

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебзастосунок для керування робочим часом»

Виконав студент групи ІПЗ-20-4
МЕЛЬНИЧЕНКО Давид Вадимович

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **МЕЛЬНИЧЕНКО Давид Вадимович**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Вебзастосунок для керування робочим часом»**,
затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **автоматизація управління процесами керування та обліку робочого часу з елементами системи управління персоналом**
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Тетяна Анатоліївна	27.02.2024р.	28.03.2024р.
2	Вакалюк Тетяна Анатоліївна	29.03.2024р.	15.05.2024р.
3	Вакалюк Тетяна Анатоліївна	16.05.2024р.	01.06.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебзастосунок для керування робочим часом з елементами системи управління персоналом та пояснювальної записки. Пояснювальна записка до випускної роботи містить 97 сторінок, 39 ілюстрацій та 24 таблиць.

Метою даної дипломної роботи є розробка вебзастосунок для управління робочим часом з елементами системи управління персоналом, який забезпечить автоматизацію, оптимізацію та спрощення процесів обліку та планування робочого часу. Цей вебзастосунок має на меті забезпечити ефективне використання робочих ресурсів та полегшити керування завданнями і проектами, надаючи користувачам зручний та інтуїтивний інтерфейс.

Архітектура відіграє ключову роль у реалізації системи, забезпечуючи високу масштабованість для подальшого розвитку проекту. Надійність і гнучкість архітектури дозволяє розширювати проект у майбутньому, перетворюючи його на комплексну систему управління персоналом. Це забезпечує можливість інтеграції нових функцій та модулів, які підвищують функціональність та ефективність системи.

Особливу увагу приділено зручності користувацького інтерфейсу, який дозволяє користувачам легко взаємодіяти з елементами системи, забезпечуючи їм приємний досвід роботи. Інтерфейс розроблений таким чином, щоб мінімізувати час навчання користувачів та максимізувати продуктивність їхньої роботи.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, АРХІТЕКТУРА, ПЕРСОНАЛ, УПРАВЛІННЯ, ЧАС.

					ІПЗ.КР.Б – 121 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Мельниченко Д. В.			Вебзастосунок для керування робочим часом	Літ.	Арк.	Аркушів
Керівник		Вакалюк Т. А.					3	97
Рецензент		Власенко О. В.				Житомирська політехніка, група ІПЗ-20-4		
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's qualification work consists of a web application for time management with elements of a human resource management system and an explanatory note. The explanatory note to the final thesis contains 97 pages, 39 illustrations, and 24 tables.

The goal of this thesis is to develop a web application for time management with elements of a human resource management system, which will provide automation, optimization, and simplification of time tracking and planning processes. This web application aims to ensure efficient use of work resources and facilitate task and project management, offering users a convenient and intuitive interface.

The architecture plays a key role in the implementation of the system, providing high scalability for further project development. The reliability and flexibility of the architecture allow the project to be expanded in the future, turning it into a comprehensive human resource management system. This ensures the possibility of integrating new functions and modules that enhance the system's functionality and efficiency.

Special attention is given to the convenience of the user interface, which allows users to easily interact with system elements, providing them with a pleasant user experience. The interface is designed to minimize user training time and maximize their productivity.

KEYWORDS: WEB APPLICATION, ARCHITECTURE, PERSONNEL, MANAGEMENT, TIME.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ РОБОЧИМ ЧАСОМ.....	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту	12
1.3. Визначення архітектури вебзастосунок.....	19
1.3.1. Визначення архітектури серверної частини вебзастосунок	19
1.3.2 Визначення структури клієнтської частини вебзастосунок	25
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	28
Висновки до розділу 1	31
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ КЕРУВАННЯ РОБОЧИМ ЧАСОМ.....	33
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	33
2.2. Розробка бази даних системи.....	41
2.3. Проектування та реалізація алгоритмів роботи системи.....	46
2.4. Реалізація застосунок веб-орієнтованої системи керування робочим часом	50

Висновки до розділу 2	61
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ ДЛЯ КЕРУВАННЯ РОБОЧИМ ЧАСОМ.....	63
3.1. Порядок встановлення та налаштування параметрів системи	63
3.2. Структура інтерфейсу веб-орієнтованої системи керування робочим часом	64
3.3. Тестування роботи програмного комплексу	76
Висновки до розділу 3	80
ВИСНОВКИ.....	82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	84
ДОДАТКИ.....	87

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

HR – людські ресурси

HRIS – інформаційна система керування персоналом

HRM – управління персоналом

PBAC – контроль доступу на основі дозволів

RP – реактивне програмування

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Сучасні підприємства, незалежно від їхньої галузі, стикаються з необхідністю ефективного управління робочим часом та персоналом. З розвитком технологій та змінами в бізнес-середовищі, автоматизація процесів обліку та планування робочого часу стає критично важливою. Використання вебзастосунків для цих завдань дозволяє підприємствам оптимізувати внутрішні процеси, підвищити продуктивність і забезпечити точність обліку робочого часу. В умовах сучасної конкуренції на ринку, такі інструменти стають стратегічно важливими для підтримання ефективності та конкурентоспроможності компаній.

Сучасні компанії потребують інструментів, які дозволяють легко планувати та контролювати робочий час, забезпечують автоматизацію підрахунку відпрацьованих годин, управління запитами на відпустки та лікарняні, а також інші адміністративні процедури. Інтеграція всіх цих функцій у одному вебзастосунку надає підприємствам значні переваги, включаючи економію часу, підвищення точності даних та поліпшення продуктивності працівників.

Мета і завдання роботи. Метою даної дипломної роботи є розробка вебзастосунку для управління робочим часом з елементами системи управління персоналом, який забезпечить автоматизацію, оптимізацію та спрощення процесів обліку та планування робочого часу. Програма має відповідати основним адміністративним потребам компанії, сприяючи підвищенню ефективності її функціонування та створенню початкової бази для подальшого розвитку проекту як системи управління персоналом.

Основні завдання роботи:

- провести аналіз існуючих рішень для планування та обліку робочого часу, а також напрямків їх автоматизації;
- визначити архітектуру та загальну структуру вебзастосунку;
- зробити вибір засобів та інструментів для реалізації системи і її функціоналу;
- розробити програмне забезпечення системи;

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

- спроектувати структурні компоненти та алгоритми роботи системи;
- реалізувати програмний комплекс вебзастосунку для обліку та планування робочого часу з можливістю подальшого розширення до повноцінної системи управління персоналом.

Об'єктом дослідження є автоматизація управління процесами керування та обліку робочого часу з елементами системи управління персоналом.

Предметом дослідження є використання веб-орієнтованих технологій для розробки системи обліку та планування робочого часу в компанії, включаючи створення архітектури проекту та інструментів (наприклад, системи надсилання пошти), які зможуть задовольнити всі бізнес-потреби і надати можливість розвитку проекту як повноцінної системи управління персоналу.

Методи аналізу та інформаційна база дослідження. Для реалізації вебзастосунку використовувалися методи системного аналізу, методи об'єктно-орієнтованого проектування, веб-програмування та дизайну. Інформаційна база дослідження включає сучасні наукові публікації, технічну документацію та практичні посібники з розробки вебзастосунків.

Практичне значення одержаних результатів. Розроблений вебзастосунок може бути впроваджений у діяльність підприємств різних галузей для автоматизації обліку та планування робочого часу. Це дозволить підвищити ефективність управління робочими процесами, знизити витрати часу на адміністративні процедури та покращити продуктивність працівників. Система також забезпечує точність даних, що сприяє оптимізації внутрішніх процесів компаній та створенню основи для подальшого розвитку повноцінної системи управління персоналом.

Випускна робота складається з вступу, трьох розділів, висновків, списку використаних джерел із 31 найменування (3 стор.), 2 додатків (11 стор.), 24 таблиць та 39 ілюстрацій по тексту, з них 3 оформлені на окремій сторінці. Повний обсяг роботи становить 97 сторінок, з них 86 сторінок основного тексту.

Апробація. За результатами роботи взято участь в конференції [\[32\]](#).

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ РОБОЧИМ ЧАСОМ

1.1. Постановка задачі

В сучасному світі наявність програмних продуктів у будь-яких сферах діяльності значно покращує умови роботи та ефективність процесів. Це твердження справедливе і для інструментів, які допомагають компаніям легше виконувати роботу із управління персоналом. Завданням даної дипломної роботи є створення одного з таких інструментів за допомогою сучасних технологій програмування.

Розробка вебзастосунку для управління робочим часом з елементами системи управління персоналом дозволить автоматизувати рутинні завдання, пов'язані з плануванням та обліком робочого часу, а також з управлінням персоналом. Це включає наявність актуальної інформації про працівників із можливістю їх гнучкого оновлення, відслідковування робочого часу та відвідуваності, управління запитами на відпустки та лікарняні, а також виконання інших адміністративних процедур.

Для виконання поставлених завдань в повній мірі потрібно:

- створити гнучку і здатну до розширення архітектуру проекту, яка забезпечить стабільне і надійне виконання поставлених завдань. Архітектура повинна дозволяти легко додавати нові функціональні можливості без суттєвих змін існуючого коду;
- забезпечити швидкий і надійний обмін даними між усіма частинами програми. Це включає інтеграцію між серверною і клієнтською частинами вебзастосунку, а також можливість інтеграції з зовнішніми системами через API;
- створити адміністративну складову, щоб надати клієнтам можливість обмежити кількість функцій, доступних для кожного користувача. Це

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

забезпечить гнучке управління правами доступу та підвищить безпеку системи;

- розробити модуль налаштувань системи, щоб клієнт міг підлаштовувати роботу системи відповідно до власних потреб бізнесу. Це може включати налаштування робочих годин, політик відпусток, параметрів обліку тощо;
- створити модуль управління користувачами, який включатиме функції авторизації, перегляду, керування та профілі користувачів. Це дозволить забезпечити безпеку даних та ефективно управління користувачами системи;
- розробити модуль планування та управління часом, з можливістю зручного перегляду та відстеження робочого часу та відвідуваності працівників. Це допоможе оптимізувати розклад та підвищити ефективність використання робочого часу;
- створити модуль управління відпустками та лікарняними, щоб надати можливість легко відслідковувати ці фактори в роботі. Це дозволить автоматизувати процеси, пов'язані з обліком відпусток та лікарняних, що сприятиме зниженню адміністративного навантаження;
- створити модуль вихідних днів, який дозволить додавати в систему власні вихідні, свята чи скорочені дні, коли це буде потрібно компанії. Це додасть гнучкості системи і буде сприяти загальному;
- розробити модуль статистики та аналізу робочого часу, який забезпечить можливість формування звітів та в подальшому нарахування заробітної плати на основі точної інформації про робочий час. Це сприятиме прозорості процесу нарахування зарплати та значно спростить процес збирання інформації.

Основні етапи розробки проекту:

1. Аналіз області розробки та особливостей сфери управління персоналом.
 - 1.1. Постановка задачі і дослідження області розробки.
 - 1.2. Аналіз аналогічних розробок і формування базових особливостей, недоліків систем управління персоналом.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

- 1.3. Вибір та обґрунтування засобів та інструментів для створення продукту, формування вимог до апаратного забезпечення.
2. Проектування системи управління робочим часом.
 - 2.1. Побудова серверної архітектури вебзастосунку.
 - 2.2. Побудова клієнтської архітектури вебзастосунку.
 - 2.3. Побудова інтерфейсу та дизайну вебзастосунку.
 - 2.4. Розробка системи авторизації та адміністрування.
 - 2.5. Реалізація основного функціоналу системи управління робочим часом.
3. Тестування вебзастосунку.
 - 3.1. Тестування роботи основних функцій системи.
 - 3.2. Тестування графічного інтерфейсу.

Результатом виконання плану є створення вебзастосунку для управління робочим часом з елементами системи управління персоналом. Вебзастосунок надасть користувачам зручні та ефективні інструменти для відслідковування робочих годин, планування часу та виконання інших завдань, пов'язаних з управлінням персоналом.

1.2. Аналіз аналогів програмного продукту

В світі існує величезна кількість систем для управління персоналів різних масштабів, починаючи від орієнтованих на дуже вузький сегмент клієнтів із специфічними бізнес-потребами, і закінчуючи гігантськими платформами із величезною кількістю функціоналу, які можуть задовільнити більшість клієнтів.

Сучасні системи мають дуже широкий список функціоналу, який розроблюється великими командами протягом багатьох років. Ми зосередимо увагу на функціоналу в області управління персоналом, що є приступним на таких платформах.

Для аналізу аналогічних розробок було вибрано три сервіси HiBob, PeopleForce та Hurma.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

HiBob – це комплексна система управління персоналом (HRIS), яка включає різноманітні інструменти для ефективного управління персоналом, включаючи модулі для керування робочим часом. Це величезна платформа, в функціоналі якої можна легко загубитися. У 2023 році HiBob отримав статус лідера в сфері Cloud HR за версією Fosway 9-Grid. Було вибрано таку

Платформа надає можливості для відстеження робочого часу, управління відпустками та лікарняними, генерації звітів та їх аналітику, автоматичного відстеження, календар, автоматичні нагадування, а також інтеграції з іншими системами.

