи їх переваги, гнучкість і відповідність вимогам проекту. React був обраний як основний фреймворк для розробки інтерфейсу користувача завдяки його потужності та популярності в сучасній розробці. React забезпечує компонентний підхід, що дозволяє розробляти модульний, масштабований і легко підтримуваний код. Його віртуальний DOM оптимізує продуктивність навіть у системах із складними інтерфейсами, а велика і активна спільнота гарантує наявність численних бібліотек, інструментів і прикладів використання для вирішення практично будь-якої задачі. Крім того, React чудово інтегрується з іншими технологіями, такими як Zustand для управління станом або Material-UI для створення інтерфейсу.

TypeScript використовується для типізації коду в React-додатку, що забезпечує вищий рівень надійності та зменшує ризики виникнення помилок. Строгий контроль типів дозволяє уникнути багатьох поширених проблем у JavaScript, таких як помилки через неправильний тип змінної чи її відсутність. TypeScript полегшує роботу з великими командами, забезпечуючи зрозумілість і

передбачуваність коду, а також спрощує рефакторинг і інтеграцію нових функцій у проект, що у подальшому спрощує масштабованість проекту. Крім того, він добре підтримується у React-додатках і незначно впливає на швидкість розробки завдяки численним інструментам автодоповнення й перевірки коду.

Material-UI обраний як бібліотека компонентів для побудови інтерфейсу, оскільки вона пропонує набір сучасних, доступних компонентів, які відповідають принципам дизайну Material Design. Це значно прискорює розробку, адже більшість компонентів уже мають продуманий функціонал і готовий дизайн, що зменшує час, витрачений на створення елементів з нуля. Material-UI легко налаштовується, що дозволяє адаптувати стиль і функціонал компонентів під потреби конкретного проекту. Крім того, бібліотека має активну спільноту, детальну документацію та регулярні оновлення, що робить її надійним вибором для проекту.

Для управління станом у додатку добре підходить бібліотека Zustand, оскільки ця бібліотека є легкою, простою у використанні та ефективною. На відміну від більш важких рішень, таких як Redux, Zustand не вимагає великого обсягу налаштувань, що пришвидшує процес розробки. Вона надає зрозумілий API для роботи з глобальним станом і підтримує асинхронні дії, що є важливим для побудови сучасних додатків. Крім того, Zustand відмінно інтегрується з React, забезпечуючи високу продуктивність завдяки мінімізації зайвих повторних рендерів компонентів.

Тепер перейдемо до вибору backend технологій. Серверна частина реалізована за допомогою Nest.js, який забезпечує ефективну обробку запитів, а також MongoDB, що дозволяє зберігати гнучкі та масштабовані дані. Nest.js — це прогресивний фреймворк для розробки серверних додатків на Node.js, заснований на TypeScript. Основною перевагою Nest.js є його масштабованість та модульність, що дозволяє створювати добре структуровані додатки. Він підтримує як REST API, так і GraphQL, що дозволяє інтегрувати систему з мобільними та вебдодатками для ефективного обміну даними. Ін'єкція залежностей та декоратори в Nest.js роблять код чистим, гнучким і легко тестованим. Використання TypeScript забезпечує

додаткову безпеку типів, що зменшує кількість помилок на етапі компіляції і підвищує якість коду. Для системи підтримки клієнтських запитів, де важливі надійність і масштабованість серверної частини, Nest.js є ідеальним вибором.

Як зазначалося раніше в дипломній роботі, для реалізації додатку обрано нереляційну базу даних, а саме – MongoDB. MongoDB — це популярна NoSQL база даних, яка зберігає дані у вигляді документів у форматі BSON (розширення JSON). Вона добре підходить для зберігання великих обсягів неструктурованих або частково структурованих даних, таких як інформація про клієнтські запити, статистику, дані про відпустки/лікарняні тощо. Основні переваги MongoDB:

- гнучкість схеми - MongoDB дозволяє зберігати дані без суворих вимог до схеми, що є корисним при роботі з динамічними даними, такими як запити клієнтів, які можуть мати різні поля в залежності від типу запиту або статусу;

- масштабованість - MongoDB підтримує горизонтальне масштабування, що дозволяє ефективно працювати з великими обсягами даних та розподіляти навантаження на кілька серверів;

- швидкість запису та читання: MongoDB оптимізована для швидкого запису даних, що важливо для системи підтримки клієнтських запитів, де користувачі можуть часто оновлювати або додавати нові запити;

- природна підтримка JSON: оскільки дані часто передаються між клієнтськими і серверними частинами у форматі JSON, MongoDB є ідеальним варіантом для зберігання таких даних, забезпечуючи простоту інтеграції з іншими компонентами системи.

Для інтеграції PUSH-повідомлень у мобільний додаток, щоб оперативно інформувати користувачів про зміни в статусі їхніх запитів або інших важливих подій, таких як схвалення відпустки чи лікарняного, обрано Firebase. Це є потужною платформою для розробки мобільних та вебдодатків, що включає сервіси для аутентифікації, зберігання даних у реальному часі та як раз таки відправки PUSH-повідомлень. Firebase Cloud Messaging (FCM) дозволяє легко налаштувати відправку повідомлень без складних налаштувань, що значно скорочує час на реалізацію цієї функції. Важливою перевагою є високий рівень

безпеки та масштабованість сервісів Firebase, що дозволяє ефективно працювати з великою кількістю користувачів і забезпечити безперебійну роботу додатка.

Мобільний додаток реалізовано за допомогою React Native, оскільки ця технологія дозволяє створювати нативні мобільні додатки для iOS та Android з використанням одного коду на JavaScript. React Native забезпечує високу продуктивність завдяки використанню рендерингу компонентів через нативні віджети, що робить додаток максимально швидким і зручним для користувачів. Крім того, React Native підтримує широкі можливості для інтеграції з нативними функціями мобільних пристроїв, такими як камера, геолокація та PUSH-повідомлення, що є важливим для системи підтримки клієнтських запитів. Завдяки цьому вибору, можна ефективно створювати мобільний додаток для двох платформ без необхідності писати окремий код для кожної з них, що значно знижує витрати часу та ресурсів на розробку.

Таким чином, для розробки системи підтримки клієнтських запитів обрано одні з найактуальніших та ефективних технологій, які забезпечують високу продуктивність, масштабованість та зручність у розробці. Завдяки цьому вибору, система буде здатна обробляти великі обсяги даних, ефективно взаємодіяти з користувачами та забезпечити безперебійну роботу на всіх платформах.

2.4 Висновки до розділу 2

Можемо зробити висновок, що у результаті аналізу можна зробити висновок, що правильний вибір архітектури та технологій для розробки системи є основою її ефективності та масштабованості. Кожен компонент, від фронтенду до бекенду та бази даних, відіграє важливу роль у забезпеченні зручності використання, швидкості обробки даних та безпеки. Вибір нереляційної бази даних для цього проекту, дозволяє оптимально вирішити завдання зберігання та обробки даних, забезпечуючи гнучкість та високу продуктивність при роботі з великими обсягами

даних і частими змінами в структурі. Вибір монолітної архітектури є виправданим, оскільки він дозволяє зосередитися на основній функціональності, спрощує управління системою та забезпечує достатню продуктивність для створення стабільного й зручного рішення для підтримки клієнтських запитів.

Бізнес-логіка додатку ретельно спроектована для забезпечення ефективного управління персоналом, робочими процесами та запитами в межах компанії. Всі основні аспекти роботи системи спрямовані на автоматизацію та спрощення складних адміністративних завдань. Використання сповіщень і автоматичних обчислень підвищує ефективність роботи працівників з роллю Vascoffice, мінімізуючи ризик помилок і покращуючи комунікацію між працівниками. Таким чином, бізнес-логіка додатку не лише забезпечує високу ефективність та автоматизацію внутрішніх процесів, але й сприяє підвищенню продуктивності та задоволеності працівників, створюючи зручне та гнучке робоче середовище.

Для розробки системи підтримки клієнтських запитів обрано одні з найактуальніших та ефективних технологій, які забезпечують високу продуктивність, масштабованість та зручність у розробці. Завдяки цьому вибору, система буде здатна обробляти великі обсяги даних, ефективно взаємодіяти з користувачами та забезпечити безперебійну роботу на всіх платформах.

3 РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

3.1 Реалізація веб- та мобільного додатків

Виходячи з мети роботи, розглянемо питання налаштування базової структури проекту, що забезпечує ефективний процес розробки та майбутню підтримку системи. У випадку веб-додатку використовується Create React App, а для мобільного додатку — Expo або React Native CLI, що дозволяють швидко створити базові середовища для проекту. Після ініціалізації створюється чітка

структура файлів і папок, яка полегшує навігацію в коді. В веб- та мобільному дуже схожі структури, адже і там і там використовується React.

Для веб-додатку структура зазвичай включає папки для компонентів інтерфейсу (UI), сторінок, кастомних хуків, централізованого управління станом, API-запитів, а також статичних файлів(рис 3.1). Наприклад, папка components зберігає усі модульні елементи інтерфейсу, такі як кнопки чи форми, тоді як pages містить логіку відображення різних сторінок.

```
src/
├─ components/      # Компоненти UI
├─ pages/           # Сторінки додатку
├─ hooks/           # Кастомні хуки
├─ store/           # Zustand для управління станом
├─ services/        # API-запити
├─ assets/          # Статичні файли
└─ App.tsx          # Головний компонент
```

Рисунок 3.1 – Структура вебдодатку

У мобільному додатку структура схожа, однак замість сторінок використовуються екрани (screens), що відповідають концепції мобільного інтерфейсу(рис 3.2).

```
src/
├─ components/      # UI-компоненти
├─ screens/         # Екрани мобільного додатку
├─ hooks/           # Кастомні хуки
├─ store/           # Zustand для стану
├─ services/        # API-запити
├─ assets/          # Іконки, зображення
└─ App.tsx          # Головний компонент
```

Рисунок 3.2 – Структура мобільного додатку

Наступним важливим кроком є налаштування залежностей. Для веб-додатку інтегрована бібліотека Material-UI, яка дозволяє швидко створювати сучасний дизайн інтерфейсу з можливістю гнучкого налаштування тем. Також встановлена бібліотека для управління маршрутизацією (React Router). React Router — це бібліотека для управління маршрутизацією в додатках (SPA), розроблена для використання з React[27]. Вона дозволяє ефективно організувати навігацію між різними компонентами або сторінками додатка без необхідності перезавантажувати сторінку. Завдяки React Router можна динамічно змінювати вміст веб-сторінки, що значно покращує користувацький досвід.