Переваги:

- зручність використання: інтерфейс HiBob інтуїтивно зрозумілий і зручний для користувачів, що спрощує процес навчання та використання системи;
- автоматизація: HiBob значно автоматизує процеси відстеження робочого часу та управління відпустками, що зменшує навантаження на HR-відділи;
- інтеграція: платформа легко інтегрується з іншими інструментами та системами, що забезпечує безперебійну роботу і обмін даними;
- аналітика та звітність: потужні інструменти аналітики дозволяють компаніям отримувати цінну інформацію про робочий час співробітників і приймати обґрунтовані рішення;

Недоліки:

- вартість: HiBob може бути досить дорогим рішенням для малих і середніх компаній, особливо з урахуванням широкого функціоналу і можливостей системи;
- складність налаштування: Хоча система є потужною, її налаштування може бути складним для невеликих компаній без відповідних технічних знань або підтримки;
- планування: HiBob має проблеми в області планування та оцінки продуктивності.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

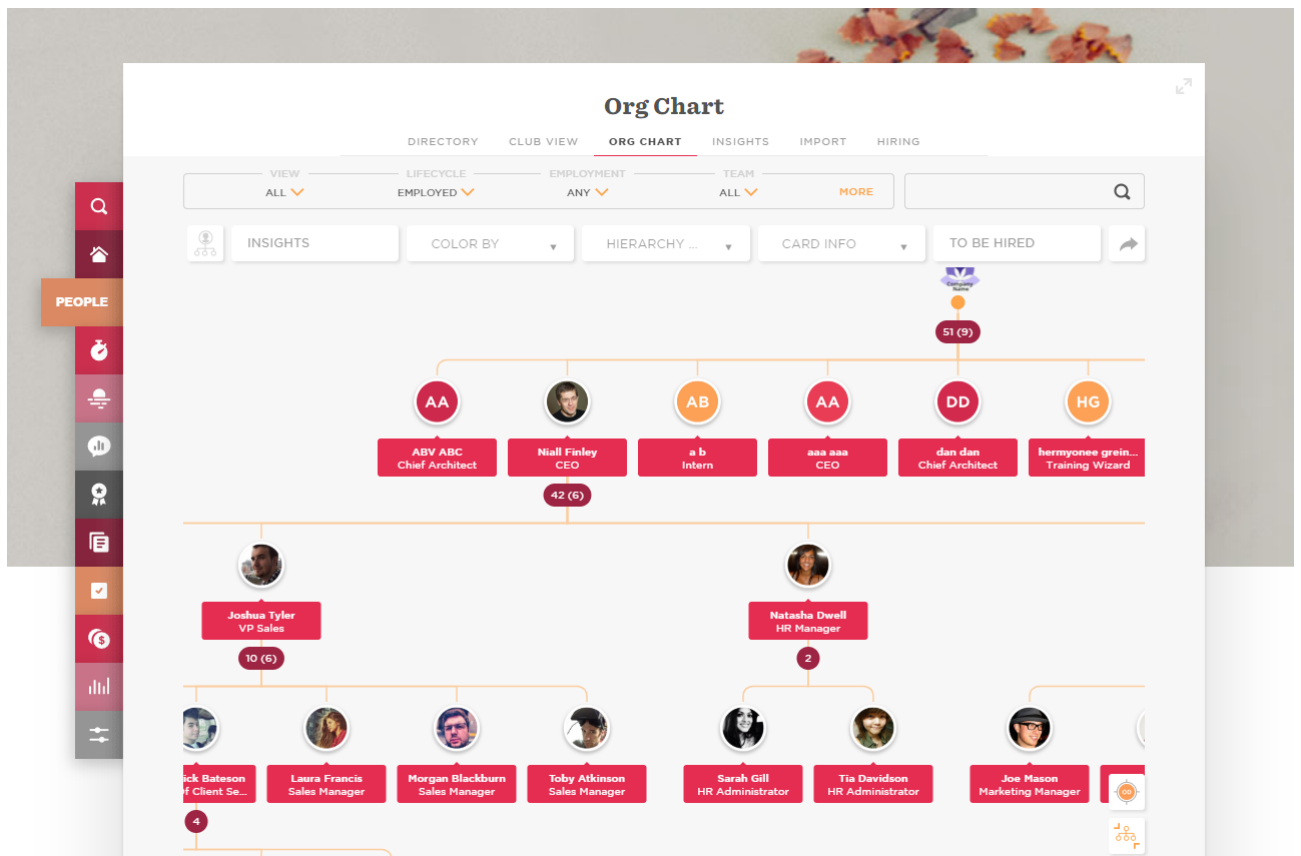


Рисунок 1.1. – Інтерфейс системи HiBob

HiBob є потужною HRIS-системою, яка надає широкий спектр функцій для керування, відслідковування та обліку робочого часу. Її зручний інтерфейс, потужні інструменти аналітики, автоматизація процесів і інтеграційні можливості роблять її привабливим вибором для багатьох компаній. Однак, висока вартість та складність налаштування можуть бути суттєвими перешкодами для малих і середніх підприємств. У порівнянні з HiBob, власна розробка може бути спрямована на створення більш доступного рішення з гнучкими налаштуваннями, орієнтованого на специфічні потреби клієнтів, зокрема в області керування робочим часом.

PeopleForce – це сучасний український стартап та міжнародний розробник all-in-one HR-платформи, яка пропонує широкий набір функцій для відслідковування та керування робочим часом. Компанія орієнтується на забезпечення зручних та ефективних інструментів для HR-фахівців та менеджерів, дозволяючи автоматизувати багато аспектів управління персоналом. Платформа включає модулі для відстеження робочого часу,

планування робочого графіку, керування відсутністю та створення аналітичних звітів.

PeopleForce надає API, що дозволяє підприємствам інтегрувати систему з різними платформами сторонніх розробників, такими як Telegram, Slack, LinkedIn, Jooble, Work.ua, G Suite та інші. Інші функції включають управління користувачами, експорт даних, аналітику, управління завданнями, відстеження відсутності, контроль доступу та автоматизовані нагадування.

Переваги:

- локація: платформа враховує специфіку місцевого ринку та вимоги українських підприємств;
- інтуїтивний інтерфейс: зручний і зрозумілий інтерфейс, який легко освоїти;
- широка функціональність: велика кількість інструментів для управління робочим часом;
- інтеграції: підтримка інтеграцій з іншими системами та сервісами.

Недоліки:

- вартість: відносно висока вартість, яка може бути значним фактором для малих підприємств;
- обмеження на налаштування: деякі налаштування можуть бути недостатньо гнучкими для специфічних потреб;
- адміністрація: процес створення рівнів доступу для робітників виглядає заплутаним;
- продуктивність: деякі окремі елементи мають невеликі проблеми із продуктивністю;

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

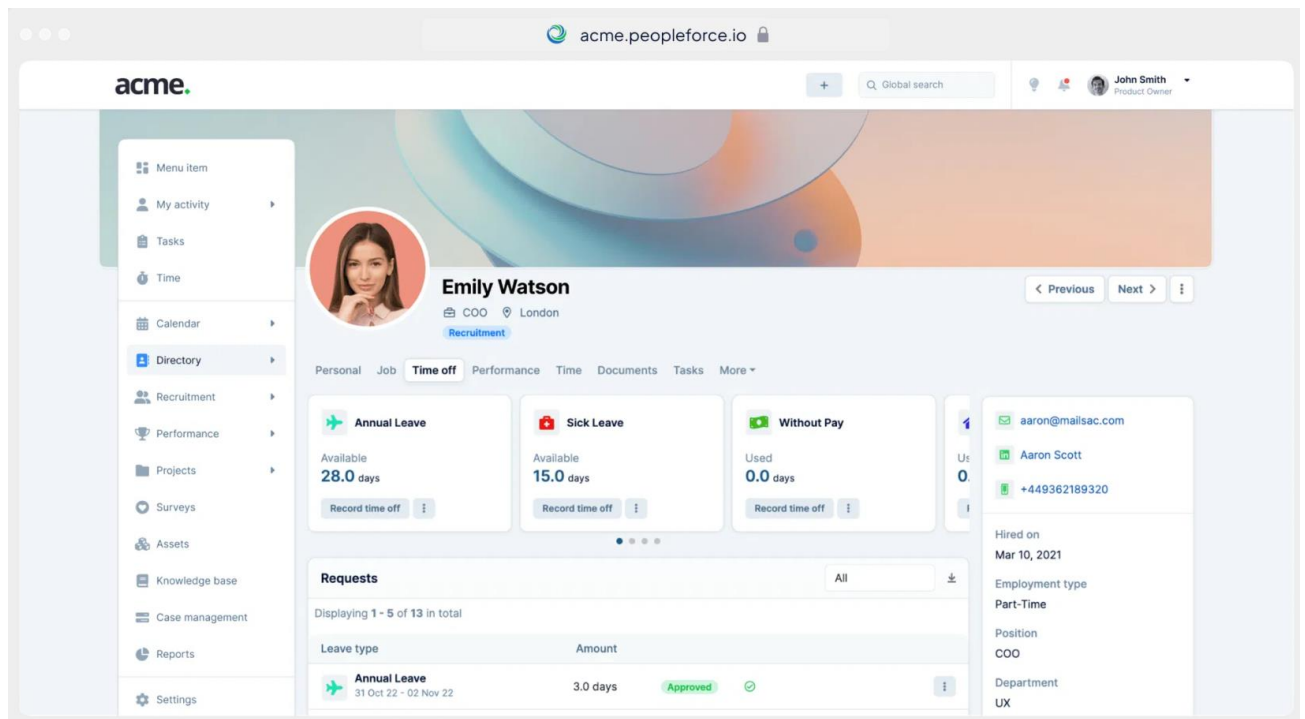


Рисунок 1.2. – Інтерфейс системи PeopleForce

Hurma – міжнародна компанія, яка спеціалізується на розробці системи для автоматизації процесів HR, рекрутингу та управління цілями компанії. Компанія представлена в країнах східної Європи. Офіс знаходиться в Україні, Харків.

Компанія орієнтована на надання інструментів для HR-фахівців та менеджерів, допомагаючи автоматизувати управління робочим часом, відпустками, лікарняними та іншими видами відсутності. Платформа пропонує зручний інтерфейс та широкий набір функцій для аналітики та звітності.

Переваги:

- відслідковування робочого часу: інструменти для точного обліку робочих годин співробітників;
- планування робочого часу: гнучкі інструменти для створення та керування графіками роботи;
- керування відсутністю: зручне керування відпустками, лікарняними та іншими видами відсутностей;
- робота зі статистикою та аналітикою: потужні інструменти для створення звітів та аналізу робочого часу.

Недоліки:

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

- проблеми із кількістю рекрутерів: стає вже працювати якщо в компанії багато рекрутерів;
- перерозподіл прав: незручний процес перерозподілу прав доступу;
- обмеження на налаштування: деякі налаштування можуть бути недостатньо гнучкими для специфічних потреб;
- продуктивність: деякі окремі елементи мають невеликі проблеми із продуктивністю;

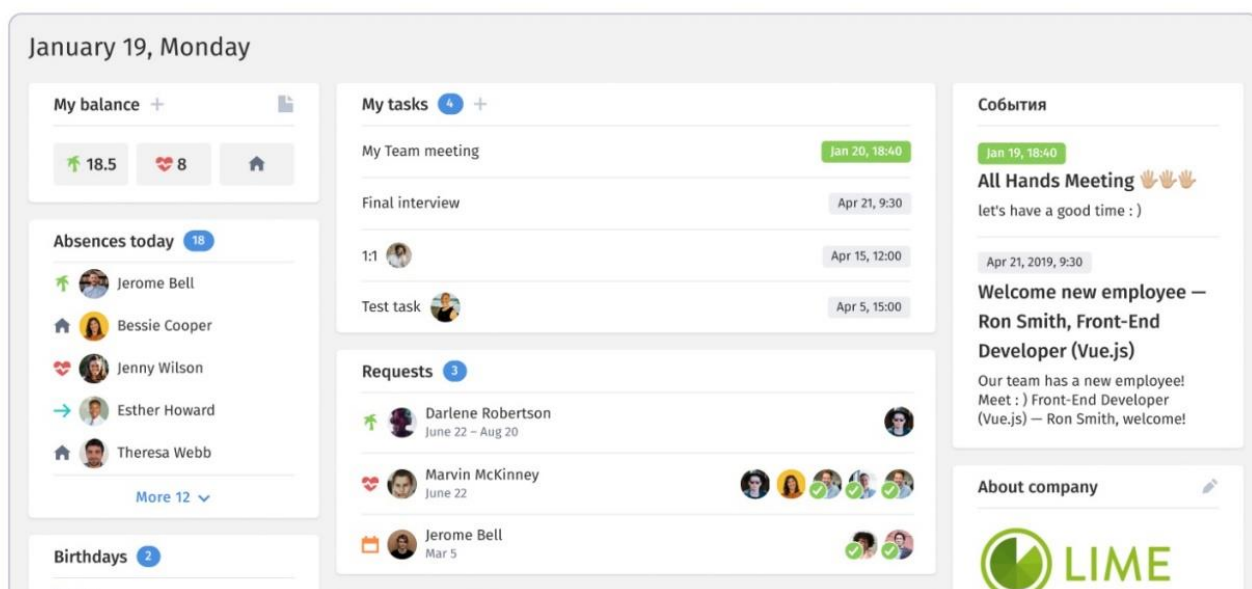


Рисунок 1.3. – Інтерфейс системи Hurma

Таблиця 1.1

Порівняння аналогічних розробок

Характеристика	HiBob	PeopleForce	Hurma	Власна розробка
Вартість	3	4	4	5
Сумісність із платформами	5	5	5	4
Загальна кількість функціоналу	5	4	4	3
Зручність інтерфейсу	4	4	4	3
Налаштування системи	5	3	3	2
Адміністрація системи	5	4	4	4
Відслідковування часу	5	4	4	4
Планування робочого часу	4	4	4	3
Керування відсутністю (відпустки, лікарняні)	5	4	4	4

Робота із статистикою та аналітикою	5	4	5	3
Управління відсутністю	5	4	5	4

Якщо провести висновок аналізу аналогів, то можна побачити що HiBob, як найбільша із наведених програм-аналогів, очікувано випереджає інші наведені аналоги в багатьох аспектах. Але врешті кожна система управління персоналом має свої плюси і мінуси. Тому можна із впевненістю сказати що вибір системи управління персоналом залежить від індивідуальних потреб організації-користувача.

У випадку власної розробки для дипломного проекту не вистачає достатньої кількості ресурсів для навіть приблизної реалізації тих функцій, які було виявлено у подібних розробках. Адже більшість таких систем, мають величзну кількість функціоналу, які можна виокремити як великі окремі програми.

Було прийнято рішення створити систему, яка наблизиться до проаналізованих розробок, та створить усі умови для подальшого розвитку вебзастосунку.

Власна розробка вебзастосунку для управління робочим часом спрямована на задоволення ключових потреб бізнесу з акцентом на простоту та масштабованість. Основні відмінності:

- розробка передбачає створення гнучкої архітектури, яка дозволяє легко додавати нові модулі та функції в майбутньому;
- система складається з окремих модулів, що дозволяє адаптувати функціонал під конкретні потреби компанії;
- основний акцент робиться на функціонал для відслідковування, планування та аналізу робочого часу, що може зробити систему спеціалізованою та ефективною в цій сфері.

1.3. Визначення архітектури вебзастосунку

При плануванні проекту було вирішено розробити систему управління персоналом за допомогою вебтехнологій. Вибір обумовлений доступністю оптимального інструментарію для розробки у вигляді вебзастосунку, що спрощує використання та підтримку системи. Одночасно потрібно реалізувати можливість одночасного доступу до ресурсу для багатьох користувачів через мережу, що і є основними перевагами вебсередовища, яке надає локальний мережевий доступ та можливість спільної роботи.

Для розробки застосунку було обрано клієнт-серверну архітектуру, що є стратегічним рішенням, орієнтованим на оптимальний розподіл функцій та завдань між клієнтською та серверною частинами системи. Цей вибір дозволяє ефективно управляти обробкою даних, взаємодією з користувачем та забезпечити гнучкість у розробці та масштабуванні системи. Клієнт-серверна архітектура дозволяє розділити відповідальності між фронтендом та бекендом, сприяючи оптимізації роботи застосунку та покращенню користувацького досвіду. Такий підхід також полегшує можливість розширення та підтримки системи в подальшому.

1.3.1. Визначення архітектури серверної частини вебзастосунку

Серверна частина є ядром системи і буде виконувати увесь основний функціонал системи. Сервер відповідальний за усі обрахунки, прийняття рішень, надання доступу до функцій, збирання та зберігання інформації. Тому критично важливим є вибір архітектури для забезпечення масштабованості, оптимізованості, простоти розробки, надійності, безпеки і швидкості роботи системи та ще й з урахуванням доступних ресурсів.

Монолітна архітектура

Даний вид архітектури є традиційним і стандартним для більшості програмних продуктів, тому більшість програмного коду, який можна знайти в інтернеті (в тому числі і навчальний матеріал), написаний саме у вигляді монолітної архітектури.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Цей тип архітектури передбачає написання програмного коду у вигляді об'єднаної автономної одиниці, яка є незалежною від інших програм. Тобто програмний продукт має представляти собою одну велику обчислювальну мережу з однією кодовою базою, яка об'єднує весь функціонал разом в одному місці.



Рисунок 1.4. – Схема монолітної архітектури [7]

Переваги:

- один виконуваний файл або каталог дозволяє значно спростити розгортання проекту;
- коли застосунок має одну кодову базу це полегшує розробку;
- у централізованій базі коду та сховищі, один API може виконувати ту саму функцію, яку виконують численні API в інших архітектурних шаблонах;
- end-to-end тестування виконувати значно легше ніж у розподіленій системі;
- централізована кодова база дозволяє простіше відслідковувати запити та знаходити помилки.

Недоліки:

- відсутність великої масштабованості проекту, складність розробки якого буде рости паралельно із розміром застосунку;
- якщо певна частина працює некоректно, це може вплинути на роботу усієї системи;
- будь-яка зміна в окремій частині застосунку тягне за собою зміни по всьому проекту, що є великою затратою часу;
- монолітна програма обмежена технологіями, які вже використовуються в ній;
- будь-яка зміна в проекті потребує повного повторного розгортання проекту.

Враховуючи вимоги до масштабованості у гнучкості власної розробки, було прийнято рішення відмовитися від неї через значні ускладнення при збільшенні розмірів проекту. Крім того розробка програмного продукту в даному випадку може зайняти значно більше часу і при цьому не дасть відчутних переваг.

Мікросервісна архітектура

Мікросервісна архітектура - це архітектурний шаблон, який ґрунтується на серії служб, які розгортаються незалежно один від одного. Кожна служба має свою власну бізнес-логіку та базу даних з конкретною метою.

Оновлення, тестування, розгортання та масштабування відбуваються всередині кожної служби. Мікросервіси роз'єднують значні бізнес-специфічні частини на окремі, незалежні кодові бази. Мікросервіси не зменшують складність, але вони роблять будь-яку складність видимою та керованою, розділяючи завдання на менші процеси, які працюють незалежно один від одного та сприяють загальному цілому.

Запровадження мікросервісів часто йде через залучення DevOps-спеціалістів, оскільки вони є основою для безперервної доставки, які дозволяють командам швидко адаптуватися до вимог користувачів.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

Microservices

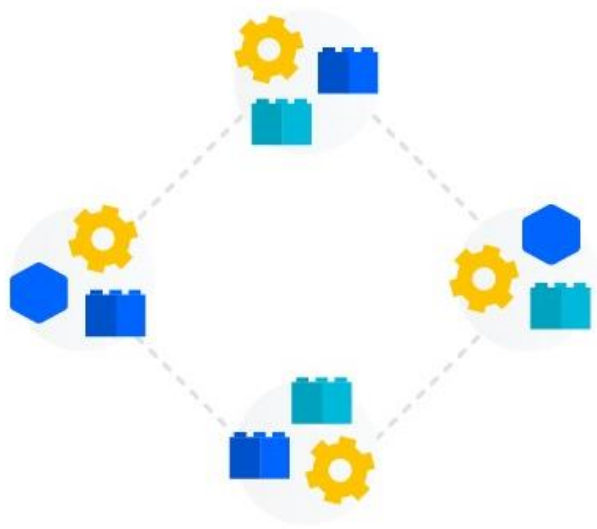


Рисунок 1.5. – Схема мікросервісної архітектури

Переваги:

- велика гнучкість при розробці із частим розгортанням проекту;
- легше налаштовувати масштабованість навантаження на окремі частини продукти;
- можливість робити оновлення частіше і швидше;
- систему набагато краще побудована для підтримки і усунення помилок;
- можна вибрати будь-які технології для реалізації окремих одиниць програми;
- зміни стосуються лише окремого сервісу системи.

Недоліки:

- система має вищу загальну складність і може мати набагато ширше коло використаних технологій;
- кожен окремий сервіс має власну інфраструктуру, а тому і розгортається окремо, через це також росте кількість ресурсів що потрібна на утримання такого проекту;
- якщо команда велика, то потрібний високий рівень комунікації між командою щоб координувати оновлення;

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

- якщо проект має помилку, то знайти її важче на фоні багатьох функціонуючих сервісів;
- через те що технології та кодова база сервісів можуть суттєво відрізнятися, то їх стає доволі важко стандартизувати;

Провівши аналіз особливостей мікросервісної архітектури, було прийнято рішення відмовитись від її застосування для проекту. Основна причина відмови це знадто велика кількість ресурсів, що потрібна для організації такої архітектури. Така архітектура буде доцільнішою для проектів що мають більшу кількість ресурсів.

N-шарова архітектура

По суті, N-рівнева програма – це розділення логіки програми на певні рівні. Архітектурний шаблон N-рівня є зрілою архітектурою та просто стосується додатків, які розділяють різні логічні рівні на окремі фізичні рівні.

Такий тип архітектури часто використовують як збалансований і перевірений архітектурний шаблон, який має багато переваг і націлений на програми що швидко розвиваються. Тому цей шаблон доволі часто використовуються через його адаптивність до різних вимог програмного застосунку.

Традиційна трирівнева програма має рівень презентації, середній рівень логіки і рівень бази даних. Середній ярус необов'язковий. Більш складні програми можуть мати більше трьох рівнів.

N-рівнева архітектура також дуже часто використовуються в багатьох проектах що реалізовані на принципах ООП, адже цей тип архітектури допомагає дотримуватися принципів SOLID, що позитивно сказується на якості та стабільності програми, а також її підтримкою.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

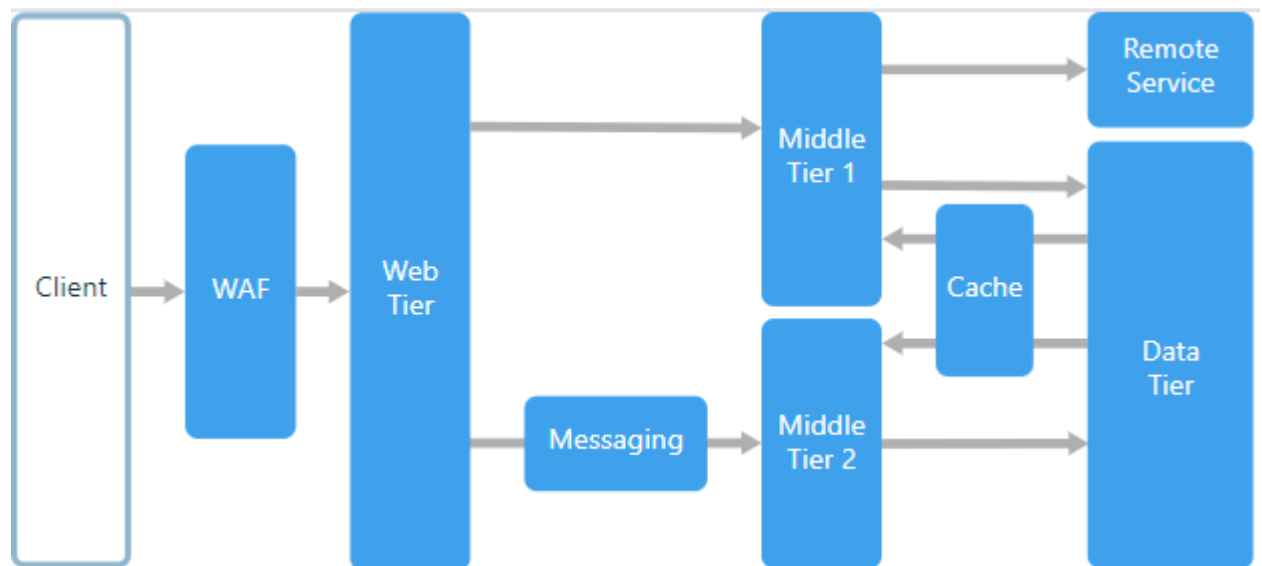


Рисунок 1.6. – Схема N-шарової архітектури [9]

Переваги:

- помилка або збій в одному з рівнів менш ймовірно вплине на функціональність інших рівнів;
- наявність кількох відокремлених компонентів в архітектурі дозволяє легко масштабувати, оновлюючи один або декілька з цих індивідуальних компонентів;
- зміни, внесені в одну секцію, не впливають на інші функції;
- кожен рівень працює на власній інфраструктурі або сервері, тому ресурси не діляться.

Недоліки:

- більша кількість рівнів може призвести до збільшення складності системи;
- розробка та підтримка багаторівневої архітектури можуть бути дорогими через її складність в порівнянні з монолітною архітектурою;
- чим більше рівнів, тим більше часу потрібно для обробки запитів, що може вплинути на продуктивність;
- управління багаторівневою архітектурою може бути складним, особливо при масштабуванні.

Через наявність одночасно масштабованості проекту, зручності розробки, якості, швидкості та надійності за збалансовану кількість ресурсів, було прийнято рішення застосувати саме цей архітектурний шаблон.

Для серверної частини застосунку була обрана трьохшарова архітектура, яка включає шар даних, шар логіки та шар API. Це рішення дозволяє чітко розділити функціональні компоненти та відповідальності між різними рівнями системи. У шарі даних відбувається управління базою даних та зберіганням інформації, шар логіки відповідає за обробку та виконання бізнес-логіки, тоді як шар API надає інтерфейс взаємодії для клієнтської частини. Ця архітектурна модель сприяє покращенню читабельності коду, підтримці та розширенню системи, а також спрощує тестування та вдосконалює загальну продуктивність.

1.3.2 Визначення структури клієнтської частини вебзастосунку

Для забезпечення масштабованості проекту критично важливим було впровадити зручну клієнтську архітектуру програми. Потрібно було обрати такий тип архітектури, який дозволив би зручно перевикористовувати код і його елементи, а також відокремити бізнес-логіку та графічну компоненти системи.

Реактивне програмування

Реактивне програмування (RP) – це парадигма програмування, у якій основна увага приділяється потокам даних та їхній автоматичній обробці в реальному часі. RP передбачає швидку реакцію програми на зміни, а також використання асинхронних операцій. [\[23\]](#)

Основні принципи RP – це створення реактивних ункцій, управління подіями та їхнє опрацювання в реальному часі, а також використання патернів проєктування, як-от Спостерігач (Observer), для зручного управління потоками даних.

Переваги реактивного програмування:

- масштабованість: легко управляти великою кількістю асинхронних подій;
- композиція: можливість комбінувати і трансформувати потоки даних у гнучкий і декларативний спосіб;

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

- керування побічними ефектами: полегшує роботу з побічними ефектами, такими як мережеві запити.

Недоліки реактивного програмування:

- складність: крута крива навчання для розуміння концепцій і операторів реактивного програмування;
- перебільшення: використання реактивного програмування може бути надмірним для простих задач.

Flux архітектура, методологія Re-ducks

Flux-архітектура дозволяє нам легко міркувати про нашу програму у спосіб, який нагадує функціональне реактивне програмування, або, точніше, програмування потоку даних або програмування на основі потоку, де дані проходять через програму в одному напрямку. Стан програми підтримується лише в сховищах, що дозволяє різним частинам програми залишатися сильно відокремленими. Там, де між магазинами виникають залежності, вони зберігаються в строгій ієрархії з синхронними оновленнями, керованими диспетчером.

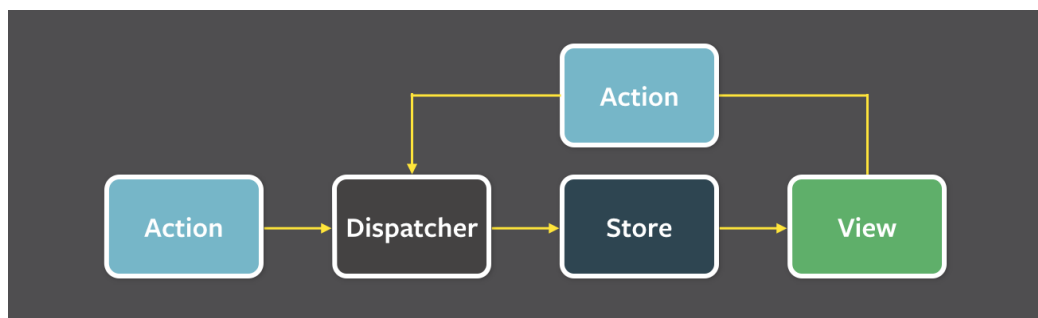


Рисунок 1.7. – Схема принципу роботи Flux архітектури [20]

Ідеї Flux архітектури також розвиває методологія Re-ducks, яка полягає у створенні файлової структури, яка є масштабованою та легкою для дотримання. Використання Redux для керування станом призводить до великої кількості дій, які використовуються в кількох компонентах, і структурі стану, яка має кілька рівнів. Як наслідок, ефективний спосіб імпортування функціональності в кілька файлів/компонентів є ключовим для уникнення помилок.

В нашому випадку було обрано взято за основу ці дві концепції і модифікувати під потреби проекту, що дає нам наступні переваги:

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

- інкапсуляція: усі пов'язані частини (типи, дії, ред'юсери) зібрані в одному місці, що полегшує навігацію та управління
- розділення відповідальності: дозволяє чітко розмежувати бізнес-логіку, управління станом і представлення даних;
- реактивність: зміни в стані створюють односторонній потік даних, що робить систему передбачуваною;
- масштабованість: збільшення масштабованості застосунку завдяки модульності;
- ефективність: забезпечує ефективне та передбачуване управління станом додатку.

Серед недоліків можна виділити:

- збільшення кількості коду: реалізація простих функцій може вимагати написання великої кількості додаткового коду;
- складність навчання: іноді може бути складно освоїти всі концепції та взаємодії між компонентами.

Component-based архітектура

Тепер, коли ми визначилися із підходом до організації бізнес логіки та потоку даних системи, потрібно вибрати принцип побудови графічних елементів системи. Було прийнято рішення використовувати компонентно-орієнтовану архітектуру, яка буде відокремленою від бізнес логіки.