У рамках нашого проекту React Router відіграє роль у забезпеченні переходів між різними розділами системи, такими як профіль користувача, панель адміністратора, налаштування тощо. Це дозволяє створити чітку та зрозумілу навігацію для користувачів, що є важливим для зручності використання веб-додатка. Завдяки можливостям React Router можна також визначати доступ до певних маршрутів на основі ролей користувачів. В рамках нашої системи, в нас є наступні ролі: SuperAdmin, Admin, Operator, Backoffice. Отже, базуючись на ролі користувача, та за допомогою React Router, система розуміє до якої сторінки користувач має доступ, а до якої ні. Це реалізується через перевірку ролі користувача перед відрисовкою відповідного компонента чи сторінки. Якщо користувач не має необхідних прав для доступу до певного маршруту, він буде перенаправлений на корневу сторінку. Таким чином, React Router не тільки спрощує навігацію в додатку, а й дає змогу реалізувати гнучку систему доступів, що є важливим аспектом для захисту даних та забезпечення безпеки в нашій системі, де різні категорії користувачів мають різний рівень доступу.

Реалізація базових компонентів інтерфейсу починається зі створення універсальних модулів, які використовуватимуться в додатку. Наприклад, компонент кнопки (Button) є максимально універсальним, щоб підтримувати різні стилі та функціональність. У веб-додатку для цього використовується Material-UI, тоді як у мобільному — бібліотека react-native-paper або власні стилі. Компоненти створюються так, щоб вони підтримували повторне використання та мали чітко

визначений функціонал. Для веб-додатку реалізовані такі ключові елементи, як форми для введення даних, модальні вікна, списки, таблиці, а також навігаційне меню. У мобільному додатку базові компоненти включають вкладки (tabs), картки (cards), списки та інтерактивні кнопки для роботи зі списками чи формами. Ці компоненти мають адаптуватися до різних екранів та підтримувати інтерактивність.

Крім того, базові компоненти інтегровані із системою управління станом (Zustand), що дозволяє синхронізувати дані між різними частинами додатку. Наприклад, стан користувача зберігається централізовано, що спрощує доступ до нього з будь-якої частини системи.

Перейдемо до реалізації бекенду. Як зазначалося, ми наша API створена за допомогою Nest.js. Nest.js використовує модульну архітектуру, що дозволяє розділяти проект на окремі, незалежні компоненти, які можуть бути легко модифіковані та тестовані без впливу на інші частини системи. Створення базової структури проекту починається з ініціалізації нового проекту за допомогою Nest CLI. Далі в рамках створеної структури за замовченням створюються різні компоненти, які дозволяють поділити логіку системи на незалежні частини. Основним компонентом у Nest.js є модулі (Modules). Модулі дозволяють організувати код за функціональними одиницями. Наприклад, для кожного основного бізнес-процесу системи (автентифікація, користувачі, відпустки тощо) створюється окремий модуль. У межах кожного модуля можна визначати:

- контролери (Controllers), які відповідають за обробку вхідних HTTP-запитів і взаємодію з користувачами;
- сервіси (Services), що містять бізнес-логіку і працюють з даними;
- моделі для взаємодії з базою даних або іншими джерелами даних.

Також важливою частиною структури є config — директорія, яка містить налаштування для підключення до зовнішніх сервісів і баз даних. Тут можна зберігати ключі для підключення до бази даних, налаштування для автентифікації або конфігурації самого застосунку (наприклад, секрети для JWT).

На рисунку 2.1 проілюстровано приклад структури для API.

```

src/
├─ app.controller.ts      # Контролер для основного модуля
├─ app.controller.spec.ts # Тести для контролера
├─ app.module.ts          # Основний модуль, де об'єднуються всі інші модулі
├─ app.service.ts         # Сервіс, який містить бізнес-логіку
├─ main.ts               # Точка входу в програму
├─ users/                # Директорія для модуля користувачів
│   └─ user.controller.ts # Контролер для обробки запитів до користувачів
│   └─ user.service.ts   # Сервіс для роботи з користувачами
│   └─ user.module.ts    # Модуль для користувачів
│       └─ dto/          # Директорія для об'єктів передачі даних (DTO)
├─ auth/                # Директорія для модуля автентифікації
│   └─ auth.controller.ts # Контролер для автентифікації
│   └─ auth.service.ts   # Логіка автентифікації
│   └─ auth.module.ts    # Модуль для автентифікації
│       └─ dto/          # DTO для автентифікації
├─ config/              # Конфігураційна директорія для налаштувань
│   └─ database.config.ts # Конфігурація для підключення до бази даних
├─ shared/              # Спільні компоненти для різних модулів
│   └─ dto/             # Загальні DTO для проекту
│       └─ pipes/       # Пайпи для валідації даних
└─ entities/           # Моделі для взаємодії з базою даних
    └─ user.entity.ts   # Модель користувача для бази даних

```

Рисунок 3.3 – Структура API

Розглянемо наступне питання, а саме налаштування API-запитів (REST), адже саме через API здійснюється взаємодія між клієнтською частиною системи та сервером. У нашій системі REST API використовується для обміну даними, забезпечення безпеки запитів і реалізації бізнес-логіки. Процес налаштування API-запитів починається зі створення ендпоінтів на серверній стороні, реалізованих за допомогою Nest.js. Ці ендпоінти розробляються відповідно до принципів CRUD-операцій:

- GET запити — для отримання даних (наприклад, список користувачів, графіки роботи, доступні відпустки тощо);
- POST запити — для створення нових записів (наприклад, створення користувача або додавання робочої зміни);
- PUT/PATCH запити — для оновлення існуючих даних (наприклад, оновлення профілю користувача);

- DELETE-запити — для видалення даних (наприклад, видалення департаменту).

Всі ендпоінти систематично документуються за допомогою OpenAPI (Swagger), що дозволяє легко зрозуміти їх структуру, параметри та можливості використання (рис. 3.4). Swagger — це набір інструментів, стандартів і специфікацій для створення, опису, документування та тестування REST API. Він допомагає розробникам і командам ефективно працювати з API, забезпечуючи зрозумілий і структурований спосіб опису функціоналу сервісів.

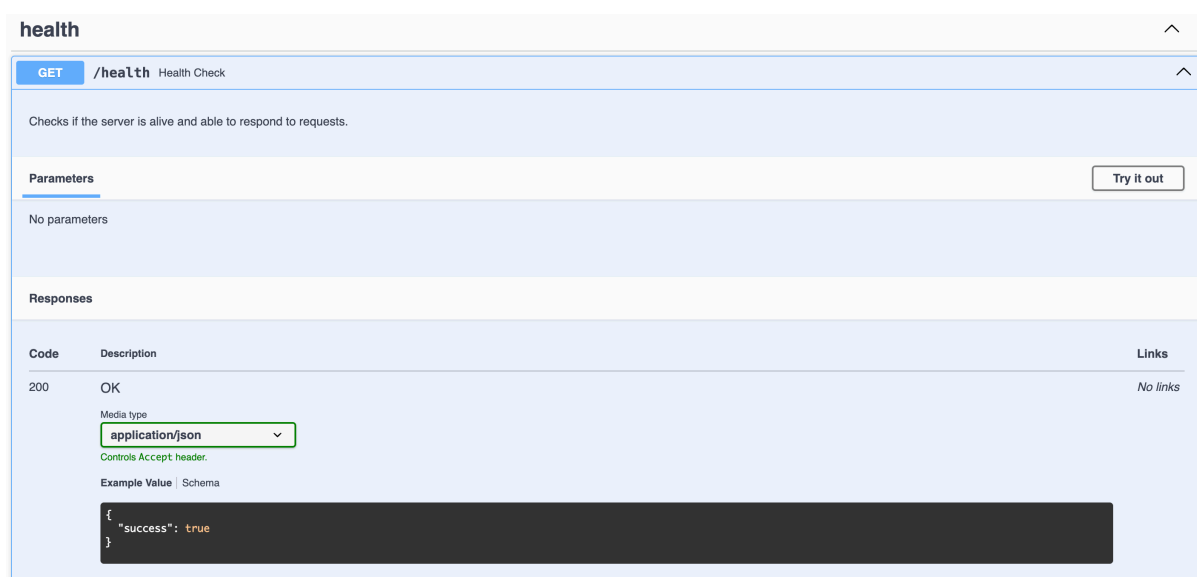


Рисунок 3.4 – Swagger документація до health ендпоінту

На клієнтській стороні, API-запити налаштовані для зручної та безпечної роботи з даними. Для здійснення запитів використовується бібліотека axios, яка забезпечує простоту інтеграції, можливість налаштування глобальних параметрів, таких як базова URL-адреса сервера, заголовки запитів, час очікування та перехоплювачі для обробки запитів і відповідей. Одним із ключових аспектів є створення базового клієнта API, який спрощує повторне використання коду. У ньому налаштована базова URL-адреса сервера, додані заголовки для передачі типу даних і токен доступу. Це дозволяє кожному запиту автоматично звертатися до відповідного ендпоінту з мінімальною кількістю додаткових налаштувань. Особлива увага приділяється обробці помилок, щоб забезпечити стабільність

роботи додатків і зрозумілий зворотній зв'язок для користувача. Якщо, наприклад, сервер повертає помилку 401 Unauthorized, це означає, що токен доступу недійсний або строк його дії закінчився. У такому випадку користувача автоматично перенаправляють на сторінку входу повторної авторизації. У разі інших помилок, таких як 404 Not Found або 500 Internal Server Error, відображаються повідомлення для пояснення причини проблеми. Такий підхід забезпечує більш зрозумілий та інтуїтивний досвід взаємодії з додатком.

Захист даних є ще одним важливим елементом налаштування API. Усі дані передаються через HTTPS-протокол, який забезпечує шифрування та запобігає перехопленню інформації. Додатково використовуються захищені заголовки та механізми CORS для обмеження доступу до API лише з авторизованих доменів. Для приватних ендпоінтів всі запити повинні містити токен доступу, який передається у заголовку Authorization. Цей токен генерується під час авторизації користувача і зберігається в безпечному сховищі, такому як localStorage для веб-додатків або secure storage для мобільних додатків. Для оптимізації роботи клієнта передбачено кешування даних, таких як список департаментів або доступні графіки роботи. Це дозволяє зменшити кількість запитів до сервера та забезпечити швидкий доступ до інформації, навіть за умови тимчасової недоступності мережі. Кеш автоматично оновлюється після отримання нових даних із сервера, що дозволяє користувачам завжди мати доступ до актуальної інформації.

Таким чином, у процесі реалізації ми створили базову структуру проекту, яка забезпечує модульність, масштабованість і зручність подальшого розвитку. Особливу увагу було приділено взаємодії з серверною частиною через API-запити, що гарантує надійність передачі даних та авторизацію доступу та забезпечує ефективну комунікацію між клієнтською та серверною частинами. Це дозволяє системі функціонувати як єдиний злагоджений механізм, готовий до масштабування та інтеграції нових функцій.

3.2 Загальні налаштування та інтерфейс користувача

Перейдемо до дослідження стилів додатку. Налаштування глобального стилю додатків, на мою думку, є найважливішим етапом у реалізації додатку. Адже це свого роду обличчя додатку, від цього залежить чи буде користувачу зручно та легко користуватись, чи буде дизайн виглядати за сучасними стандартами і тд. Від добре продуманого дизайну залежить загальний вигляд і зручність користування веб- та мобільними додатками. Основним завданням є створення візуальної гармонії та єдиного стилю, який буде відповідати вимогам бренду, сучасним тенденціям дизайну та забезпечувати зручний користувацький досвід.