Компонентно-орієнтована архітектура ставить у центр уваги інтерфейс користувача, розглядаючи його як композицію незалежних компонентів. Кожен компонент відповідає за власну функціональність, візуальний вигляд та логіку поведінки. Це відкриває для нас такі переваги:

- повторне використання: полегшує розробку та зменшує кількість дублювання коду;
- модульність: кожен компонент є самодостатньою одиницею, що дозволяє зосередитися на його функціональності та легко змінювати або замінювати його без впливу на інші частини додатку;

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

- чистота коду: компоненти дозволяють створювати чистий та організований код, оскільки кожен компонент відповідає за обмежений набір функцій та є відокремленими від бізнес-логіки;
- простота тестування: компоненти можуть бути легко тестовані окремо, що сприяє створенню ефективних та надійних тестів;
- швидкість розробки: інтерфейс користувача може бути швидко створений шляхом комбінування готових компонентів, замість написання всього з нуля.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для створення архітектури застосунку, що створюється для такої критично важливої сфери як управління персоналом, потрібно обрати технології що зможуть забезпечити ефективність, стабільність та зручність впровадження нового функціоналу.

Вибір бази даних є критично важливим під час планування розробки. Від цього залежить ефективність та швидкодія проекту, його масштабованість та можливість оптимізації в подальшому. Важливо обрати базу даних, яка відповідає конкретним вимогам проекту, забезпечить надійність зберігання та швидкість доступу до інформації. Окрім того, важливо враховувати масштаби та обсяги даних, які будуть оброблятися, а також можливості інтеграції з іншими системами. Всі ці фактори сприятимуть успішній реалізації та подальшому успіху системи управління людськими ресурсами.

Було прийнято рішення використовувати реляційну базу даних SQL і СУБД SQL Server для досягнення оптимальних рішень. Реляційна база даних зможе виокремити чітку структуру і таблиці, які можна зручно використовувати та розширювати. Крім того важливу роль у виборі зіграла високий рівень сумісності із технологіями що будуть використані при розробці серверної частини вебсервісу.

Для реалізації серверу вебзастосунку було обрано використання мови програмування C# та фреймворку ASP.NET. Це стратегічне рішення дозволило

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

забезпечити ефективність, надійність та швидкість в розробці серверної частини системи. Мова С# володіє великою популярністю в розробці вебзастосунків, а фреймворк ASP.NET забезпечує високий рівень абстракції та простоту в роботі з вебтехнологіями.

В основі шару даних серверної трьох-шарової архітектури для взаємодії з базою даних використовується Dapper - простий та швидкий ORM (Object-Relational Mapping) для мови програмування С#. Dapper надає простий і дуже ефективний інтерфейс для виконання запитів до бази даних, що сприяє оптимальній роботі з даними та підвищує продуктивність вебзастосунку. Це рішення допомагає забезпечити ефективну та швидку взаємодію з базою даних, роблячи систему більш масштабованою та продуктивною.

Для того щоб задовільнити додаткові вимоги проекту були використані інші додаткові пакети. Наприклад був застосований MailKit, для створення і надсилання електронної пошти. Для вирішення проблеми розвитку схеми бази даних для кількох баз даних були задіяні міграції за допомогою FluentMigrator. Quartz.Net було використано для виконання запланованих завдань. А пакет EPPlus надає зручний інструментарій для створення XML (.xml) та EXCEL (.xlsx) файлів.

У процесі розробки серверу веб-сервісу було вирішено використовувати GraphQL як мову запитів до API для забезпечення гнучкості та ефективності. Це технологія спрощує взаємодію і зменшуючи мережевий трафік завдяки можливості отримувати лише необхідні дані. Для продуктів, що націлені на велику масштабованість, GraphQL забезпечує гнучкість відносно структури даних, підтримку кешування для оптимізації швидкості та масштабованості системи, а також гнучкий механізм керування доступом до даних, забезпечуючи безпеку та конфіденційність.

Реалізація GraphQL для серверу веб-сервісу була здійснена за допомогою бібліотеки GraphQL.Net. Цей інструментарій надає потужний фреймворк для впровадження GraphQL в .NET-середовищі, який лежить в основі шару API трьох-шарової архітектури серверу. Використання GraphQL.Net дозволяє

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

ефективно вирішувати завдання з обробки та відповіді на GraphQL-запити, забезпечуючи зручний та ефективний механізм обміну даними між клієнтом та сервером.

Для реалізації клієнтської частини технологій було обрано мову TypeScript як розширену типізовану версію JavaScript та бібліотеку React для впровадження інтерфейсів на основі технології SPA (Single-Page Application). React є гарним прикладом бібліотеки що має в основі концепції компонентно-орієнтованої архітектури відображення графічних елементів.

Для керування внутрішнім станом клієнтського застосунку і реалізацією Flux архітектури використовувалися такі технології як Redux та Redux Toolkit для керування станом програми. Вони запроваджують єдине та централізоване для усього застосунку джерело стану в передбачуваному стані та методи його зміни.

Для реалізації концепції реактивного програмування також використовувалися бібліотеки RxJS та Redux-Observable. Використовування реактивного програмування дозволяє нам створити неперервний, контрольований та асинхронний потік даних в застосунку. Також бібліотеки надають великий функціонал для створення сторонніх ефектів як асинхронні запити на API.

Вимоги до середовища виконання для вебсистеми управління персоналом:

1. Серверна частина:

- Операційна система: Windows Server або Linux.
- Процесор: AMD Ерус 7313 або аналогічний.
- Пам'ять: Мінімум 128 ГБ RAM.
- Дисковий простір: Мінімум 500 ГБ вільного простору на жорсткому диску.

2. База даних:

- Система управління базами даних (СУБД) реляційного доступу.

3. Клієнтська частина:

- Операційна система: Будь-яка, яка підтримує сучасні браузер.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

- Браузер: Chrome 50, Firefox 47, Edge 18, Opera 38 або їх наступні версії.
- Оперативна пам'ять: Мінімум 512 МБ RAM.

Висновки до розділу 1

Під час дослідження напрямків використання інформаційних технологій для реалізації вебсистеми управління персоналом, було досліджено усі важливі аспекти початкового планування вебсистеми та участі інформаційних технологій в них.

Було проведено детальну роботу для постановки завдання розробки. Під час виконання цієї частини роботи було визначено тип вебсистеми, її мету, її масштаби і детальні кроки для досягнення бажаного результату.

Також було проведено аналіз аналогічних розробок на прикладі таких систем управління персоналом як HiBob, PeopleForce та Nurta. Було визначено їхні масштаби, кількість функціоналу та особливості. Проведено детальний аналіз сильних та слабких сторін кожної аналогічної розробки. Було здійснено пошук додаткових ідей та базових потреб при створенні системи управління персоналом. Було обрано напрямок розвитку власної розробки із урахуванням проведеного аналізу.

Проведено критично важливу роботу із визначенням архітектури вебсистеми. Прийнято рішення зупинитися на використанні вебтехнологій для досягнення найкращих результатів. Було обрано використовувати клієнт-серверну архітектуру для розподілення функцій системи.

Проведено аналіз монолітної, N-шарової та мікросервісної архітектури для серверної частини вебсервісу. Прийнято рішення використовувати N-шарову архітектуру як рішення збалансоване між технічними перевагами та економією ресурсів.

Було проведено детальний аналіз можливих варіантів клієнтської архітектури. Було прийнято рішення використовувати концепцію реактивного програмування, архітектури Flux та методологію Re-ducks для клієнтської

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

бізнес-логіки та компонентно-орієнтовану архітектуру створення графічних елементів інтерфейсу.

Було обрано та обґрунтовано цей вибір інструментальних засобів для реалізації вебсистеми із урахуванням усіх бізнес-вимог проекту та обраних архітектурних рішень. Провівши аналіз вибраних технологій, було сформульовано вимоги до апаратного забезпечення.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ КЕРУВАННЯ РОБОЧИМ ЧАСОМ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Вебзастосунок надає доступ до своїх функцій через графічний вебінтерфейс. Тому потрібно чітко визначити, як різні користувачі будуть взаємодіяти із системою для виконання власних завдань.

Враховуючи специфіку проекту, ми використовуємо модель керування доступом на основі дозволів (permission-based access control, RBAC), що забезпечує більш гнучке та детальне налаштування прав доступу для кожного користувача.

Модель RBAC дозволяє визначати та надавати специфічні дозволи окремим користувачам або групам користувачів залежно від їхніх функціональних обов'язків та потреб. Це забезпечує підвищену безпеку та ефективність управління доступом, дозволяючи користувачам виконувати лише ті операції, на які вони мають відповідні дозволи.

Для відтворення усіх можливих варіантів використання системи було створено діаграму варіантів використання, яку наведено в додатках (див. рис. А.1).

Опис прецедентів системи наведено нижче (табл. 2.1 – 2.12).

Таблиця 2.1

Опис прецеденту «Перевірка дозволу»

Назва варіанту використання	Перевірка дозволу
Основні актори	Будь-який користувач
Короткий опис	Більшість прецедентів включають в себе попередню перевірку дозволу, що завершується або в виконанням бажаної операції, або у відображенні помилки

Таблиця 2.2

Опис прецеденту «Авторизація»

Назва варіанту використання	Авторизація
Основні актори	Будь-який користувач
Короткий опис	Користувач вводить email-адресу та пароль для того щоб увійти в систему

Таблиця 2.3

Опис прецеденту «Відновлення паролю»

Назва варіанту використання	Відновлення паролю
Основні актори	Будь-який користувач
Короткий опис	Користувач вводить email-адресу для того щоб отримати листа із посиланням для відновлення паролю. За посиланням користувач може повторно ввести пароль

Таблиця 2.4

Опис прецеденту «Онбординг»

Назва варіанту використання	Онбординг
Основні актори	Будь-який користувач
Короткий опис	Користувач, якого зареєстрували в системі, може ввести початкові дані профілю та встановити собі пароль для отримання доступу до системи.

Таблиця 2.5

Опис прецеденту «Первинне налаштування»

Назва варіанту використання	Первинне налаштування
Основні актори	Неавторизований користувач
Короткий опис	Цей прецедент можливий у випадку встановлення чистої системи, де ще не було зареєстровано жодного користувача

Таблиця 2.6

Опис прецеденту «Робочий час»

Назва варіанту використання	Авторизація
Основні актори	Авторизований користувач
Короткий опис	Користувачі отримують можливість відслідковувати власний робочий чи запланований час, керувати власними або, за наявності дозволу, чужими робочими сесіями.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Продовження таблиці 2.6

	Користувачі можуть відкрити планувальник і звірити власні робочі години, переглянути і редагувати робочий час інших.
--	--

Таблиця 2.7

Опис прецеденту «Відпустки»

Назва варіанту використання	Авторизація
Основні актори	Авторизований користувач
Короткий опис	Користувач має можливість створити запит на відпустку, відмінити запит, бачити історію власних відпусток. За наявності дозволу користувач може затверджувати або відмінити запити інших на відпустку. Якщо той хто просить відпустку має менеджера, то тільки менеджер може затвердити йому відпустку.

Таблиця 2.8

Опис прецеденту «Лікарняні»

Назва варіанту використання	Лікарняний
Основні актори	Авторизований користувач
Короткий опис	Користувач бачить історію своїх лікарняних. За наявності відповідного дозволу він може керувати цими записами.

Таблиця 2.9

Опис прецеденту «Вихідні»

Назва варіанту використання	Вихідні
Основні актори	Авторизований користувач
Короткий опис	Користувач має можливість бачити інформацію про вихідні на сторінці планувальника. За наявності дозволу можна керувати вихідними. При створенні вихідного можна надіслати листа-оповіщення для кожного користувача.

Таблиця 2.10

Опис прецеденту «Управління користувачами»

Назва варіанту використання	Управління користувачами
Основні актори	Авторизований користувач

Продовження таблиці 2.11

Короткий опис	Користувач може деактивувати чи повторно активувати акаунт іншого користувача. Редагувати профіль користувача, а користувач з усіма дозволами може редагувати дозволи інших. Користувачі із без дозволу можуть лише переглядати дані профілю.
---------------	---

Таблиця 2.11

Опис прецеденту «Перегляд статистики»

Назва варіанту використання	Перегляд статистики
Основні актори	Авторизований користувач
Короткий опис	Користувач має можливість подивитися інформацію про відпрацьовані години за місяць, чи інші дані на сторінці статистики. За наявності дозволу доступна функція завантаження звіту у вигляді Excel-файлу.

Таблиця 2.12

Опис прецеденту «Налаштування системи»

Назва варіанту використання	Налаштування системи
Основні актори	Авторизований користувач
Короткий опис	За умови наявності дозволу користувач може змінювати декі параметри системи, наприклад текст, який бачили щойно зареєстрований користувач чи кількість днів відпустки, яку компанія надає на рік.

Вебзастосунок орієнтований на масштабованість і гнучку архітектуру. Тому важливо впровадити більшість інструментів, які дозволять згодом реалізувати всі необхідні функції для такої великої і складної програми, як система управління персоналом. Наявність цих інструментів у системі є критично важливою, оскільки вони зможуть вирішувати більшість проблем, навіть якщо наразі функціонал, реалізований за їх допомогою, ще не є значним.

Сформулюємо вимоги до системи, беручи до уваги принцип контролю доступу на основі дозволів. Відповідно до потреб компанії користувач може мати найрізноманітніший набір дозволів, тож наведені нижче вимоги користувачів це лише декілька із прикладів потреб компанії.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				36
Змн.	Арк.	№ докум.	Підпис	Дата		

Вимоги користувачів (залежить від індивідуальної кількості дозволів):

1. Звичайний робітник (без особливих дозволів):
 - 1.1. авторизація і відновлення паролю, онбориднг;
 - 1.2. відстеження робочого часу;
 - 1.3. керування власним робочим часом та перегляд планувальника інших;
 - 1.4. перегляд записів про старі відпустки, наявність інформації про наявну кількість балансу відпустки, створення нового запиту на відпустку;
 - 1.5. перегляд записів про лікарняні;
 - 1.6. перегляд детальної статистики про відпрацьовані години місяця;
2. Бухгалтер (додаткові дозволи, наприклад дозвіл на завантаження Excel-файлу із статистикою):
 - 2.1. завантаження файлу із статистикою за місяць;
 - 2.2. чітко бачити відпрацьовані години користувача протягом певного періоду часу;
 - 2.3. мати інформацію про відпустки, лікарняні інших робітників і вихідні дні.
3. HR-менеджер (додаткові дозволи, наприклад дозвіл на керування вихідними днями):
 - 3.1. керування вихідними даними;
 - 3.2. редагування даних профілю;
 - 3.3. активація та деактивація акаунту;
 - 3.4. реєстрація нових користувачів;
 - 3.5. керування відпустками інших користувачів.

Функціональні вимоги:

1. відсутність традиційної реєстрації, користувача в системі має реєструвати людина із відповідним дозволом. Після отримання листа користувач завершує реєстрацію самостійно встановивши собі пароль;
2. можливість зручно і швидко знайти потрібного робітника;
3. потрібно надати можливість зручно відслідковувати час користувачам. Для тих, хто працює повний робочий день, має бути присутнє автоматичне нарахування 8 годин на день;

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

4. візуалізація статистичної інформації для кращого сприйняття;
5. зручний перегляд інформації про години за допомогою планувальника;
6. створення сторінки налаштувань для додавання гнучкості елементам системи;
7. отримувати сповіщення у вигляді email-листів коли це необхідно.

Не функціональні вимоги:

1. Сприйняття:

- 1.1. час, потрібний для освоєння вебсайту – від 3-х до 5-ти хвилин, для
- 1.2. досвідчених менше 2-х хвилини;
- 1.3. час відповіді вебсайту на дії користувача не повинні перевищувати 5 секунд, а для складних виконання складних операцій обчислення – 15-ти секунд;
- 1.4. інтерфейс вебсайту повинен бути інтуїтивно зрозумілим та зручним для користувача, та не вимагати від нього додаткової підготовки.

2. Надійність:

- 2.1. вебсайт має бути доступний на протязі усієї доби;
- 2.2. мати можливість працювати на локальній мережі без доступу до інтернету;

3. Продуктивність:

- 3.1. Повинен надати достатній рівень продуктивності для роботи в системі від 10 користувачів одночасно.

Системні вимоги:

1. Вимоги до середовища виконання: платформа повинна задовільнять вимогам на пристроях, що знаходиться в наступній мінімальній комплектації:
 - 1.1. 4 Гб оперативної пам'яті.
 - 1.2. Процесор з тактовою частотою не нижче 1.4 GHz.
2. Вимоги до сервера інформаційної системи:
 - 2.1. Мати операційну систему Windows чи Linux.

Аналіз вимог до вебзастосунку дозволив краще сформулювати об'єктно орієнтовану структуру системи і реалізувати потрібні елементи системи. Для прикладу розглянемо діаграму класів модулю користувачів (рис. 2.1).

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

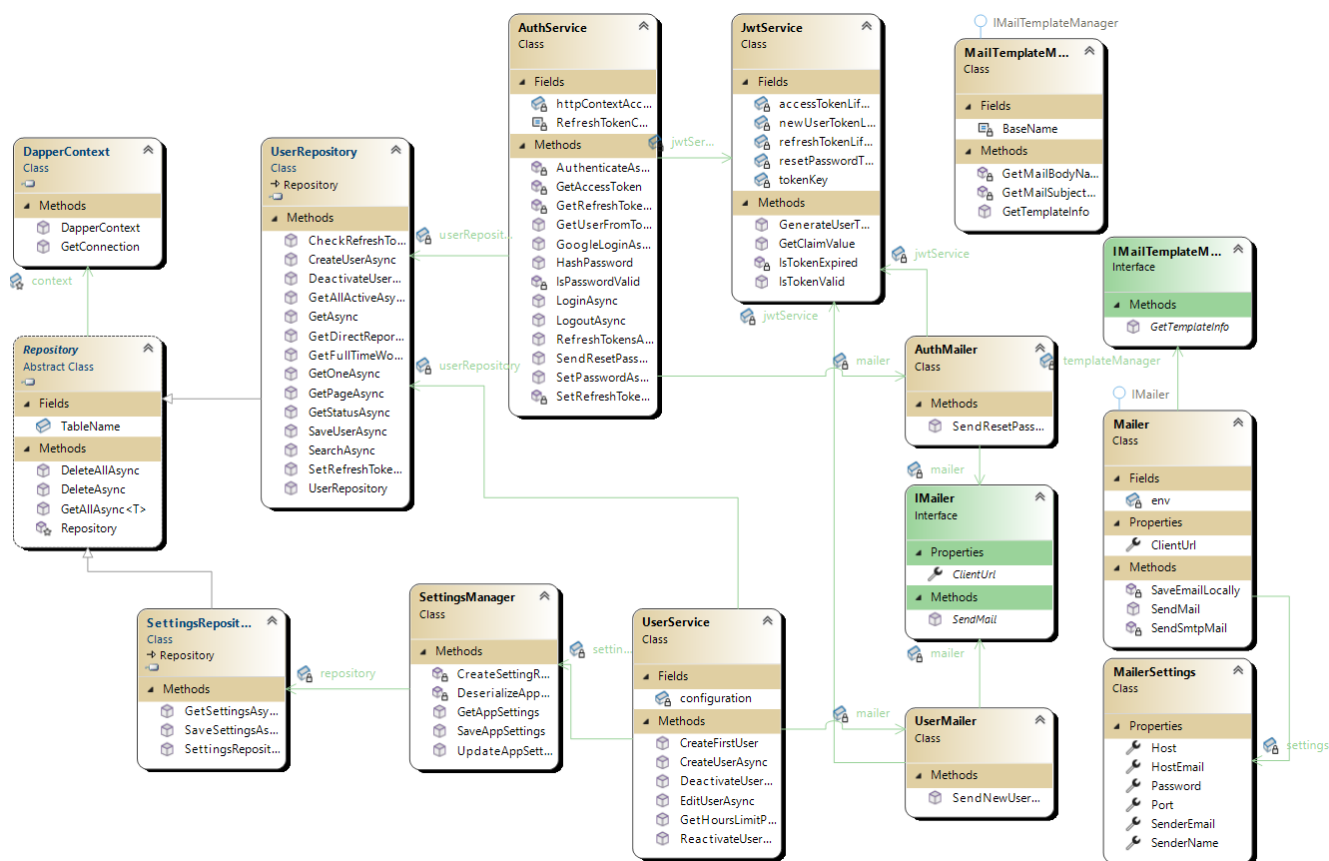


Рисунок 2.1 – Діаграма класів для модулю користувача

На діаграмі класів добре видно взаємодії елементів системи, розглянемо їх трохи детальніше.

DapperContext: Створює і керує підключенням до бази даних за допомогою Dapper ORM, яка лежить в основі взаємодії із базою даних. Абстрактний клас **Repository**, який є батьківським для усіх репозиторіїв, забезпечує базові спільні операції CRUD для сутностей, а також використовує DapperContext для взаємодії із базою даних.

Кожен клас репозиторію прив'язаний до своєї таблиці в базі даних, наприклад **UserRepository** (таблиця Users) чи **SettingsRepository** (таблиця Settings). Іноді потрібна більш комплексна логіка для взаємодії із об'єктами бази даних, наприклад клас **SettingsManager** керує налаштуваннями застосунку, включаючи збереження та оновлення конфігурацій, які зберігаються в серіалізованому форматі. Таким чином використовуючи таку обгортку над репозиторієм, ми можемо взаємодіяти із налаштуваннями так, ніби виконуємо стандартні CRUD-операції.

Інтерфейс **IMailTemplateManager**, який реалізується класом **MailTemplateManager**, створений для того щоб читати інформацію із ресурсних файлів (.resx). В нашому випадку ресурсний файл зберігає шаблони листів для відправки.

Для відправки email-листів створено інтерфейс **IMailer**, реалізований через відповідний клас **Mailer**. Mailer використовує раніше описаний **IMailTemplateManager** для читання шаблону листу, підстановки в нього потрібних значень та його відправки. Для зберігання таких налаштувань пошти як хост, порт, пароль та інших, використовується клас **MailSettings**, який читає ці дані із конфігурації.

Хоч клас **Mailer** із відповідним інтерфейсом вже можна використовувати для відправки листа, набагато зручніше використовувати класи-обгортки, які будуть робити це за нас. Наприклад клас **AuthMailer** реалізовує логіку надсилення листів пов'язаних із автентифікацією (встановлення паролю), для чого необхідно просто викликати метод.

Такі класи як **UserServive** виконують основні функції системи, до них звертаються класи що реалізують API. **UserService** це клас для управління користувачами, що виконує логіку управління цими користувачами. Він використовує інші класи для реалізації алгоритмів, такі як **UserRepository**, **SettingsManager** чи **UserMailer**. Алгоритми класу визначають коли потрібно звернутися до бази даних, надіслати лист чи просто провести обрахунки.

JwtService – це клас, що лежить в основі автентифікації користувачів і містить методи для генерації, валідації та читання JWT токенів для користувачів. **AuthService** покладається на цей клас для автентифікації користувачів, включаючи управління токенами доступу, хешування паролів, встановлення cookie, встановлення токенів і паролів користувачів в базу даних за допомогою класу **UserRepository**, відправки листів за допомогою класу **AuthMailer**.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

2.2. Розробка бази даних системи.

Однією з ключових переваг SQL Server для проектів є його здатність масштабуватися — від невеликих додатків до великих корпоративних рішень, що дозволяє легко адаптуватися до зростаючих потреб бізнесу. Завдяки інструментам, таким як індексація, розподілена обробка запитів і підтримка паралельного виконання операцій, SQL Server забезпечує високу швидкість обробки даних.

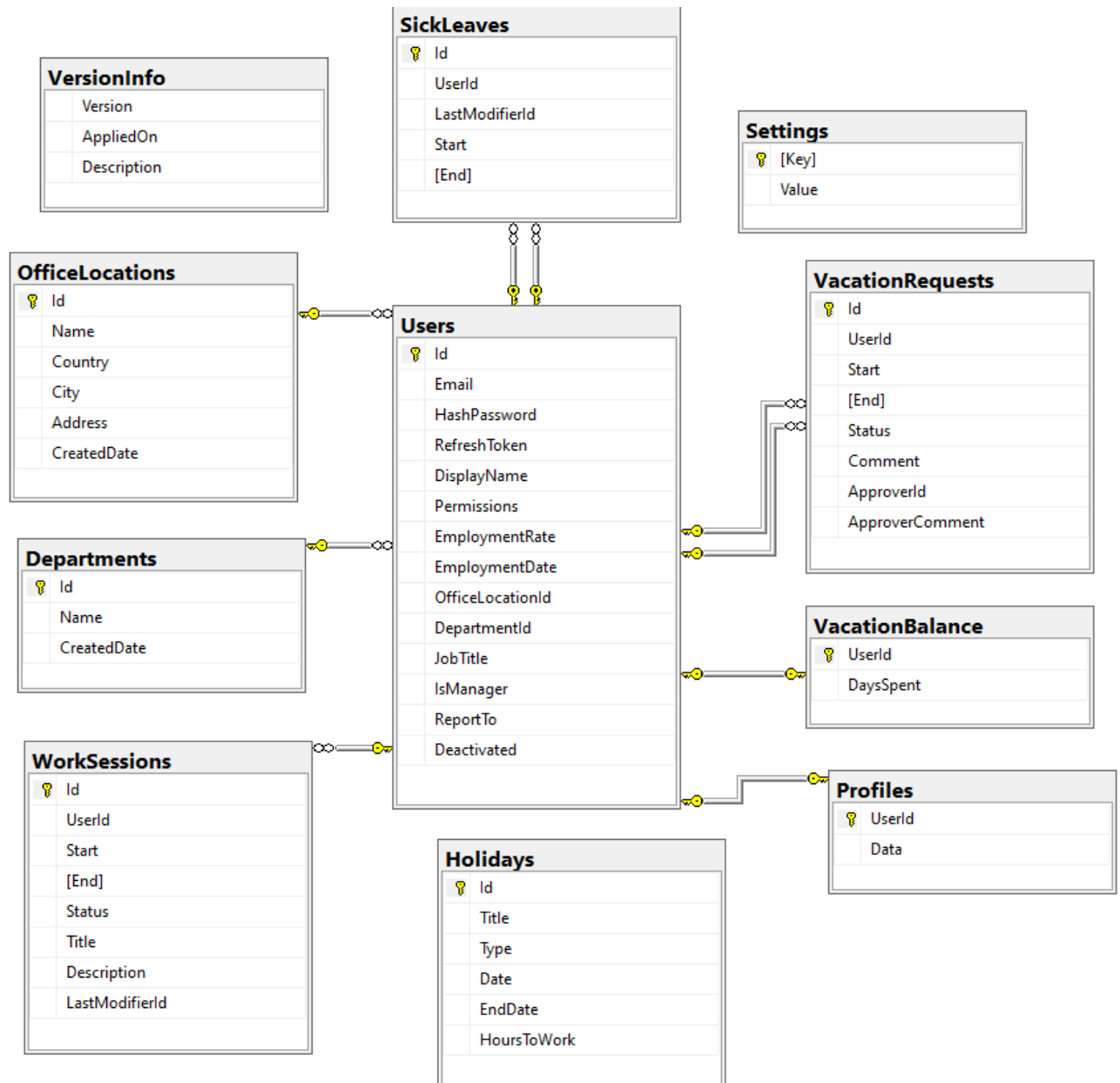


Рисунок 2.2 – Схема бази даних

У проекті активно застосовуються індекси для пришвидшення пошуку серед часто використовуваних запитів колонок.

У проєкті, де важливий швидкий розвиток і ефективне управління змінами структури бази даних, використовується FluentMigrator. Це інструмент для управління міграціями бази даних, який дозволяє автоматизувати процеси створення, оновлення та модифікації схеми бази даних, що значно спрощує та прискорює процес розробки і розгортання нових функцій, особливо коли над проєктом працює команда, що паралельно працює на д продуктом.

Список таблиць із їх призначенням:

1. **VersionInfo**: Зберігає інформацію про версії схеми бази даних для FluentMigrator.
2. **Departments**: Зберігає інформацію про відділи компанії.
3. **OfficeLocations**: Зберігає інформацію про місцезнаходження офісів компанії.
4. **Holidays**: Зберігає інформацію про святкові дні та інші події.
5. **Settings**: Зберігає налаштування системи.
6. **Users**: Зберігає інформацію про користувачів системи.
7. **Profiles**: Зберігає профілі користувачів з додатковими даними.
8. **WorkSessions**: Зберігає інформацію про робочі сесії користувачів.
9. **SickLeaves**: Зберігає інформацію про лікарняні листи користувачів.
10. **VacationBalance**: Зберігає баланс відпусток користувачів.
11. **VacationRequests**: Зберігає запити на відпустки від користувачів.