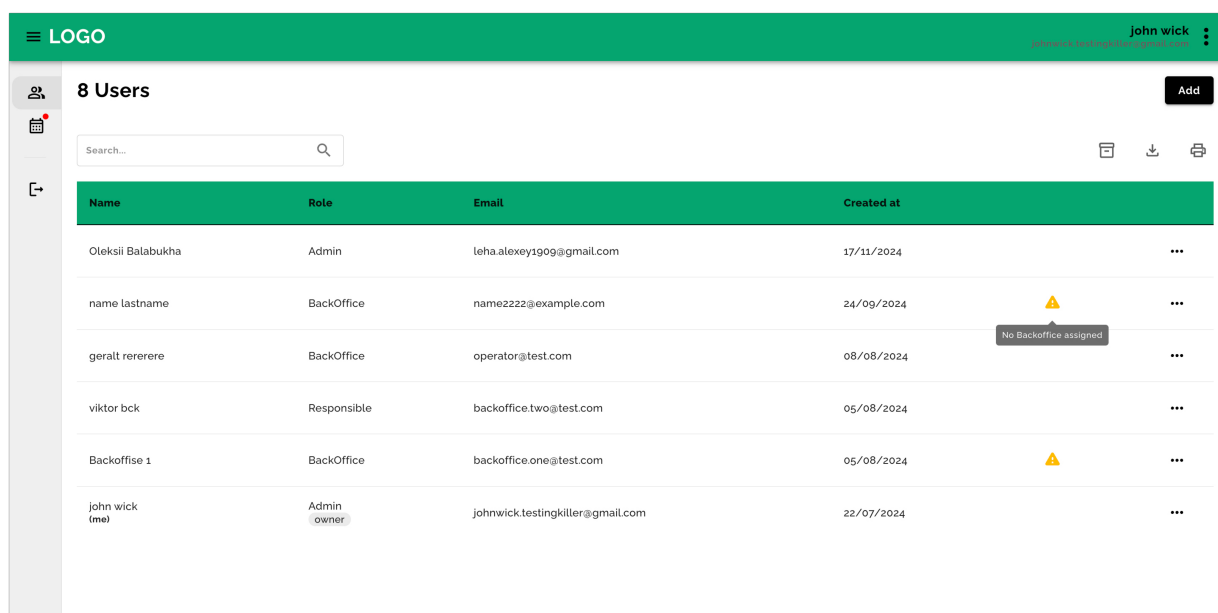
Розпочнемо з кольорової палітри. Кольорова палітра виконує функцію не лише естетичного оформлення, але й інструменту для побудови зручної навігації та акцентування уваги на важливих елементах інтерфейсу. Для проєкту було розроблено наступну структуру кольорів:

- основні кольори – використовується для ключових елементів, таких як активні кнопки, посилання, заголовки. Це забезпечує впізнаваність і логічну послідовність в інтерфейсі;
- акцентні кольори – застосовуються для сповіщень, важливих індикаторів або підкреслення інтерактивних елементів;
- фонові кольори включають світлі та темні відтінки, які допомагають створити розділення між різними інформаційними блоками;
- нейтральні кольори (чорний, білий, сірий) – використовуються для тексту, іконок та вторинних елементів.

Вибір кольорів здійснювався з урахуванням принципів контрастності для забезпечення доступності додатка. На рисунку 3.5 показано зовнішній вигляд одного з модулів, а саме таблиця працівників компанії.

Налаштування глобального стилю та кастомної теми створює єдність і зручність у користуванні системою. Це не лише підвищує естетику інтерфейсу, а й сприяє більш ефективній навігації та сприйняттю інформації, що особливо важливо

для системи з кількома ролями користувачів та широким спектром функціональності.



Name	Role	Email	Created at
Oleksii Balabukha	Admin	leha.alexey1909@gmail.com	17/11/2024
name lastname	BackOffice	name222@example.com	24/09/2024
geralt rerere	BackOffice	operator@test.com	08/08/2024
viktor bck	Responsible	backoffice.two@test.com	05/08/2024
Backoffice 1	BackOffice	backoffice.one@test.com	05/08/2024
john wick (me)	Admin owner	johnwick.testingskiller@gmail.com	22/07/2024

Рисунок 3.5 – Дизайн вебдодатку

Для текстових елементів системи було обрано шрифт Raleway. Цей шрифт відзначається сучасним, стильним виглядом, що підкреслює інноваційність системи, та високою читабельністю:

- витончені пропорції, які створюють професійний вигляд інтерфейсу;
- чітка структура символів, що забезпечує легкість сприйняття тексту на будь-яких пристроях;
- гнучкість у використанні завдяки різноманітності ваг шрифту (від тонкого до жирного). Основний шрифт використовується для тексту, а варіації шрифту (наприклад, жирний або курсив) застосовуються для заголовків, підписів чи акцентів.

Перейдемо до локалізації. Локалізація є важливим аспектом сучасних веб- та мобільних додатків, особливо якщо система орієнтована на міжнародну аудиторію або різні регіони. У рамках даного проєкту механізм локалізації реалізовано за допомогою бібліотеки react-intl, яка є одним із найпопулярніших рішень для роботи з багатомовністю в React-додатках. Цей інструмент надає потужний і гнучкий

функціонал для управління текстовими елементами, форматування дат, чисел та інших даних відповідно до мовних і регіональних стандартів. React-intl пропонує зручний API для динамічного вибору мов, підтримки складних мовних конструкцій і інтеграції форматування чисел, дат та інших локалізованих даних. Завдяки використанню стандарту ICU MessageFormat, react-intl підтримує складні текстові структури, такі як форми множин, умовні вирази, форматування чисел, дат і валют. Основна ідея роботи з react-intl полягає в створенні словників для кожної підтримуваної мови, які містять переклади текстових елементів додатка. Бібліотека надає контекст для передачі локалізованих даних до компонентів, забезпечуючи динамічне відображення тексту залежно від обраної мови. Переваги react-intl наступні:

- відповідність стандартам - бібліотека використовує ECMAScript Internationalization API, що гарантує точність і відповідність локальним мовним стандартам;

- гнучкість - React-intl дозволяє легко додавати нові мови, змінювати існуючі переклади або адаптувати форматування для конкретної мови;

- робота з динамічними значеннями - бібліотека підтримує вставку змінних у текст, що зручно для роботи зі складними повідомленнями, які залежать від контексту;

- форматування даних - React-intl включає функціонал для локалізованого форматування чисел, валют, дат, часу тощо;

- сумісність із сучасним React - бібліотека інтегрується через контекст API, що забезпечує ефективне поширення мовних параметрів через компонентну структуру;

- легка інтеграція - React-intl має простий у використанні API, який дозволяє швидко налаштувати локалізацію навіть у складних додатках.

Додаток підтримує механізм динамічного перемикання мов, що дозволяє користувачу обрати потрібну мову через елементи інтерфейсу через випадаюче меню. Вибір зберігається в профілі користувача на сервері, щоб при наступному вході система автоматично застосувала відповідні налаштування. У проекті, де

присутні різні ролі користувачів, локалізація сприяє підвищенню інклюзивності системи, дозволяючи кожному користувачу працювати з додатком рідною мовою. В нашому додатку підтримується дві мови, англійська та італійська. На рисунку 3.6 проілюстровано приклад локалізації на основі сторінки налаштування користувача.

The image displays two side-by-side versions of a user profile page, demonstrating localization. The left version is in Italian, titled 'Profilo personale', and the right version is in English, titled 'Personal profile'. Both pages feature a language dropdown menu in the top right corner, currently set to 'Italian' and 'English' respectively. The form fields are identical in structure but labeled in the respective languages. The password requirements section lists the same criteria in both languages: a minimum of 8 characters, at least one number (0-9), at least one special character (# - !), and at least one capital letter (A - Z). The bottom of each page has an 'Aggiorna' (Update) button and a 'Password dimenticata' (Forgot password) link in the Italian version, and an 'Update' button and a 'Forgot password' link in the English version.

Рисунок 3.6 – Локалізація сторінки налаштування користувача

Реалізація механізму локалізації за допомогою react-intl не лише підвищує зручність роботи з системою для користувачів різних мов, але й забезпечує адаптивність проекту до нових ринків, роблячи його більш універсальним і конкурентоспроможним. Таким чином, інтеграція локалізації з використанням react-intl дозволяє створити зручний, сучасний інтерфейс, що відповідає стандартам доступності та враховує індивідуальні потреби кожного користувача.

Отже, ми розглянули налаштування глобальних стилів та компонентів інтерфейсу. Вибір кольорової палітри та шрифтів, зокрема використання Raleway, дозволив створити сучасний та стильний вигляд додатку. Механізм локалізації

через `react-intl` забезпечив багатомовність додатка, дозволяючи зручно перемикатися між мовами. У результаті ми досягнули функціональності та естетики, що відповідають сучасним вимогам.

3.3 Функціональна схема роботи системи

Виходячи з теми підрозділу, розглянемо більш детально взаємодію між фронтендом та бекендом. Обмін даними між компонентами системи забезпечує коректну роботу веб- та мобільного додатків. Система побудована на взаємодії трьох основних компонентів: фронтенду, бекенду та бази даних. Фронтенд відповідає за інтерфейс користувача і використовує `React` для динамічного відображення інформації. Він працює із запитами до бекенду через `REST API`, який забезпечує передачу даних між клієнтом і сервером. За допомогою `HTTP`-запитів фронтенд надсилає дані, наприклад, про створення нових запитів, перегляд робочих графіків або зміну профільних даних, до бекенду. Бекенд, у свою чергу, виступає як посередник між фронтендом і базою даних. Його завдання – приймати запити від фронтенда, обробляти їх, перевіряти на валідність, виконувати бізнес-логіку та формувати відповіді. Якщо запит потребує взаємодії з даними, бекенд передає відповідні інструкції до бази даних. У випадках, коли дані потребують додаткової обробки або виконання складних операцій, бекенд це реалізує, а потім повертає відповідь фронтенду у зручному форматі `JSON`. База даних використовується для зберігання структурованих та неструктурованих даних. У ній зберігаються всі дані, пов'язані із запитам, профілями користувачів, графіками роботи, доступними змінами тощо. Кожна дія, яку виконує бекенд, відбивається на рівні бази даних. Наприклад, при створенні нового робочого графіка записуються відповідні зміни з датами, які потім можуть бути використані для формування аналітики.

Ця взаємодія побудована таким чином, щоб забезпечити ефективність, швидкість і надійність. Розподіл функціональності між компонентами дозволяє підтримувати масштабованість і спрощує розробку нових функцій. Фронтенд відповідає за зручність користування, бекенд – за логіку і безпеку, а база даних – за надійне зберігання інформації(рис 3.7).