Таблиця 2.13

Опис таблиці «VersionInfo»

Полу	Тип	Опис
Version	bigint	Версія схеми бази даних
AppliedOn	datetime	Дата застосування версії
Description	nvarchar(1024)	Опис змін

Таблиця 2.14

Опис таблиці «Departments»

Полу	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор відділу
Name	nvarchar(100)	Назва відділу
CreatedDate	datetime	Дата створення відділу

Таблиця 2.15

Опис таблиці «Office location»

Полю	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор офісу
Name	nvarchar(100)	Назва офісу
Country	nvarchar(100)	Країна розташування (опціонально)
City	nvarchar(100)	Місто розташування (опціонально)
Address	nvarchar(250)	Адреса (опціонально)
CreatedDate	datetime	Дата створення відділу

Таблиця 2.16

Опис таблиці «Holidays»

Полю	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор свята
Title	nvarchar(255)	Назва свята
Type	smallint	Тип свята
Date	date	Дата початку свята
EndDate	date	Дата завершення свята (опціонально)
HoursToWork	smallint	Кількість годин роботи (опціонально)

Таблиця 2.17

Опис таблиці «Settings»

Полю	Тип	Опис
Key	nvarchar(100)	Унікальний ключ налаштування
Value	nvarchar(max)	Значення налаштування

Таблиця 2.18

Опис таблиці «Users»

Полю	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор користувача
Email	nvarchar(255)	Електронна адреса користувача
HashPassword	nvarchar(max)	Захешований пароль (опціонально)
RefreshToken	nvarchar(max)	Токен оновлення (опціонально)
DisplayName	nvarchar(255)	Відображуване ім'я користувача
Permissions	nvarchar(max)	Права доступу користувача (опціонально)
EmploymentRate	smallint	Відсоток зайнятості
EmploymentDate	datetime	Дата початку роботи
OfficeLocationId	uniqueidentifier	Ідентифікатор офісного місця (опціонально)
DepartmentId	uniqueidentifier	Ідентифікатор відділу (опціонально)
JobTitle	nvarchar(200)	Посада користувача
IsManager	bit	Вказує, чи є користувач менеджером
ReportTo	uniqueidentifier	Ідентифікатор керівника (опціонально)
Deactivated	bit	Вказує, чи деактивований користувач

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Таблиця 2.19

Опис таблиці «Profiles»

Полю	Тип	Опис
UserId	uniqueidentifier	Унікальний ідентифікатор користувача
Data	nvarchar(max)	Дані профілю користувача

Таблиця 2.20

Опис таблиці «WorkSessions»

Полю	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор робочої сесії
UserId	uniqueidentifier	Ідентифікатор користувача
Start	datetime	Дата та час початку сесії
End	datetime	Дата та час завершення сесії (опціонально)
Status	smallint	Статус сесії
Title	nvarchar(100)	Назва сесії (опціонально)
Description	nvarchar(200)	Опис сесії (опціонально)
LastModifierId	uniqueidentifier	Ідентифікатор останнього модифікатора запису

Таблиця 2.21

Опис таблиці «SickLeaves»

Полю	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор лікарняного
UserId	uniqueidentifier	Ідентифікатор користувача
Start	datetime	Дата початку лікарняного
End	datetime	Дата завершення лікарняного
LastModifierId	uniqueidentifier	Ідентифікатор останнього модифікатора запису

Таблиця 2.22

Опис таблиці «SickLeaves»

Полю	Тип	Опис
UserId	uniqueidentifier	Унікальний ідентифікатор користувача
DaysSpent	int	Кількість витрачених днів відпустки

Таблиця 2.23

Опис таблиці «VacationRequests»

Полю	Тип	Опис
Id	uniqueidentifier	Унікальний ідентифікатор запиту на відпустку
UserId	uniqueidentifier	Ідентифікатор користувача
Start	date	Дата початку відпустки
End	date	Дата завершення відпустки
Status	smallint	Статус запиту

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

Comment	nvarchar(255)	Коментар користувача (опціонально)
ApproverId	uniqueidentifier	Ідентифікатор того, хто затвердив (опціонально)
ApproverComment	nvarchar(255)	Коментар затверджувача (опціонально)

Для актуального відображення статусу користувача також було вирішено зробити динамічну вибірку статусу із бази даних. Для цього було використано SQL функцію, яка дає змогу включати результат її виконання в додатково під час звичайної вибірки даних. Функція аналізує актуальні лікарняні, відпустки та робочі сесії щоб визначити статус користувача.

Функція має такий вигляд:

```
CREATE FUNCTION [dbo].[GetCurrentUserStatus] (@UserId UNIQUEIDENTIFIER)
RETURNS SMALLINT
AS
BEGIN
    DECLARE @CurrentStatus SMALLINT;

    IF EXISTS (
        SELECT 1
        FROM SickLeaves
        WHERE UserId = @UserId
        AND [Start] <= CAST(GETDATE() AS DATE)
        AND [End] >= CAST(GETDATE() AS DATE)
    )
    BEGIN
        SET @CurrentStatus = 1; -- Ill
    END
    ELSE
    BEGIN
        IF EXISTS (
            SELECT 1
            FROM VacationRequests
            WHERE UserId = @UserId
            AND [Start] <= CAST(GETDATE() AS DATE)
            AND [End] >= CAST(GETDATE() AS DATE)
            AND [Status] = 1 -- Approved vacation request
        )
        BEGIN
            SET @CurrentStatus = 2; -- Vacation
        END
        ELSE
        BEGIN
            SET @CurrentStatus = 0; -- Working
        END
    END

    RETURN @CurrentStatus;
END
GO
```

Використання такого підходу у визначенні статусу користувача змогло допомогти зменшити кодову базу проекту за рахунок відсутністю великої

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

кількості коду, пов'язаного із синхронізацією статусу користувача. Це сприяє покращенню кодової бази, робить простір для помилки значно меншим та зменшує кількість колонок в таблиці користувача.

Таким чином нам вдалося спроектувати базу даних, що зможе задовільнити потреби вубзастосунку керування часом та в подальшому системи управління персоналом.

2.3. Проектування та реалізація алгоритмів роботи системи

Для забезпечення усіх вимог поставлених до вебзастосунку, потрібно спроектувати основні алгоритми роботи системи. Загальний алгоритм роботи системи можна описати як безперервну взаємодію користувача з вебінтерфейсом для, який в свою чергу використовуючи потоків даних сигналізує клієнтській логіці про необхідність створення запиту до серверу. Сервер, використовуючи GraphQL, обробляє запит використовуючи внутрішню бізнес логіку, за потреби використовує базу даних для діставання чи мутації даних. Результатом є надання відповіді клієнтській частинні застосунку після чого нові дані оновлюють стан системи і перемальовують інтерфейс.

Тож розглянемо загальний алгоритм роботи на конкретному прикладі за допомогою діаграми активності (див. рис. 2.3). Також для аналізу логіки виконання операцій, реакції на події та обміну повідомленнями між об'єктами в ході взаємодії, що налає зрозуміле та інтуїтивне уявлення про порядок виконання дій у системі в конкретних умовах (див. рис. А.2).

Для прикладу візьмемо додавання нового користувача в систему.

1. Користувач з дозволом на додавання нових користувачів до системи:

- 1.1. Користувач авторизується на сайті.
- 1.2. Користувач переходить до панелі керування користувачами.
- 1.3. Користувач створює запит на додавання користувача.
- 1.4. Система перевіряє, чи має користувач дозвіл.

1.4.1. Якщо ні, операція відхиляється.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

- 1.4.2. Якщо так, то відображається форма для введення даних нового користувача.
- 1.5. Система перевіряє правильність даних.
 - 1.5.1. Якщо ні, відображається повідомлення про помилку.
 - 1.5.2. Якщо так, новому користувачеві буде надіслано електронний лист.
2. Користувач, якого потрібно додати:
 - 2.1. Отримує електронний лист із посиланням для реєстрації.
 - 2.2. Користувач переходить за посиланням.
 - 2.3. Система перевіряє посилання. Якщо перевірка не вдається, відображається повідомлення про помилку. Якщо перевірка пройшла успішно, дає доступ до введення паролю.
 - 2.4. Користувач вводить свій пароль.
 - 2.5. Система перевіряє правильність пароля. Якщо ні, відображається повідомлення про неправильний пароль. Якщо так, користувач додається до системи.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

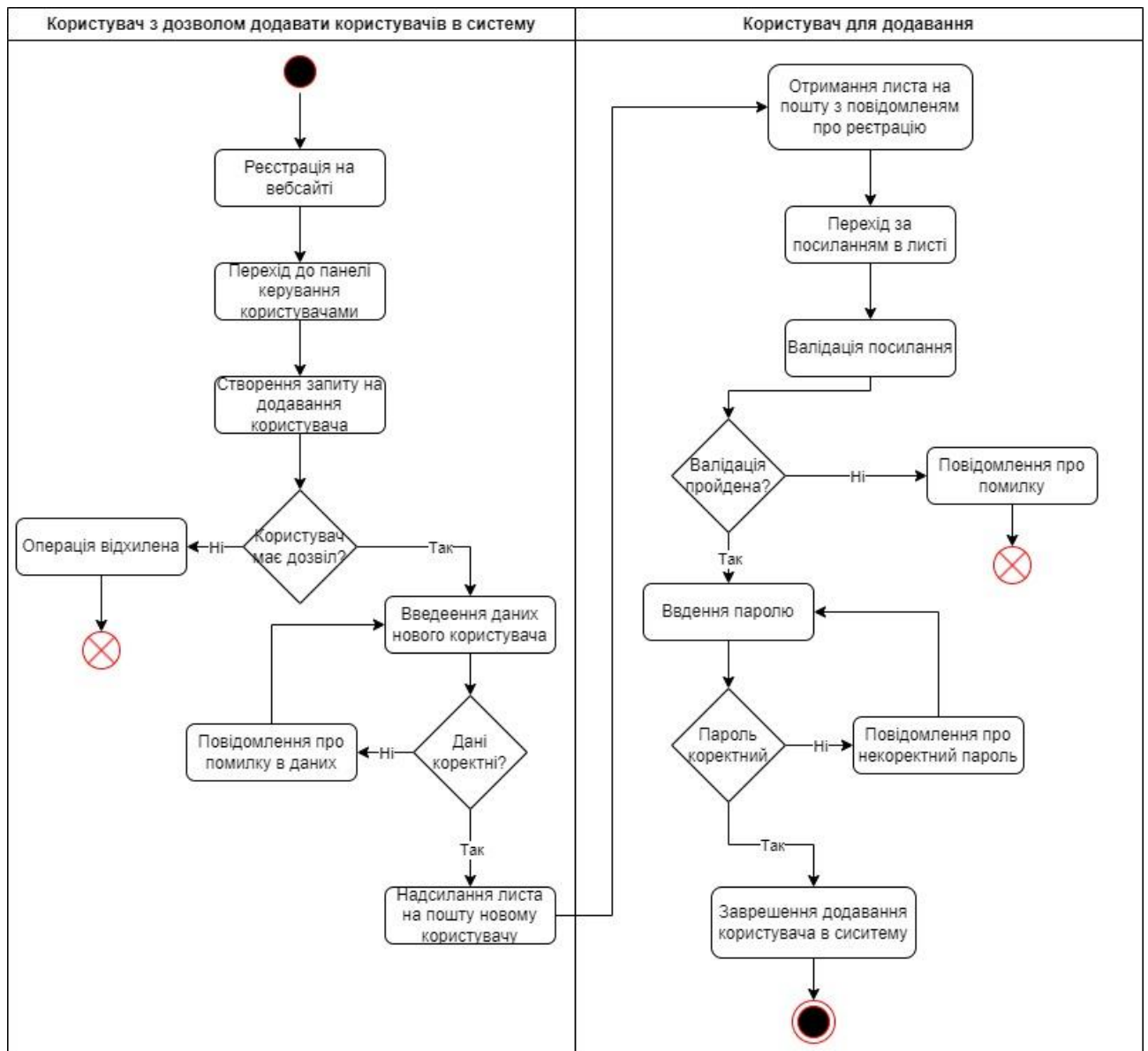


Рисунок 2.3 – Діаграма активності для додавання нового користувача в систему

Розглянемо ще один приклад реалізації алгоритму системи – створення запиту на відпустку (рис. 2.4). Діаграму послідовності для цього алгоритму можна знайти в додатках (див. рис. А.3).

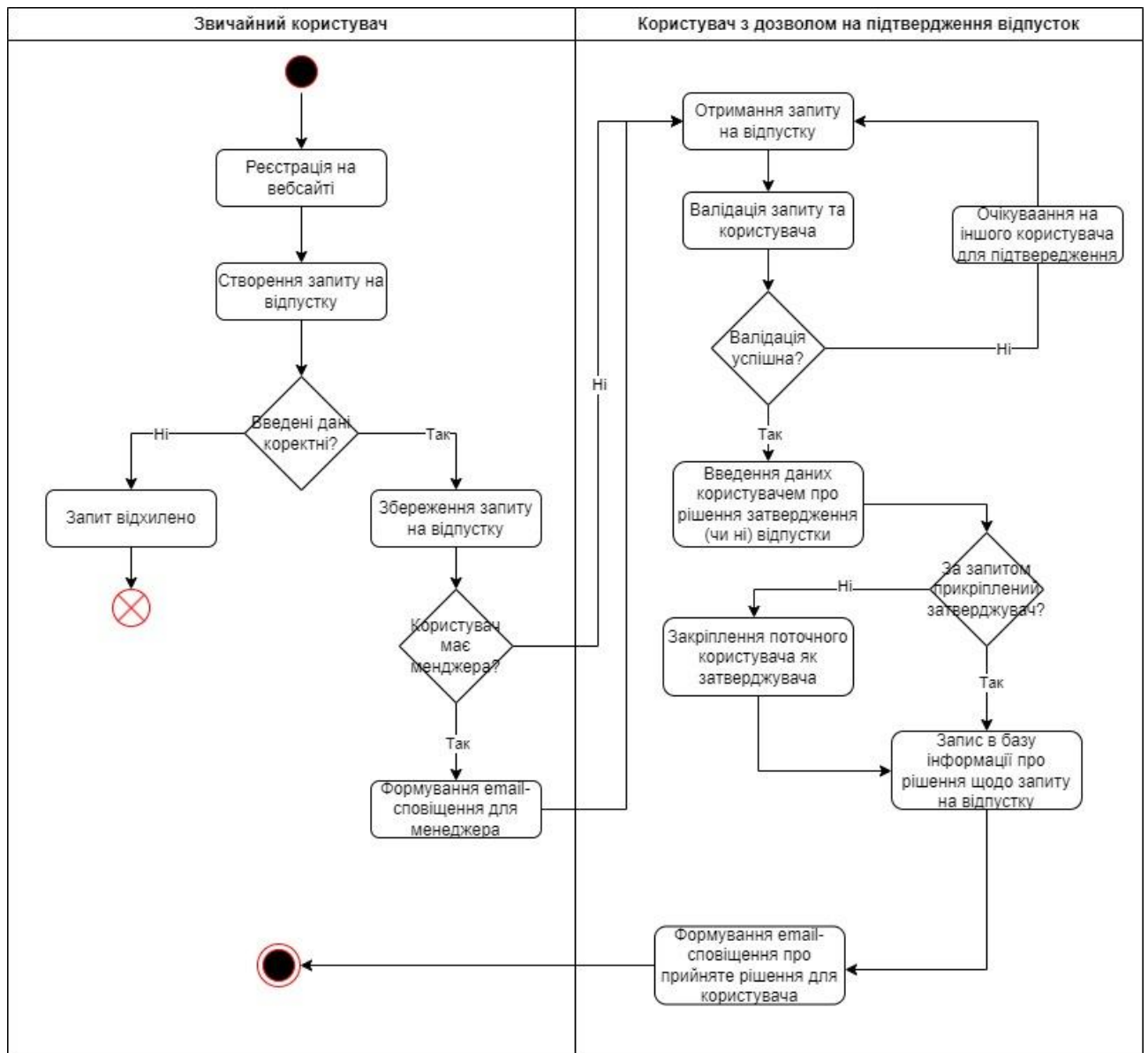


Рисунок 2.5 – Діаграма активності для створення і затвердження запиту на відпустку

1. Звичайний користувач:

- 1.1. Авторизується на сайті.
- 1.2. Створює запит на відпустку.
- 1.3. Система валідує запит:
 - 1.3.1. Якщо не коректний, то запит відхилено.
 - 1.3.2. Якщо коректний, то запит зберігається.
- 1.4. Система перевіряє чи у користувача є менеджер:
 - 1.4.1. Якщо менеджера немає, то нічого не відбувається.