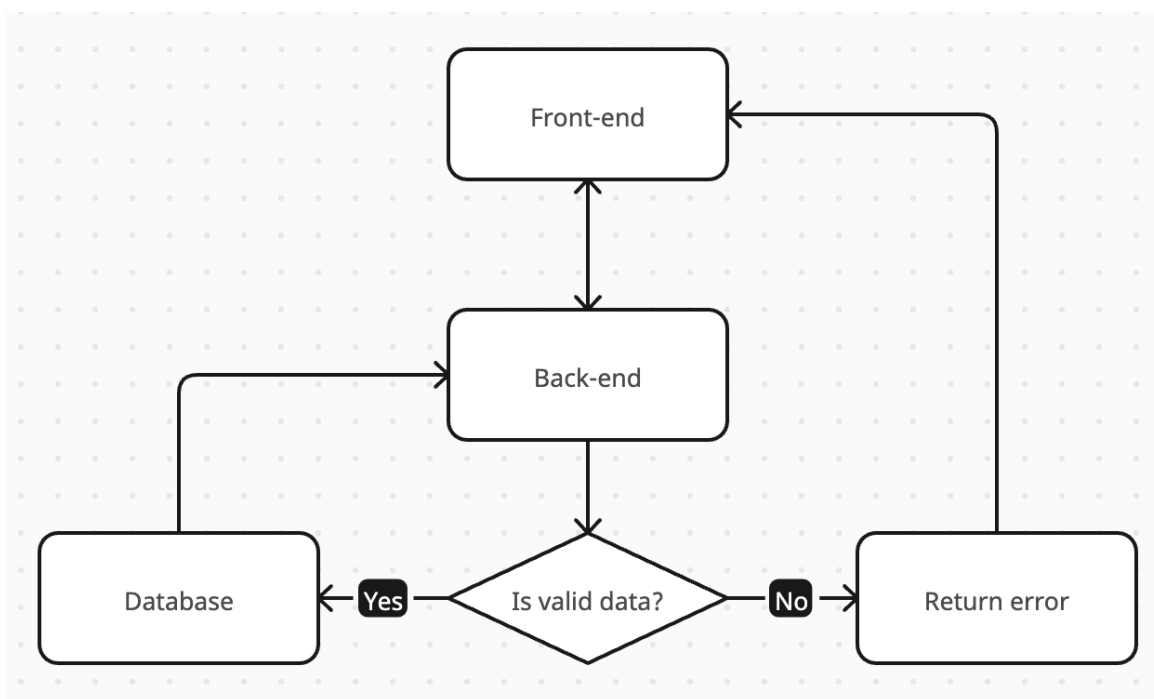


Рисунок 3.7 – Схема взаємодії фронтенду, бекенду та бази даних.

MongoDB, як нереляційна база даних, зберігає інформацію у вигляді документів у форматі JSON, що дозволяє ефективно працювати з динамічними структурами даних. У розробленій системі базу даних організовано за принципом колекцій, які можна уявити як аналоги таблиць у реляційних базах. Кожна колекція містить документи, які відповідають окремим записам. Розглянемо наступні колекції, які передбачені нашою системою: Users, Requests, Schedules та Statistics(рис 3.8).

Коли користувач реєструється або авторизується в системі, його дані зберігаються в колекції Users. Документ користувача включає унікальний ідентифікатор (`_id`), контактну інформацію, тип ролі та інші метадані. Усі паролі зберігаються у зашифрованому вигляді для забезпечення безпеки. Запити, створені

користувачами, потрапляють до колекції Requests. Кожен запит містить унікальний ідентифікатор, дані про користувача, який його створив, статус запиту (pending review, approved, reject), а також дату створення та останнього оновлення. Запити мають посилання на відповідні документи з інших колекцій, якщо це необхідно, наприклад, на документи з Schedules, для того щоб перевіряти графіки роботи користувачів за їх розкладом. Графіки роботи співробітників зберігаються в колекції Schedules. Ця колекція містить записи з інформацією про дати, типи змін (ранкові, денні, нічні) та користувачів, які призначені на ці зміни. У разі змін у графіку система оновлює відповідний документ. Для зберігання аналітичних даних використовується колекція Statistics. Ця колекція містить агреговану інформацію про кількість створених запитів, середній час їх обробки, кількість змін у графіках тощо. Агрегації реалізуються за допомогою оператора aggregate, що дозволяє виконувати складні обчислення, групування та сортування прямо в базі даних.

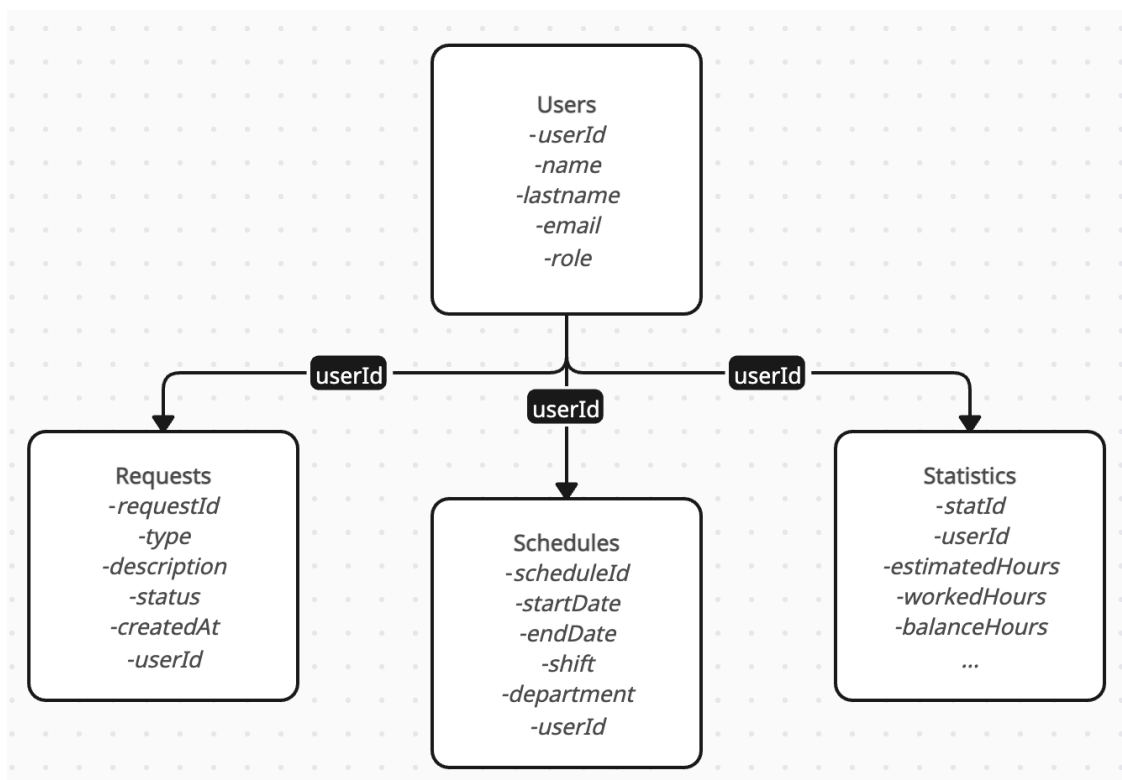


Рисунок 3.8 – Схема структури даних у колекціях MongoDB

Доступ до бази даних здійснюється через бекенд. Для кожної операції доступу використовуються відповідні методи бібліотеки Mongoose, що надає ORM

(Object-Relational Mapping) для MongoDB. Наприклад, для отримання інформації про користувача використовується метод `findOne`, а для створення нового запиту — `create`. Усі запити до бази даних захищені авторизацією через токени JWT, що гарантує безпечний доступ до даних. Однією з ключових переваг MongoDB є можливість горизонтального масштабування, що дозволяє розширювати базу даних при збільшенні навантаження на систему. Завдяки індексації за ключовими полями, такими як `_id`, продуктивність системи залишається високою навіть при обробці великого обсягу даних.

У попередніх розділах ми детально розглянули бізнес-логіку, яка описує, як система повинна функціонувати в рамках однієї компанії, забезпечуючи її повсякденні потреби. Однак, у реальному світі компанії можуть суттєво відрізнятися одна від одної, мати свої унікальні процеси, структуру та потреби. Крім того, часто виникає необхідність об'єднати всі ці компанії під однією системою, яка буде централізовано управляти їх даними та користувачами, але при цьому забезпечувати незалежність кожної компанії одна від одної.

Для вирішення цього завдання передбачено створена Супер адмін панель — це є окремим модулем управління системою, який використовується для адміністрування системи на найвищому рівні. У цьому модулі реалізовані ролі `SuperAdmin` і `SuperUser`, які дозволять ефективно управляти структурою компаній і правами доступу користувачів. На рисунку 3.9 зображена таблиця з такими типами користувачів.

< LOGO				Oleksii bale	
2 Users				Add	
Name	Role	Email	Created at		
Oleksii bale	Admin	balabukha.oleksii.frl@gmail.com	09/07/2024	...	
name ad	User	admin@mail.com	12/07/2024	...	

Рисунок 3.9 – Таблиця користувачів супер адмін панеллю

Розберемо більш детально які права мають ці користувачі. Перш за все, ці користувачі мають доступ лише до Супер адмін панелі, тобто вони не можуть увійти до системи та бачити інформацію як до прикладу звичайний Admin або Backoffice. Основною задачею SuperUser та SuperAdmin є забезпечення гнучкого, безпечного і централізованого управління компаніями, які використовують систему. SuperAdmin володіє найвищим рівнем доступу в системі. Його основні права включають створення, видалення та редагування користувачів, які мають доступ до Супер адмін панелі. Це дозволяє контролювати доступність та функціонал адміністративної частини платформи. SuperAdmin також відповідає за створення та управління компаніями, що включає введення таких даних, як назва компанії, електронна пошта, адреса компанії, номер телефону та ім'я з прізвищем власника. Після створення компанії система автоматично генерує користувача з роллю Admin. На вказану електронну адресу власника компанії надсилається лист із запрошенням, що містить посилання для входу та початковий пароль. Після першого входу в систему обліковий запис адміністратора компанії активується, і він отримує повний доступ до функціоналу своєї компанії. Окрім створення компаній, SuperAdmin має можливість видаляти компанії, включати та виключати доступ до функціоналу, а також переглядати детальну інформацію про кожну компанію. SuperUser в свою чергу має всі ті ж самі права, окрім секції з створенням, та взагалі з переглядом користувачів, які мають доступ до Супер адмін панелі. Цей підхід забезпечує чітку ієрархію доступу в системі. СуперАдмін контролює всю платформу, включаючи користувачів Супер адмінки, тоді як СуперЮзер обмежений лише управлінням компаніями.

Нижче на рисунку 3.10 проілюстрована таблицю користувачів власників компаній, тобто користувачі з роллю Admin.

У системі управління користувачами передбачено механізм статусів, який дозволяє ефективно керувати життєвим циклом кожного користувача. Для цього впроваджено чотири статуси: draft, invited, active та not active(рис. 3.11). Кожен із цих статусів відображає поточний стан користувача у системі та визначає його права і доступи.