1.4.2. Якщо менеджер присутній, то система формує email-сповіщення про відпустку для менеджера.

2. Користувач із дозволом на затвердження відпустки:

- 2.1. Користувач отримує запиту про відпустку, що очікує затвердження (через лист чи систему).
- 2.2. Користувач переглядає список очікуваних відпусток.
- 2.3. Система перевіряє чи користувач може затвердити відпустку:
 - 2.3.1. Якщо не може, то відпустка очікує іншого менеджера.
 - 2.3.2. Якщо може, то користувач отримує доступ до форми.
- 2.4. Користувач заповнює і надсилає форму.
- 2.5. Система перевіряє чи за запитом на відпустку закріплений менеджер:
 - 2.5.1. Якщо закріплений, то пропускає встановлення поточного користувача як затверджувача.
 - 2.5.2. Якщо закріплений, то встановлює поточного користувача як затверджувача.
- 2.6. Система записує в базу інформацію щодо рішення затверджувача.
- 2.7. Система надсилає email-сповіщення для користувача, який залишав запит на відпустку.

2.4. Реалізація застосунку веб-орієнтованої системи керування робочим часом

Розглянемо деякі фрагменти коду, що можуть бути складними для розуміння чи потребувати додаткового пояснення. Більшість частин коду не потребують додаткового пояснення, так як при написанні проекту, особливу увагу приділялося читабельності коду.

Ось наприклад фрагмент GraphQL.net коду, який відповідальний за обробку запиту на затвердження відпустки.

Цей фрагмент є частиною VacationMutation типу GraphQL тому при запиті цієї операції нам також потрібно вказати правильний запит, де має бути вказано що це запит-мутація, спрямований на модифікацію даних, шлях до потрібної

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

мутації, аргументи що ми йому надаємо, та поля які ми хочемо отримати у відповідь, що нашому випадку відсутні. Запит буде мати такий вигляд:

```
mutation ($input: VacationApproveInputType!) {  
  vacation {  
    sendVacationReview(input: $input)  
  }  
}
```

Цей запит потрапляє до шару API серверу, який оброблює запити з клієнту.

Розглянемо відповідний обробник для цього запиту:

```
Field<BooleanGraphType>("SendVacationReview")  
  .Argument<NonNullGraphType<VacationApproveInputType>>("input")  
  .Resolve()  
  .WithScope()  
  .WithService<VacationService>()  
  .ResolveAsync(async (context, service) =>  
  {  
    var input = context.GetArgument<VacationApproveInput>("input");  
    var user = context.GetUser()!;  
  
    await service.ApproverUpdateVacationAsync(input, user);  
  
    return true;  
  }).AuthorizeWithPolicy(nameof(UserPermissions.ManageVacations));
```

1. В нашому прикладі поле має повертати тип BooleanGraphType (bool в .Net), що насправді потрібний лише для сповіщення клієнту що запит був завершений.
2. Рядком нижче вказані аргументи які він приймає, в нашому випадку це не пустий об'єкти типу VacationApproveInputType (VacationApproveInput в .Net) з назвою input.
3. Далі реалізований механізм DependencyInjection, який присутній в ASP.NET програмах, для цього конкретного поля. В нашому випадку він потребує VacationService, який буде переданий, як другий аргумент в Resolver() методі.
4. Тепер присутня сама логіка обробки, яка написана у вигляді лямбда-функції всередині ResolveAsync методу, тут ми дістаємо потрібні аргументи, дістаємо користувача з контексту і викликаєм потрібний метод сервісу, який раніше ввели за допомогою Dependency Injection та повертаємо результат.
5. AuthorizeWithPolicy метод потрібний щоб захистити обробник за допомогою політики, в нашому випадку це наявність дозволу на керування відпустками.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Після первинної обробки запиту, за виконання операції береться шар логіки. Тепер розглянемо метод який ми викликали в обробнику GraphQL запиту:

```
public async Task ApproverUpdateVacationAsync(VacationApproveInput
vacationApproveInput, User user)
{
    var vacation = await vacationRepository.GetSingleAsync(vacationApproveInput.Id)
        ?? throw new ExecutionError("Vacation not found");

    if (vacation.ApproverId.HasValue && user.Id != vacation.ApproverId.Value)
        throw new InvalidOperationException("This vacation request require another
approver");

    var newStatus = vacationApproveInput.IsApproved ? VacationRequestStatus.Approved :
VacationRequestStatus.Declined;
    await vacationRepository.ApproverUpdateVacationAsync(vacationApproveInput,
newStatus, user.Id);

    var updatedVacation = await vacationRepository.GetSingleAsync(vacation.Id);
    await UpdateVacationBalanceAsync(vacation, updatedVacation);

    await vacationMailer.SendVacationResponseAsync(user, vacation.UserId);
}
```

1. Першим кроком метод валідує дані що прийшли, аби не допустити помилок, на практиці клієнтська частина зроблена таким чином, щоб запобігти виникненню ситуацій коли валідація провалиться, але нам потрібно бути впевненим що у разі помилки дані не будуть записані.
2. Далі записує дані за допомогою відповідного методу репозиторію.
3. Отримує оновлений запит на відпустку за допомогою репозиторію.
4. Оновлює баланс відпусток через інший приватний метод UpdateVacationBalanceAsync.
5. Надсилає листа для користувача, з результатами розгляду його запиту на відпустку.

Як ми бачимо, під час виконання операцій ми неодноразово звертаємося до третього шару – шару даних. Розглянемо один приклад того як виконується взаємодія із базою даних і запис інформації в базу даних за допомогою Dapper:

```
public async Task ApproverUpdateVacationAsync(
    VacationApproveInput input,
    VacationRequestStatus status,
    Guid approverId)
{
```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

```

var query = $"UPDATE {TableName} SET " +
    $"[{nameof(VacationRequest.ApproverId)}] = @{nameof(approverId)}, " +
    $"[{nameof(VacationRequest.Status)}] = @{nameof(status)}, " +
    $"[{nameof(VacationRequest.ApproverComment)}] = " +
    $"[{nameof(input.ApproverComment)}] " +
    $"WHERE [{nameof(VacationRequest.Id)}] = @{nameof(input.Id)}";

using var connection = context.GetConnection();

await connection.ExecuteAsync(query, new
{
    approverId,
    status,
    input.ApproverComment,
    input.Id
});
}

```

1. спочатку ми формує сам SQL-запит за допомогою отриманих даних;
2. потім створюємо з'єднання зі базою даних;
3. виконуємо створений запит.

Отже, тепер ми бачимо наскільки зручно використовувати трьох-шарову архітектуру серверу для обробки даних. Ми чітко поділили обов'язки між кожним шаром і це дозволяє нам набагато легше розширювати і підтримувати кодову базу.

Варто зауважити що іноді шар логіки можна оминати, бо не завжди потрібно виконувати якусь логіку, іноді нам потрібно просто дістати якусь інформацію з бази даних.

Розглянемо для прикладу поле Status в GraphQL для типу User. Для того щоб GraphQL знав як обробити кожне поле для користувача, яке у нього просять, йому потрібно вказати як їх оброблювати. Ось так виглядає UserType, що був створений саме з цією метою:

```

public sealed class UserType : ObjectGraphType<User>
{
    public UserType()
    {
        Name = "User";

        Field<NonNullGraphType<IdGraphType>>(nameof(User.Id))
            .Description("Id field for user object")
            .Resolve(context => context.Source.Id);

        Field<NonNullGraphType<StringGraphType>>(nameof(User.Email))
            .Description("Email field for user object")
            .Resolve(context => context.Source.Email);

        Field<NonNullGraphType<StringGraphType>>(nameof(User.JobTitle))
            .Resolve(context => context.Source.JobTitle);
    }
}

```

```

...
        Field<NonNullGraphType<UserStatusEnum>>("Status")
            .Description("Status for user object")
            .Resolve()
            .WithScope()
            .WithService<UserRepository>()
            .ResolveAsync(async (context, repository) => await
repository.GetStatusAsync(context.Source.Id));
...
    }
}

```

Тут ми бачимо опис кожного поля, що присутній в імплементованому нами типу User, але клас User не має поля Status, а GraphQL UserType, його має. Це було досягнуто завдяки створенній sql-функції для динамічного визначення статусу користувача (див. розділ 2.2.) і її виконання завдяки методу репозиторію:

```

public async Task<UserStatus> GetStatusAsync(Guid userId)
{
    var query = $"SELECT [dbo].GetCurrentUserStatus(@{nameof(userId)})";
    using var connection = context.GetConnection();
    return await connection.QuerySingleAsync<UserStatus>(query, new { userId });
}

```

В цьому випадку відсутня логіка, просто створення запиту, тому ми переходимо від шару API до шару даних уникаючи шару логіки, аби не створювати дублювання коду. І якщо на клієнті не потрібен статус користувача, то ми маємо змогу просто пропустити це поле, аби не виконувати зайвого запиту до бази даних.

Тепер розглянемо статистично-аналітичний модуль. Він виконує логіку для отримання статистичних чи аналітичних даних. Розглянемо як відбувається обрахунок відпрацьованої кількості днів, для чого було створено окремий клас (повний лістинг наведено в додатку Б):

```

async Task<HoursSummaryResult> CalculateHoursSummaryAsync(IEnumerable<User> users,
DateTime from, DateTime to)
{
    var generalWorkingDays = GetWorkingDays(from, to);

    var holidays = await holidayRepository.GetByDateRangeAsync(from, to);
    var dayOffDays = GetDayOffDays(holidays, from, to);
    var shortDays = GetShortDays(holidays, from, to);

    var userIds = users.Select(u => u.Id);
    var vacations = await vacationRepository.GetUsersVacationsInRangeAsync(userIds,
from, to);
    var sickLeaves = await sickLeaveRepository.GetSickLeaveInRangeAsync(userIds, from,
to);
}

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

```

var workSession = await workSessionRepository.GetInRangeAsync(userIds, from, to);

var individualHoursSummaries = users.Select(u =>
{
    var hoursPerDay = FullHoursPerDay * (u.EmploymentRate / 100.0);
    var totalHoursToWork = generalWorkingDays * hoursPerDay;

    totalHoursToWork -= dayOffDays * hoursPerDay;
    totalHoursToWork -= GetShortDaysHours(holidays, from, to, hoursPerDay);

    var vacationDays = GetVacationDays(vacations, u.Id, from, to);
    totalHoursToWork -= vacationDays * hoursPerDay;

    var sickLeaveDays = GetSickLeaveDays(sickLeaves, u.Id, from, to);
    totalHoursToWork -= sickLeaveDays * hoursPerDay;

    var actualHours = GetActualHours(workSession, u.Id, from, to);
    var percent = actualHours / totalHoursToWork;

    return new IndividualHoursSummary(u, totalHoursToWork, actualHours, percent,
        vacationDays, sickLeaveDays);
});

var generalHolidaysDays = dayOffDays + shortDays;

return new(generalWorkingDays, generalHolidaysDays, dayOffDays,
    individualHoursSummaries);
}

```

1. Обраховується загальна кількість робочих днів у заданому діапазоні часу за допомогою методу GetWorkingDays(from, to).
2. Отримання даних про свята та скорочені дні:
 - 2.1.Отримуються дати свят у вказаному діапазоні за допомогою репозиторію holidayRepository.
 - 2.2.Обраховуються дні відпочинку на основі свят через метод GetDayOffDays(holidays, from, to).
 - 2.3.Обраховуються скорочені дні через метод GetShortDays(holidays, from, to).
3. Отримання даних про відпустки, лікарняні та робочі сесії:
 - 3.1.Отримуються ідентифікатори користувачів.
 - 3.2.Викликаються відповідні репозиторії для отримання відпусток, лікарняних та робочих сесій у вказаному діапазоні часу:
4. Обрахунок індивідуальних зведень для кожного користувача:
 - 4.1.Для кожного користувача обраховується кількість годин на день, враховуючи його ставку зайнятості.
 - 4.2.Обраховується загальна кількість годин, які користувач має відпрацювати.
 - 4.3.Віднімаються години на основі днів відпочинку та скорочених днів.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

- 4.4.Віднімаються години на основі відпусток та лікарняних днів.
 - 4.5.Обраховуються фактичні години, які користувач відпрацював.
 - 4.6.Обраховується відсоток відпрацьованих годин від запланованих.
 - 4.7.Створюється об'єкт IndividualHoursSummary для кожного користувача з розрахованими даними.
5. Повертається об'єкт HoursSummaryResult, що містить загальну кількість робочих днів, кількість святкових днів, кількість днів відпочинку та індивідуальні зведення для кожного користувача.

Таким чином, метод забезпечує детальний обрахунок відпрацьованих годин для кожного користувача з урахуванням святкових днів, відпусток та лікарняних, його годинної ставки та інших деталей як виключення годин, які вже були виключені під час відрядження усіх днів суботи та неділі.

Тепер розглянемо клієнтську частину та деталі її реалізації. Одним із найскладніших завдань було реалізувати систему підвантажування сторінок.

Підвантажування сторінок це певний механізм, який дозволяє зробити запит на отримання інформації із серверу, для функціонування сторінки. При цьому навігація не відбудеться доки запит не завершиться.

Найбільшою перевагою такого підходу є майже повня ізоляція всієї логіки роботи навігації і підвантаження від компонентів, які займаються промальовуванням вебінтерфейсу. Зі сторони компонента, відбувається лише перехід за посиланням, в той час як система аналізує нове посилання, робить запит до серверу, якщо це потрібно, формує данні для сторінки і встановлює їх для загального стану системи, знаходить і промальовує потрібну сторінку. При цьому компонент не має жодної додаткової логіки, на момент завантаження він використовує дані що вже будуть в стані системи.

В реалізації такого підходу дуже сприяло реактивне програмування, яке працювало в тісній взаємодії із внутрішнім станом системи. Тож розглянемо принцип роботи цієї системи.

Почнемо із реалізації стану компоненту за допомогою такого елемента як slice із бібліотеки керування станом Redux та Redux toolkit:

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