Company name	Email	Created at	N° Users	First login	Status
test1w	asa@qma.com	28/07/2024	1	-	Invited
testCompany	testrestqwert@gmail.com	25/07/2024	7	28/11/2024	Not active
continental	johnwick.testingkiller@gmail.com	22/07/2024	9	28/11/2024	Active
company name	test@test123.com	22/07/2024	1	-	Draft

Рисунок 3.10 – Таблиця компаній в супер адмін панелі

- draft - встановлюється одразу після створення нового користувача. На цьому етапі користувач ще не є активним і не має доступу до системи. Це дозволяє супер адміністратору ретельно перевірити введені дані, внести необхідні зміни або видалити користувача, якщо він був створений помилково. Цей статус корисний для підготовчого етапу, коли необхідно влаштувати доступи чи уточнити інформацію перед тим, як надати користувачу можливість роботи у системі;

- invited - застосовується, коли адміністратор впевнений у необхідності додати цього користувача до системи. У цьому випадку користувачу надсилається електронний лист із запрошенням, що містить унікальне посилання для входу до системи та інструкції для активації облікового запису. У цьому статусі користувач уже очікує активації, але ще не має можливості увійти до системи;

- active - означає, що користувач прийняв запрошення, виконав активацію облікового запису та має повний доступ до функціоналу системи згідно з його роллю. Це основний робочий статус, у якому користувач виконує свої завдання в рамках платформи;

- not active - призначений для тих випадків, коли з певних причин необхідно тимчасово або постійно призупинити доступ користувача до системи. Перехід у цей статус здійснюється за рішенням SuperAdmin або SuperUser. У статусі not active користувач повністю втрачає доступ до платформи, а всі його підлеглі (у разі

наявності таких) також не можуть виконувати жодних дій у системі. Цей механізм дозволяє забезпечити контроль над доступами у разі зміни обставин. При цьому користувач зі статусом not active може бути активований у будь-який момент, що забезпечує гнучкість у керуванні обліковими записами.

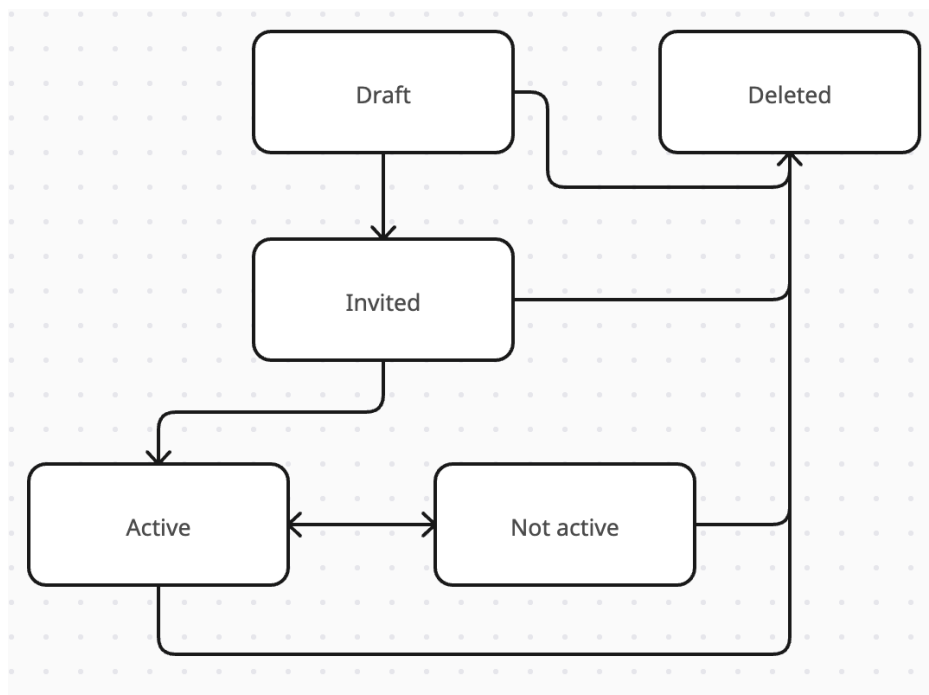


Рисунок 3.11 – Схема статусів створеного користувача з роллю Admin

Така система статусів дозволяє організувати чіткий контроль над користувачами, забезпечуючи підготовчий етап перед активацією, можливість швидкого введення нового користувача в роботу, а також механізм тимчасового або постійного призупинення доступу. Це гарантує безпеку, структурованість і гнучкість у роботі системи, що є важливим аспектом для управління компаніями в рамках платформи.

Можемо підсумувати, ми детально проаналізували функціональну схему роботи системи, яка забезпечує взаємодію між основними її компонентами: фронтендом, бекендом і базою даних. Розглянуто механізм обміну даними між цими складовими, починаючи від відправлення запитів користувачем через інтерфейс, їх обробки на серверній частині та закінчуючи збереженням і отриманням даних з бази даних. Таким чином, функціональна схема роботи

системи демонструє її надійність, структурованість та здатність гнучко адаптуватися до потреб компанії. Взаємодія між компонентами реалізована таким чином, щоб забезпечити високу продуктивність, безпеку передачі даних та зрозумілість усіх етапів роботи системи для кінцевих користувачів і адміністраторів.

3.4 Забезпечення авторизованого доступу

Ґрунтуючись на вимогах до функціональності, окрему увагу потрібно приділити процесу аутентифікації, який захищає дані користувачів і забезпечує безпечну взаємодію з системою. Для авторизації використовується JWT (JSON Web Token), який дозволяє безпечно передавати інформацію про користувача між клієнтом і сервером. Аутентифікація користувачів реалізується через протокол OAuth2, що забезпечує високий рівень захисту особистих даних. Авторизація є важливою складовою сучасних інформаційних систем, що працюють із персональними або конфіденційними даними. Її головна мета — гарантувати, що доступ до ресурсів і функцій системи отримують лише ті користувачі, які мають відповідні права. Вона забезпечує захист від несанкціонованого доступу, підвищує загальний рівень безпеки системи та дозволяє реалізувати чіткий контроль доступу до критичної інформації.

Процес авторизації починається з аутентифікації користувача, під час якої він вводить свої облікові дані, такі як логін і пароль. Після успішної перевірки цих даних сервер генерує спеціальні токени доступу, які служать для підтвердження його ідентичності протягом певного часу. Ці токени, створюються у форматі JWT (JSON Web Token) — сучасному стандарті для безпечної передачі даних між клієнтом і сервером. JWT складається з трьох частин: заголовка, корисного навантаження та підпису. Заголовок містить інформацію про тип токена і використовуваний алгоритм підпису, корисне навантаження включає дані про

користувача, наприклад, його ідентифікатор або роль, а підпис забезпечує цілісність та достовірність токена. Токен підписується за допомогою секретного ключа, що дозволяє серверу потім перевірити його автентичність(рис 3.12).

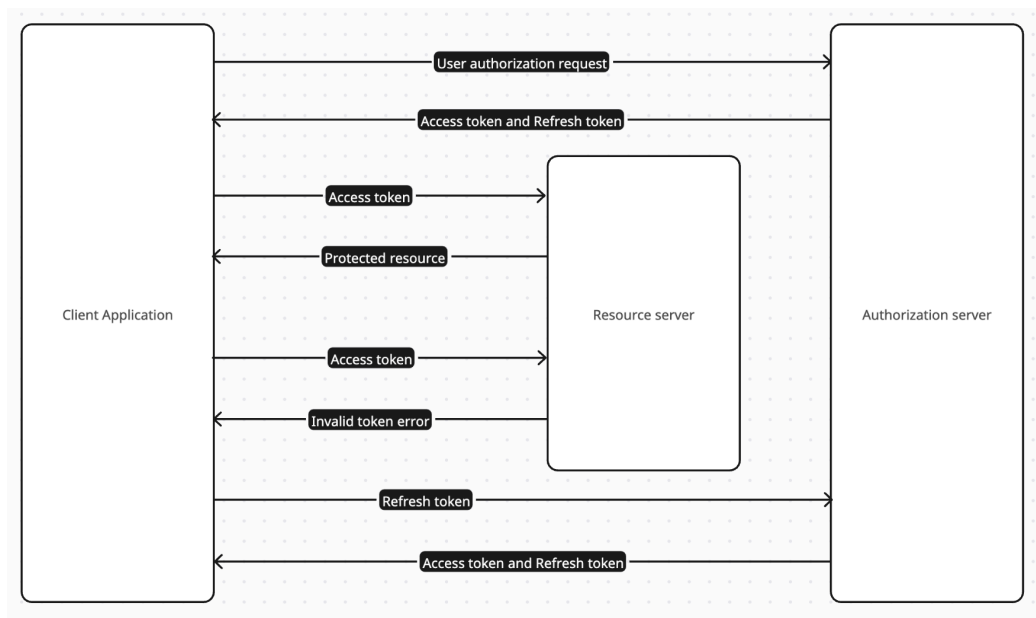


Рисунок 3.12 – Схема роботи Access token та Refresh token

Після отримання токена клієнт включає його в кожний запит до серверу, наприклад, додаючи його до HTTP-заголовка Authorization. Сервер, у свою чергу, перевіряє токен, щоб переконатися, що він дійсний і має відповідні права доступу. Якщо токен успішно проходить перевірку, сервер виконує запит і надає доступ до необхідних ресурсів. Однією з важливих властивостей токенів є їхній обмежений термін дії, що знижує ризики, пов'язані з їхньою компрометацією. Однак, коли термін дії токена доступу (access token) закінчується, використовується токен оновлення (refresh token), який дозволяє отримати новий токен доступу без необхідності повторної аутентифікації. Refresh token зазвичай має триваліший термін дії, ніж access token, і використовується лише для взаємодії із спеціальними кінцевими точками сервера.

JWT токени є зручними завдяки своїй простоті та автономності. Вони не потребують збереження стану на сервері, оскільки вся необхідна інформація закодована всередині токена. Це значно спрощує масштабування системи,

особливо в умовах розподілених архітектур. Проте важливо зазначити, що JWT не шифрує дані, а лише підписує їх, тому їхній вміст може бути переглянутий будь-ким, хто має до них доступ. Саме тому в токенах не слід зберігати конфіденційну інформацію. Авторизація також дозволяє керувати доступом на основі ролей. Наприклад, токени можуть містити інформацію про роль користувача, таку як SuperAdmin, Admin, Backoffice і тд, що дозволяє серверу надавати доступ до різних функцій залежно від цієї ролі. Для підвищення безпеки токени повинні зберігатися у захищених місцях. Наприклад, refresh token зазвичай зберігається у HTTP-only cookies, щоб уникнути їхнього викрадення через JavaScript.

Таким чином, авторизація забезпечує не лише захист даних, але й комфортну взаємодію користувача із системою. Використання таких механізмів, як JWT, а також токенів доступу і оновлення, дозволяє створити надійну, масштабовану та зручну систему авторизації, яка відповідає вимогам сучасних веб- і мобільних застосунків.

3.5 Висновки до розділу 3

Проаналізувавши впровадження системи, можемо зробити висновки, що у процесі реалізації ми створили базову структуру проекту, яка забезпечує модульність, масштабованість і зручність подальшого розвитку. Особливу увагу було приділено взаємодії з серверною частиною через API-запити, що гарантує надійність передачі даних та авторизацію доступу та забезпечує ефективну комунікацію між клієнтською та серверною частинами. Це дозволяє системі функціонувати як єдиний злагоджений механізм, готовий до масштабування та інтеграції нових функцій.

Ми розглянули налаштування глобальних стилів та компонентів інтерфейсу. Вибір кольорової палітри та шрифтів, зокрема використання Raleway, дозволив створити сучасний та стильний вигляд додатку. Механізм локалізації через react-

intl забезпечив багатомовність додатка, дозволяючи зручно перемикатися між мовами. У результаті ми досягнули функціональності та естетики, що відповідають сучасним вимогам.

Також, детально проаналізували функціональну схему роботи системи, яка забезпечує взаємодію між основними її компонентами: фронтендом, бекендом і базою даних. Розглянуто механізм обміну даними між цими складовими, починаючи від відправлення запитів користувачем через інтерфейс, їх обробки на серверній частині та закінчуючи збереженням і отриманням даних з бази даних. Функціональна схема роботи системи демонструє її надійність, структурованість та здатність гнучко адаптуватися до потреб компанії.

На останок, ми зрозуміли, що авторизація забезпечує не лише захист даних, але й комфортну взаємодію користувача із системою. Використання таких механізмів, як JWT, а також токенів доступу і оновлення, дозволяє створити надійну, масштабовану та зручну систему авторизації, яка відповідає вимогам сучасних веб- і мобільних застосунків.

4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПІДТРИМКИ КЛІЄНТСЬКИХ ЗАПИТІВ

4.1 Функціональні можливості додатків, порівняння з аналогами

З урахуванням функціональних вимог до системи, розглянемо чим саме наша система є особливою в порівнянні з іншими аналогами. Однією з ключових переваг нашої системи є її унікальні функціональні можливості, які відрізняють її від існуючих аналогів. У багатьох системах управління персоналом модулі статистики обмежені лише базовими функціями, такими як відображення загальної кількості відпусток або робочих годин. Наша система виходить далеко за ці рамки, надаючи гнучкі та автоматизовані інструменти для обчислення, моніторингу та управління робочими процесами.

Гнучке налаштування графіків роботи. Більшість аналогів дозволяють задавати лише стандартні шаблони робочого часу, наприклад, 8-годинний робочий день. У нашій системі Admin або Vascoffice може створювати індивідуальні графіки роботи для кожного співробітника. Наприклад, для працівника з частковою зайнятістю можна встановити 5 годин роботи на день, а для фрілансерів — взагалі нерегулярний графік. Також можна обриту індивідуальні дні роботи, наприклад графік працює лише по понеділках та середам, в усі інші дні він не працює. Всі ці дані інтегруються зі статистикою, автоматично враховуючи відхилення від звичайного графіка(рис 4.1).

Якщо співробітник бере лікарняний, відпустку або відгул, система автоматично коригує розрахунки. Тобто уся статистика вираховується без необхідності ручного втручання. Статистика дозволяє компаніям контролювати та оцінювати ефективність роботи співробітників, планувати робочі процеси, а також прогнозувати витрати. Наприклад, на основі зібраних даних про використання відпусток, лікарняних чи відгулів можна аналізувати тенденції відсутності персоналу та оптимізувати графіки роботи, щоб уникнути браку працівників у критично важливі періоди.

Attendance register

Register • Requests • Absences • **Planning** • Analyses

Work shifts

Type	Hours	Days of the week	
Afternoon	14:00 - 18:00	Mo Tu We Th Fr Sa Su	✓ ✕
Morning	08:00 - 12:00	Th, Fr, Sa, Su	✎ ✕
Regular	08:00 - 12:00 13:30 - 18:00	Tu, We, Th, Fr	✎ ✕

Departments

Name	
department 1	✓ ✕
department 2	✎ ✕

Employee schedules

User	Shift	Shift period	Department	
viktor bck	Future shift enabled Off On	Current Afternoon	19/11/2024 DD/MM/YYYY	Name ✓ ✕
geralt rereere	Current Afternoon	07/11/2024 --	ITAGRI	✎ ✕

Рисунок 4.1 – Модуль створення графіків, відділів та робочих змін

У нашій системі статистика реалізована через інтеграцію кількох ключових функціональних компонентів. Як зазначалось раніше, при створенні нового користувача адміністратор має можливість задавати кількість доступних днів для відпустки, лікарняних або відгулів. Наприклад, якщо співробітнику встановлено 20 днів відпустки на рік, ця кількість фіксується у базі даних. Надалі кожен запит на відпустку віднімається від початкового ліміту, а оновлена інформація відображається у звітах адміністратора.

Розглянемо приклад із графіками роботи. Система дозволяє адміністратору призначати графік роботи користувачам. Припустимо, що для співробітника встановлено графік роботи з 5 робочими годинами на день. За умови, що в місяці є 22 робочі дні, загальна кількість годин роботи для цього співробітника розраховується за формулою:

$$H_{total} = H_d \cdot D_w \quad (4.1)$$

де H_d - кількість годин роботи на день;

D_w - кількість робочих днів у місяці.

Якщо співробітник взяв відпустку тривалістю 5 днів, та лікарняний один день ця кількість віднімається від загальної кількості робочих днів:

$$D_{actual} = D_w - D_v - D_i \quad (4.2)$$

де D_v - кількість днів відпустки;

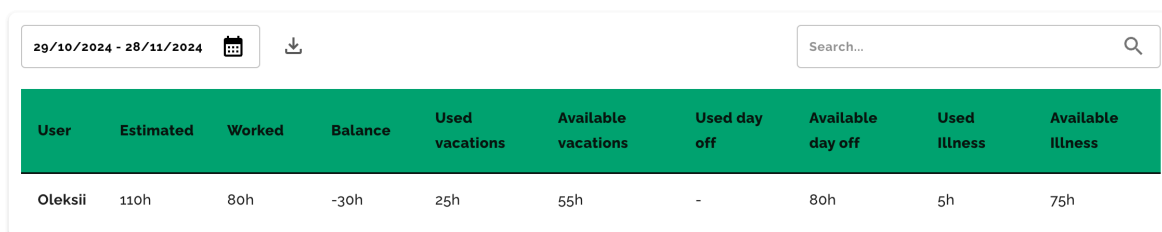
D_i - кількість днів лікарняного.

Загальна кількість годин роботи в такому випадку розраховується так:

$$H_{adjusted} = H_d \cdot D_{actual} \quad (4.3)$$

де $H_{adjusted}$ - скоригована кількість годин з урахуванням використаних днів на відгул, відпустку або лікарняний.

Таким чином, система автоматично враховує всі зміни у графіках роботи та використанні відпусток, оновлюючи статистичні дані в реальному часі. Це дозволяє як працівникам, так і адміністраторам бачити актуальну інформацію про доступний баланс відпусток, фактично відпрацьований час і відповідність затвердженому графіку роботи(рис 4.2).



User	Estimated	Worked	Balance	Used vacations	Available vacations	Used day off	Available day off	Used illness	Available illness
Oleksii	110h	80h	-30h	25h	55h	-	80h	5h	75h

Рисунок 4.2 – Секція з таблицею статистики системи

Перейдемо до наступної особливості - інтерактивний робочий календар. Цей модуль розроблений таким чином, щоб надати адміністраторам і працівникам максимально наочний і гнучкий спосіб відслідковування робочого процесу. Календар за замовчуванням відкривається на поточному місяці, відображаючи всі дні цього періоду. Інтуїтивно зрозумілий інтерфейс дозволяє легко переключатися між місяцями, використовуючи стрілки або меню вибору. Кожен день у календарі поділений на блоки, які відповідають графікам роботи, закріпленим за цим днем(рис 4.3).

У кожному дні календаря є окремі секції, що відображають активні графіки роботи для цього дня. Наприклад, це може бути ранкова, денна чи ранкова зміна. У межах кожного графіка відображається список співробітників, які працюють за ним у зазначений день. Якщо співробітник доданий до декількох змін, його ім'я відображається у відповідних блоках. Таким чином, адміністратор може швидко побачити розподіл робочої сили.

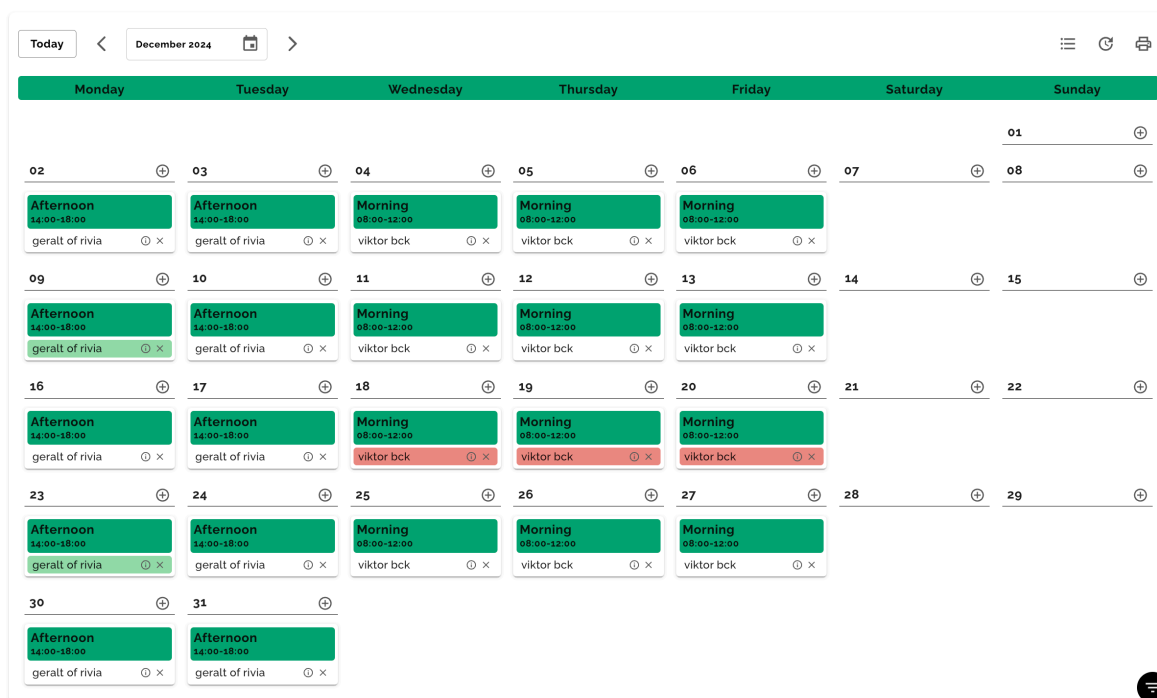


Рисунок 4.3 – Інтерактивний робочий календар

Особливістю календаря є інтуїтивний спосіб позначення відсутніх працівників. Якщо співробітник запланував відпустку, лікарняний або взяв відгул, це відображається через кольорове маркування:

- зелений колір – відпустка;
- жовтий колір – відгул;
- червоний колір – лікарняний.

Таке маркування дозволяє адміністраторам швидко оцінити, хто буде відсутній у певний день, і відкоригувати розподіл графіків у разі потреби. Ця функція зменшує ризик помилок, пов'язаних із браком робочої сили в конкретний день.

Проаналізувавши особливості нашої системи, наша система орієнтована на автоматизацію та зручність. Вона допомагає не тільки заощадити час адміністратора, але й підвищити ефективність управління персоналом. Це робить нашу систему не просто конкурентоспроможною, але й такою, що виділяється серед аналогів завдяки своїй продуманій структурі, високій продуктивності та інноваційному підходу до управління статистикою.

4.2 Дослідження механізмів повідомлення про статус запитів

З опорою на опис бізнес-логіки, в нашій системі передбачений механізм повідомлення користувачу про статус запитів різного типу. Механізм повідомлення про статус запитів у системі реалізується через три основні канали комунікації: сокети, пуш-повідомлення та emails. Кожен із цих методів має свої переваги та застосовується залежно від контексту, забезпечуючи надійне та оперативне інформування користувачів про зміни статусу їхніх запитів, таких як відпустка, лікарняний чи відгул.

Перший і найоперативніший механізм — це повідомлення через сокети. Для цього використовується технологія WebSocket, яка дозволяє встановлювати постійне двостороннє з'єднання між клієнтською частиною і сервером. Наприклад, коли користувач відкриває веб- або мобільний додаток, між ним і сервером автоматично встановлюється WebSocket-з'єднання. Це з'єднання дозволяє серверу миттєво надсилати повідомлення про зміну статусу запиту без потреби у перезавантаженні сторінки. У додатку реалізовані спеціальні слухачі подій, які отримують ці повідомлення, наприклад, про те, що запит на відпустку було затверджено чи відхилено. Як тільки сервер надсилає сигнал про зміну статусу, відповідна інформація відображається у реальному часі, забезпечуючи максимальну швидкість та актуальність даних.

Другий канал комунікації — це пуш-повідомлення, які особливо зручні для мобільних пристроїв. Якщо користувач неактивний у системі, але має увімкнені пуш-сповіщення, система надсилає короткі повідомлення на його пристрій. Наприклад, якщо статус запиту змінився, користувач одразу отримає сповіщення на свій смартфон із текстом на кшталт: "Ваш запит на відпустку було затверджено"(рис 4.4).



Рисунок 4.4 – Push-повідомлення за підтвердженням запитом на відпустку

Третім універсальним каналом комунікації є email-сповіщення, які використовуються для надсилання деталізованої інформації. Електронна пошта забезпечує зручний спосіб збереження інформації про запити для подальшого використання, що робить її незамінною в корпоративному середовищі. Наприклад, якщо запит на відпустку було затверджено чи відхилено, користувач отримує лист із зазначенням деталей: дати, типу запиту (відпустка, лікарняний, відгул). Такий формат дозволяє створювати історію дій, яку можна легко переглянути в поштової скриньці, що особливо корисно для офіційного підтвердження рішень. Окрім цього, email використовується для інформування про події, які потребують уваги або дій від користувача. Наприклад, якщо запит знаходиться в стані "на розгляді" протягом певного часу, система автоматично відправляє нагадування із закликом до дії, що спрощує взаємодію між користувачами та адміністраторами. Крім того,

при створенні нового запиту користувач отримує підтвердження про те, що його запит успішно зареєстровано, разом із унікальним ідентифікатором, який може бути використаний для подальшого відстеження.

Отже, можемо зробити висновок, що злагоджена робота механізмів сповіщення про якісь дії користувачу забезпечує максимальну ефективність та зручність для користувачів. Сокети гарантують миттєвість push-повідомлення забезпечують зручність, а email надає офіційну та деталізовану інформацію. Завдяки такій інтеграції, система задовольняє різні потреби користувачів, підвищуючи рівень залученості та оперативності взаємодії.

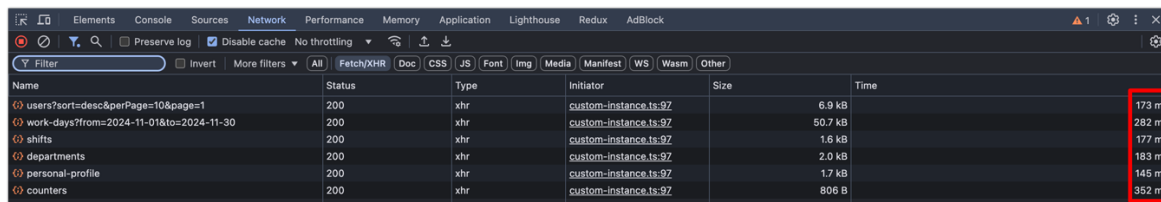
4.3 Аналіз швидкодії та стабільності системи

Розглянемо питання тестування нашої системи. Аналіз швидкодії та стабільності системи є важливою складовою, яка дозволяє оцінити, наскільки ефективно система виконує свої функції за умов реального використання. Цей аналіз включає в себе оцінку продуктивності при різних навантаженнях, визначення точок можливого зниження швидкодії, а також тестування на стійкість до помилок і збоїв. Для розкриття цього пункту доцільно розглянути кілька аспектів.

Час відгуку серверу, що є основним критерієм швидкодії — це час, необхідний серверу для обробки запиту та повернення відповіді. У нашій системі сервер на базі Nest.js оптимізовано для роботи з асинхронними запитами, що дозволяє обробляти кілька задач одночасно без блокування основного потоку. У середньому час відгуку на базові запити, такі як отримання графіків чи статусу користувачів, не перевищує 200 мс(рис 4.5).

У системі реалізовано централізоване логування помилок через сервіси типу Sentry. Це дозволяє миттєво виявляти та виправляти проблеми, що виникають як на стороні клієнта, так і на стороні сервера. Також додаток налаштований на

автоматичне перезавантаження серверних процесів у випадку збоїв, що забезпечує мінімальний час простою. Для збереження даних використовуються транзакції в базі даних, які гарантують консистентність навіть у випадку непередбачених збоїв.



Name	Status	Type	Initiator	Size	Time
users?sort=desc&perPage=10&page=1	200	xhr	custom-instance.ts:97	6.9 kB	173 ms
work-days?from=2024-11-01&to=2024-11-30	200	xhr	custom-instance.ts:97	50.7 kB	282 ms
shifts	200	xhr	custom-instance.ts:97	1.6 kB	177 ms
departments	200	xhr	custom-instance.ts:97	2.0 kB	183 ms
personal-profile	200	xhr	custom-instance.ts:97	1.7 kB	145 ms
counters	200	xhr	custom-instance.ts:97	806 B	352 ms

Рисунок 4.5 – Network вкладка консолі браузера з часом швидкості запитів

Розглянемо етап тестування на навантаження системи, що дозволяє оцінити її здатність обробляти великий обсяг запитів за умов пікового навантаження. Це тестування проводиться для забезпечення стабільної роботи додатка навіть у випадках значного збільшення кількості користувачів або обробки складних операцій. Метою такого тестування є виявлення потенційних вузьких місць у продуктивності, визначення максимально допустимого навантаження та оцінка стійкості системи до можливих збоїв. У процесі тестування моделюються реальні сценарії використання, включаючи одночасне надсилання запитів, обробку великого обсягу даних, генерацію звітів або масове виконання бізнес-логіки. Під час тестування система може піддаватися зростаючому навантаженню, що дає змогу визначити точку, в якій її продуктивність починає погіршуватися. Наприклад, якщо при певній кількості одночасних користувачів час відповіді системи починає значно збільшуватися, це сигналізує про необхідність оптимізації коду або масштабування інфраструктури.

Основними показниками, які аналізуються під час тестування, є:

- Requests Per Second (RPS) — кількість запитів, які система може обробити за секунду. Цей показник дозволяє оцінити пропускну здатність системи;
- Response Time — середній час відповіді сервера на запити. Він демонструє, наскільки швидко система реагує на запити користувачів під навантаженням;
- Resource Usage — використання системних ресурсів (процесор, пам'ять, дискова підсистема) при виконанні навантаження.

Результати навантажувального тестування зазвичай представлені у вигляді графіків, які візуалізують залежність ключових метрик від рівня навантаження. Результат такого аналізу проілюстровано на рисунку 4.6.

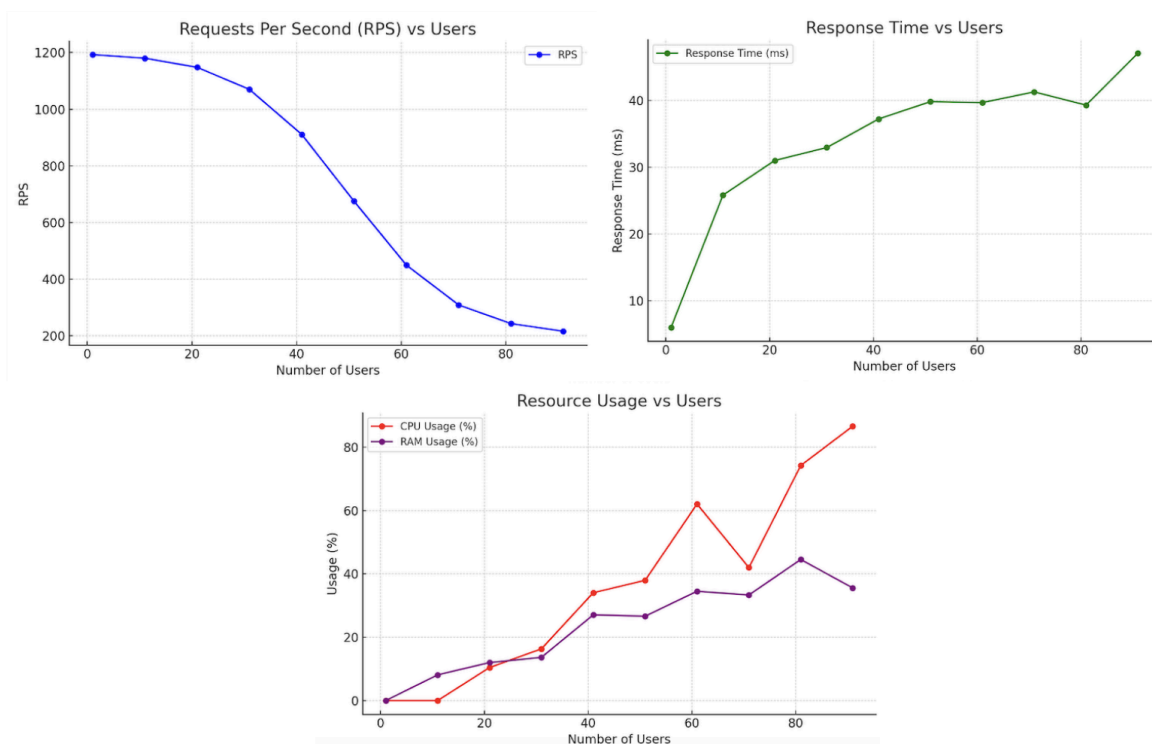


Рисунок 4.6 – Графіки результатів навантаження на систему

Тепер давайте перевіримо якість наших сторінок. З цим нам допоможе Lighthouse(рис 4.7). Lighthouse - це відкритий вебінструмент, розроблений командою Google Chrome, який допомагає перевірити якість веб-сторінки. Він використовується для оцінки різних аспектів веб-сторінки:

- продуктивність - аналізує швидкість завантаження сторінки, розмір і кешування ресурсів, ефективність виконання коду та інші фактори, що впливають на продуктивність. Він надає рекомендації щодо оптимізації, такі як мініфікація та стиснення файлів, використання кешування, видалення непотрібного коду та інше;
- доступність - перевіряє веб-сторінку на відповідність нормам доступності. Він перевіряє наявність атрибутів alt для зображень, правильне використання заголовків, наявність текстових описів для відео та аудіо елементів, контрастність

кольорів та інші аспекти, що забезпечують доступність для людей з різними обмеженнями;

- найкращі практики - аналізує веб-сторінку на предмет дотримання найкращих практик розробки. Він перевіряє правильне використання HTML-тегів, уникнення блокуючого JavaScript на першому етапі завантаження, використання безпечних протоколів;

- SEO - перевіряє веб-сторінку на оптимізацію для пошукових систем. Він аналізує наявність метатегів, унікальність заголовків, використання правильних URL-адрес, структуровані дані та інші фактори, які впливають на SEO. За допомогою Lighthouse можна отримати рекомендації щодо покращення показників пошукової оптимізації.

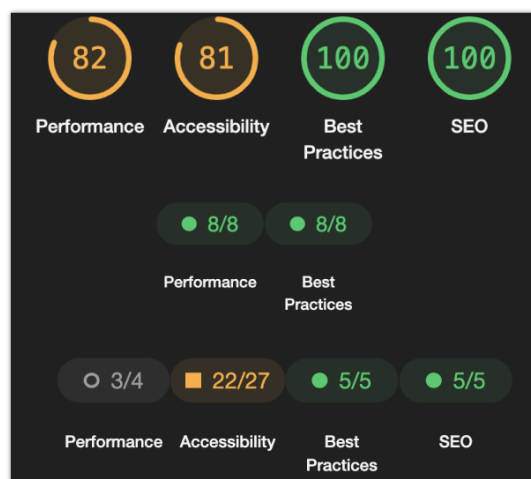


Рисунок 4.7 - Аналіз тестів Lighthouse

Враховуючи усі аналізи тестів, можемо впевнитись, що успішне тестування системи продемонструвало її здатність ефективно обробляти навантаження в умовах реального використання. Проведені навантажувальні випробування довели, що система здатна підтримувати стабільну швидкість відповіді навіть при значній кількості одночасних запитів. Це свідчить про надійність архітектури, оптимізовану бізнес-логіку та правильно налаштовану інфраструктуру. Виявлені потенційні вузькі місця були успішно усунені, що забезпечує високу продуктивність системи та позитивний досвід для кінцевих користувачів.

4.4 Висновки до розділу 4

В цьому розділі ми проаналізували особливості нашого додатку. Вона допомагає не тільки заощадити час адміністратора, але й підвищити ефективність управління персоналом. Це робить нашу систему не просто конкурентоспроможною, але й такою, що виділяється серед аналогів завдяки своїй продуманій структурі, високій продуктивності та інноваційному підходу до управління статистикою.

Злагоджена робота механізмів сповіщення про якісь дії користувачу забезпечує максимальну ефективність та зручність для користувачів. Сокети гарантують миттєвість push-повідомлення забезпечують зручність, а email надає офіційну та деталізовану інформацію. Завдяки такій інтеграції, система задовольняє різні потреби користувачів, підвищуючи рівень залученості та оперативності взаємодії.

Враховуючи усі аналізи тестів, можемо впевнитись, що успішне тестування системи продемонструвало її здатність ефективно обробляти навантаження в умовах реального використання. Проведені навантажувальні випробування довели, що система здатна підтримувати стабільну швидкість відповіді навіть при значній кількості одночасних запитів.

ВИСНОВКИ

У ході дослідження, яке було спрямоване на аналіз актуальної проблеми управління клієнтськими запитами шляхом аналізу можливостей існуючої інформаційної системи, яка базується на сучасних технологіях мобільної та веброзробки. Завдяки функціоналу цієї системи вдалося досягти автоматизації процесів обробки клієнтських запитів, що суттєво підвищує ефективність бізнес-процесів і якість обслуговування клієнтів.

В процесі виконання проведено детальний аналіз існуючих рішень на ринку, виявлено їхні переваги та недоліки. На основі цього визначено ключові вимоги до системи, що включають підтримку мобільних і вебплатформ, зручний інтерфейс, високий рівень безпеки та гнучкість у налаштуванні.

В рамках роботи ми змогли дослідити структуру та модель існуючої системи. Система базується на багаторівневій архітектурі, що забезпечує розподіл обов'язків між фронтендом, бекендом та базами даних. В деталях проаналізовано бізнес-логіку, яка дозволяє створювати запити, керувати робочими графіками, відстежувати час роботи та отримувати аналітику. Вибір технологій у системі, таких як React, React Native і Nest.js, забезпечує її відповідність сучасним стандартам, високу продуктивність і масштабованість.

Також було детально розглянуто функціональна схема роботи системи, яка включає всі ключові процеси: авторизацію, управління графіками, обробку запитів на відпустки чи лікарняні та отримання статистики.

Під час дослідження проведено аналіз функціональних можливостей системи, порівняння її з аналогами. Виявлено, що система перевершує конкурентів завдяки гнучкому налаштуванню робочих графіків, інтеграції статистичних модулів і механізмів швидкого обміну інформацією через push-повідомлення, email та сокети.

Для оцінки стабільності та продуктивності проведено тестування на навантаження, яке підтвердило здатність системи обробляти велику кількість

одночасних запитів без значних втрат у швидкодії. Виявлені недоліки були усунуті, що підтверджує високу стабільність розробленої архітектури.

Узагальнюючи результати, можна стверджувати, що досліджена система повністю відповідає сучасним вимогам до управління клієнтськими запитами. Вона дозволяє знизити навантаження на персонал, автоматизувати обробку клієнтських запитів, забезпечити прозорий контроль робочого часу та оптимізувати ресурси компанії. Інтеграція мобільного та вебінтерфейсів у системі гарантує зручність для користувачів і забезпечує можливість взаємодії з системою у будь-який час. Система демонструє високу ефективність у своїй предметній області та є прикладом надійного і продуктивного інструменту для бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Структура та функції інформаційних систем управління. URL: <https://mis.ptngu.com/lekc/07.pdf> (Дата звернення 04.11.24 р.).
2. Ситнік Б. Т. Основи інформаційних систем і технологій: Навч. посібник. – Харків: УкрДУЗТ, 2019. – 175 с.
3. Остапчук Я. Ю. Аналіз переваг та недоліків існуючих інформаційних систем для покращення управління підприємством [Електронний ресурс] / Я. Ю. Остапчук, А. О. Азарова // Матеріали LI науково-технічної конференції підрозділів ВНТУ, Вінниця, 31 травня 2022 р. – Електрон. текст. дані. – 2022. – Режим доступу: <https://conferences.vntu.edu.ua/index.php/all-fm/all-fm-2022/paper/view/14927>
4. Азарова А.О, Присяжнюк М.В. Обґрунтування вибору оптимального програмного продукту управління збутом на підприємстві засобами лінійної моделі зважених сум. Глобальні та національні проблеми економіки. 2018. № 22. С. 10–14. URL: <http://global-national.in.ua/archive/22-2018/191.pdf> (Дата звернення 10.04.22 р.).
5. Азарова А. О., Ткачук А., Лозан Б. О. Інформаційні засоби підвищення ефективності управління персоналом на вітчизняних підприємствах. Науковий вісник Ужгородського університету. 2018. № 18. С. 6–9.
6. SCALA HR. Сайт ресурсу компанії SCALA HR. URL: <https://www.scala.hr/> (Дата звернення 20.10.24 р.).
7. HURMA. Сайт ресурсу компанії HURMA. URL: <https://hurma.ua/> (Дата звернення 20.10.24 р.).
8. SAP ERP. Сайт ресурсу компанії. URL: <https://www.sap.com/ukraine/products/erp.html>
9. Cappelli P. HBR's 10 Must Reads on Performance Management: Harvard Business Review Press. 2023. p.192
10. Bordeaux J. Identifying Critical IT Products and Services: RAND Corporation. 2022. p. 52

11. REACTJS. Сайт ресурсу REACTJS. URL: <https://uk.legacy.reactjs.org/> (Дата звернення 04.11.24 р.).

12. VUEJS. Сайт ресурсу VUEJS. URL: <https://ua.vuejs.org/guide/introduction.html#what-is-vue> (Дата звернення 04.11.24 р.).

13. ANGULARJS. Сайт ресурсу ANGULARJS. URL: <https://angular.dev/overview> (Дата звернення 20.10.24 р.).

14. REACT NATIVE. Сайт ресурсу REACT NATIVE. URL: <https://reactnative.dev/docs/getting-started> (Дата звернення 20.10.24 р.).

15. FIREBASE. Сайт ресурсу FIREBASE. URL: <https://firebase.google.com/docs/firestore> (Дата звернення 20.10.24 р.).

16. MONGODB. Сайт ресурсу MONGODB. URL: <https://www.mongodb.com/docs> (Дата звернення 20.10.24 р.).

17. Сайт компанії Geniusee. URL <https://geniusee.com/single-blog/monolith-vs-microservices-vs-serverless-architecture> (Дата звернення 16.11.24 р.).

18. Особливості роботи мікросервісної архітектури. Сайт <https://microservices.io/> (Дата звернення 16.11.24 р.).

19. Визначення безсерверної архітектури. Сайт https://aws.amazon.com/lambda/serverless-architectures-learn-more/?nc1=h_ls (Дата звернення 16.11.24 р.).

20. Визначення РБД. Сайт <https://cloud.google.com/learn/what-is-a-relational-database> (Дата звернення 16.11.24 р.).

21. Визначення NoSQL. Сайт <https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data> (Дата звернення 16.11.24 р.).

22. React Router. Сайт ресурсу React Router. URL: <https://reactrouter.com/home> (Дата звернення 25.11.24 р.).

23. Киричек Г.Г., Тягунова М.Ю., Балабуха О.М. Логістична вебплатформа з використанням React.js. Міжнародна науково-практична конференція «Особливості інноваційного розвитку у сфері техніки: порівняльний досвід

України та Європейського Союзу», м. Влоцлавек, Республіка Польща, 6–7 серпня 2023. С. 9-12.