```

export interface GeneralAppState {
  page: Page | null;
  urlParams: Params;
  isLoading: boolean;
  isPageLoading: boolean;
  isInitialLoading: boolean;
}

const InitialState: GeneralAppState = {
  page: null,
  urlParams: {},
  isLoading: false,
  isPageLoading: false,
  isInitialLoading: true,
};

export const GeneralAppSlice = createSlice({
  name: 'general',
  initialState: InitialState,
  reducers: {
    setPageState: (state, action: PayloadAction<Page>) => {
      state.page = action.payload;
      state.isPageLoading = false;
    },
    setUrlParams: (state, action: PayloadAction<Params>) => {
      state.urlParams = action.payload;
    },
    setLoadingIndicator: (state, action: PayloadAction<boolean>) => {
      state.isLoading = action.payload;
    },
    setInitialLoading: (state, action: PayloadAction<boolean>) => {
      state.isInitialLoading = action.payload;
    },
    setPageLoading: (state, action: PayloadAction<boolean>) => {
      state.isPageLoading = action.payload;
    },
  },
});

export const {
  setPageState,
  setUrlParams,
  setLoadingIndicator,
  setInitialLoading,
  setPageLoading,
} = GeneralAppSlice.actions;

export default GeneralAppSlice.reducer;

```

Ключовими елементами є його змінні стану, а саме:

1. `page` – елемент, який зберігає усю інформацію сторінки, він не має чіткої структури, це інтерфейс і будь-яка сторінка може додавати в нього нові поля. Наприклад, якщо перейти на сторінку профілю, то відповідний розширений інтерфейс `ProfilePageState` замінить `page`, який був до цього. Тепер поле `page` буде містити інформацію про дані профілю користувача, після переходу на

нову сторінку, поле буде перезаписане. Але базовий інтерфейс все ж має одне обов'язкове поле, яке містить назву сторінки, яку потрібно промалювати.

2. `urlParams` – зберігає додаткові параметри, які були записані в `url` сторінки, наприклад `id` користувача, на сторінці профілю якого ви знаходитесь.
3. `isLoading` – ідентифікатор, який відповідальний за відображення елементу завантаження.
4. `isPageLoading` – ідентифікатор, який потрібний для блокування перемальовування сторінки доки дані для сторінки не будуть сформовані.
5. `isInitialLoading` – ідентифікатор початкового завантаження, потрібний для завантаження даних із серверу, які лежать в основі, наприклад отримання користувача, якщо в нього був збережений `refreshToken`;

Маючи можливість поставити в стан програми будь-яку потрібну сторінку і дані, ми можемо не тільки ізолюємо логіку від відображення, а й забезпечуємо актуальність даних, бо кожен раз переходячи на сторінку профілю ми будемо повторно робити запит, що буде завжди тримати дані свіжими.

Тепер розглянемо як саме ми визначаємо переадресацію на новий шлях, знаходимо потрібний обробник, робимо запит за потреби і оновлюємо стан сторінки. В цьому нам допоміг `epic`, який є частиною бібліотеки `Redux-Observable`, що дає нам змогу керувати побічними ефектами в `Redux` за допомогою реактивного програмування `RxJS`.

```
export const loadPageStateEpic: Epic<LocationChangedAction> = (action$, state$, dependencies) =>
  action$.pipe(
    ofType(LOCATION_CHANGE),
    switchMap(({ payload: { location } }) => {
      const routes = routeBuilder.allRoutes;
      const possibleMatches = matchRoutes(routes, location.pathname);

      if (!possibleMatches || possibleMatches.length !== 1)
        return of(push(routeBuilder.routes.notFound));

      const match = possibleMatches[0];
      const pattern = possibleMatches[0].route.path;
      const handler = getPageHandler(pattern);

      const routeData: RouteData = {
        location,
        params: match.params,
        pattern,
      };

      return state$.pipe(
```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    first(({ general }) => !general.isInitialLoading),
    switchMap(state => handler(state, dependencies, routeData).pipe(
      concatMap(page => of(
        setUrlParams(match.params),
        setPageState(page),
        setPageLoading(false),
      )),
      startWith(setPageLoading(true)),
    )),
  ),
},
);

```

Розглянемо покроково дії що виконує епік:

1. Епіс підписується на action (подія в Redux), який відповідний за зміну шляху в url. Тобто як тільки шлях змінився, епіс реагує на це і починає свою роботу, що гарно пояснює концепції реактивного програмування.
2. Action, що спричинив роботу цього епіку, містить інформацію про новий url-шлях, за допомогою чого ми маємо знайти його шаблон, наприклад «/user/profile/5» буде відповідати шаблону «/user/profile/:userId», де userId дорівнює 5.
3. Після знаходження потрібного шаблону, підготовлюємо дані для передачі його в обробник сторінки (pageHandler), про який розказано нижче. Дані включають актуальний стан системи, всю інформацію маршруту та залежності, такі як клас для взаємодії із API чи об'єкт для створення рорир сповіщень.
4. Тепер коли все готово, ми отримуємо потрібний обробник сторінки (pageHandler), який є унікальним для кожної сторінки. Саме обробник формує дані для сторінки і робить запит на сервер, якщо це потрібно. Наприклад, при коли користувач переходить на сторінку створення відпустки, відпрацьовує наступний обробник:

```

export const vacationCreateHandler: PageHandler<VacationCreatePageState> = (_, { api }) => {
  return
  api.Request<VacationCreateStateResponse>(getVacationCreatePageStateRequest).pipe(
    map(({ data }) => ({
      pageName: PageName.vacationsRequest,
      balanceS: data.vacation.getVacationBalance,
      bookedDays: data.vacation.getBookedDays,
    })),
  ),
};

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

Обробник отримав дані із серверу про наявний баланс відпусток користувача, створив новий об'єкт сторінки і повернув його до епіку.

5. Тепер коли епик має нову сторінку, йому залишається лише встановити її до загального стану застосунку.
6. Крім того епик валідує багато інших випадків що мають вплив на роботу вебзастосунку. Наприклад наступна лінія змушує епик почекати перед виконанням обробника сторінки, якщо первинне завантаження не закінчилося:

```
first(({ general }) => !general.isInitialLoading),
```

Тепер розглянемо компонент, який дістає назву сторінки із стану і відображує потрібний компонент по ключу:

```
function App() {  
  ...  
  
  useInitialRefresh();  
  
  const { isInitialLoading, isPageLoading, page, isInitialSetupCompleted } =  
  useAppSelector(({ general, settings }) => ({  
    isInitialLoading: general.isInitialLoading,  
    isPageLoading: general.isPageLoading,  
    page: general.page,  
    isInitialSetupCompleted: settings.general?.isInitialSetupCompleted,  
  }), shallowEqual);  
  
  ...  
  
  const hidePages = isInitialLoading || isPageLoading || !page;  
  
  if (hidePages)  
    return null;  
  
  ...  
  
  return pages[page.pageName];  
}
```

Коли поле `page` в стані оновилося, це спричиняє перемальовування головного компоненту програми разом із новим компонентом-сторінкою, що записані в окремому файлі `pages` і мають ключ, що встановлюється в обробнику сторінки.

Використання цього механізму доводить наскільки корисними і зручними можуть бути концепції реактивного програмування. Але на цьому його корисність не обмежується.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

Наведемо для прикладу також інший епіс, який є відповідальним за підвантаження актуальної сесії користувача для відображення в компоненті для відслідковування робочих годин:

```
export const getTrackerInfoEpic: Epic = (_, state$, { api }) =>
  state$.pipe(
    filter(({ currentUser, tracker }) => currentUser.isLoggedIn &&
      tracker.requireUpdate),
    exhaustMap(({ currentUser }) => makeTrackerRefreshRequest(api,
      currentUser.user!.id)),
  );

const makeTrackerRefreshRequest = (api: Api, userId: string) =>
  api.Request<GetTrackerInfoResponse>({
    getTrackerInfoRequest,
    {
      userId,
      dayDate: formatDateOnlyString(dayjs()),
    },
  }).pipe(
    map(({ data: { workSession, user } }) => {
      const activeWs = workSession.getActive;
      const workedTime = workSession.getWorkedTimeForDay;
      const dayWorkTimeRate = user.getHoursPerDay;

      return loadTrackerInfo({
        activeWorkSession: activeWs,
        workedTimeForDayMS: workedTime,
        workRatePerDayMS: dayWorkTimeRate,
      });
    }),
  );
```

1. Епіс підписується на загальний стан системи застосунку.
2. Епіс проводить фільтрацію стану, щоб визначити чи треба йому починати роботу. В даному випадку він почне роботи тільки якщо користувач увійшов у систему та трекер потребує оновлення даних актуальної робочої сесії.
3. Якщо стан пройшов фільтрацію, то епіс робить запит до серверу для отримання актуальної сесії користувача.
4. Після отримання відповіді від серверу, епіс встановлює дані в стан системи, що спричинить перемальовування компоненту для відслідковування робочого часу.

Висновки до розділу 2

Під час виконання другого розділу дипломної роботи було проведено детальний аналіз варіантів використання системи, включаючи побудову

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

діаграми варіантів використання і детальним описом прецедентів. Це дало нам змогу оцінити кількість роботи та сформувані вимоги до програмного продукту. Із сформованими вимогами та варіантами використання було побудовано об'єктно-орієнтовану структуру системи та наведено діаграму класів як приклад реалізації програмного продукту.

Після визначення варіантів використання і об'єктно орієнтованої структури системи було проведено роботу із визначенням структури бази даних для забезпечення усіх поставлених вимог. Було наведено схему бази даних із детальним описом її елементів, а також інші її елементи як функція для визначення статусу користувача. Також для було створено міграції за допомогою FluentMigrator для забезпечення актуальності бази даних на різних пристроях. Це допоможе забезпечити актуальну структуру бази даних навіть коли над проектом паралельно працює команда.

Було проведено проектування та реалізацію алгоритмів роботи вебзастосунку. Було сформовано алгоритм загальної взаємодії частин вебзастосунку та окремих її елементів. Було наведено приклади роботи алгоритмів для створення нового користувача та подання і затвердження запиту на відпустку. Для опису роботи алгоритмів використовували діаграми активностей та послідовності.

Фінальним етапом другого розділу стала повна реалізація усіх функцій системи із виконанням усіх поставлених вимог та архітектурних шаблонів. Було наведено приклади реалізації окремих частин програмного продукту, та найскладніших елементів, які можуть потребувати пояснення. Було доведено ефективність застосованих архітектурних шаблонів, рішень та обраних інструментів на конкретних прикладах реалізації.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ ДЛЯ КЕРУВАННЯ РОБОЧИМ ЧАСОМ

3.1. Порядок встановлення та налаштування параметрів системи

Для встановлення вебзастосунку потрібно правильно визначити схему розгортання її елементів. Так як ми використовуємо вебзастосунок, то звичайному користувачу необхідно лише мати посилання на вебсторінку. Залежно від потреб компанії, застосунок також може бути встановлений по локальній мережі. Загалом вебзастосунок орієнтований на встановлення окремої копії для кожного клієнта.

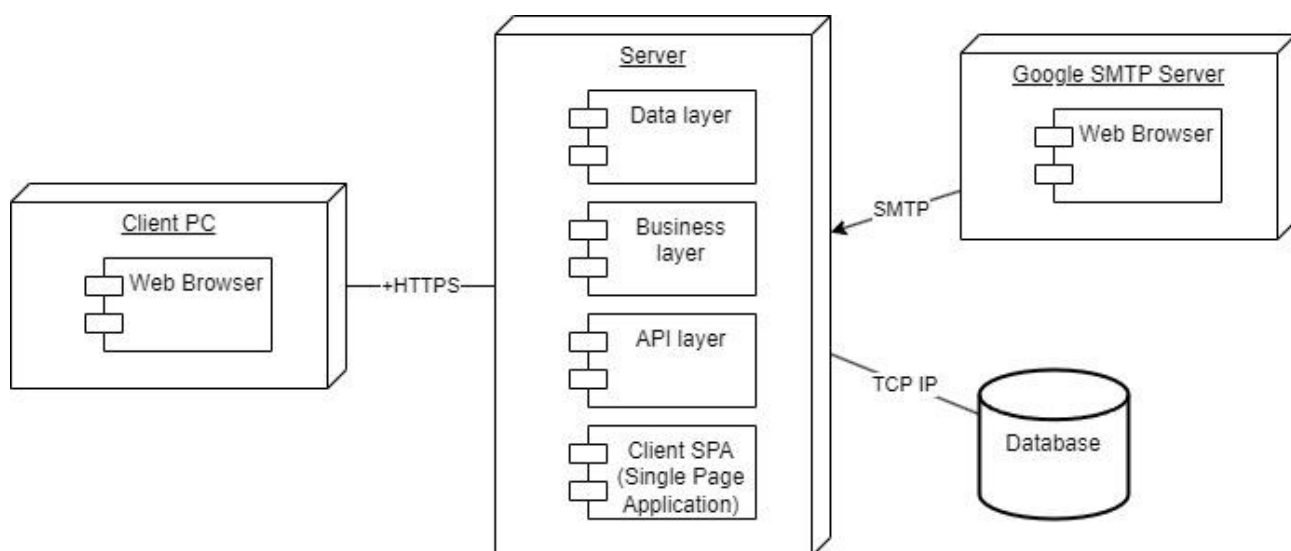


Рисунок 3.1- Діаграма розгортання

Ключовими елементами системи є:

- **Google SMTP Server:** Надає можливість надсилати email листи. Цей елемент буде використовуватися сервером системи, і його не потрібно встановлювати чи хостити, оскільки ми звертаємося до існуючого серверу, який компанія Google надає для використання. Це забезпечує надійність і безперебійність надсилання листів, а також знижує витрати на утримання власного поштового сервера;
- **База даних:** Потрібна для зберігання інформації, яку буде використовувати наш застосунок. Важливо забезпечити надійність та безпеку бази даних, оскільки вона містить критичні дані про користувачів і їх робочий час;

- **Сервер:** Містить сам вебзастосунок з усіма його модулями та функціоналом. У нашому випадку ми використовуватимемо один сервер для серверної і клієнтської частини, що пов'язано з наявними ресурсами. Такий підхід спрощує адміністрування і знижує витрати, однак при потребі можна розгорнути обидві частини на різних серверах для підвищення продуктивності і масштабованості. Сервер повинен мати достатньо ресурсів для обробки запитів користувачів і забезпечення стабільної роботи системи;
- **Пристрій клієнта, який підтримує веб-браузер:** Цей елемент є незалежним, користувач може використовувати будь-який сучасний браузер (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge тощо) для роботи з вебзастосунком. Важливо забезпечити сумісність застосунку з різними браузерами та пристроями для зручності користувачів.

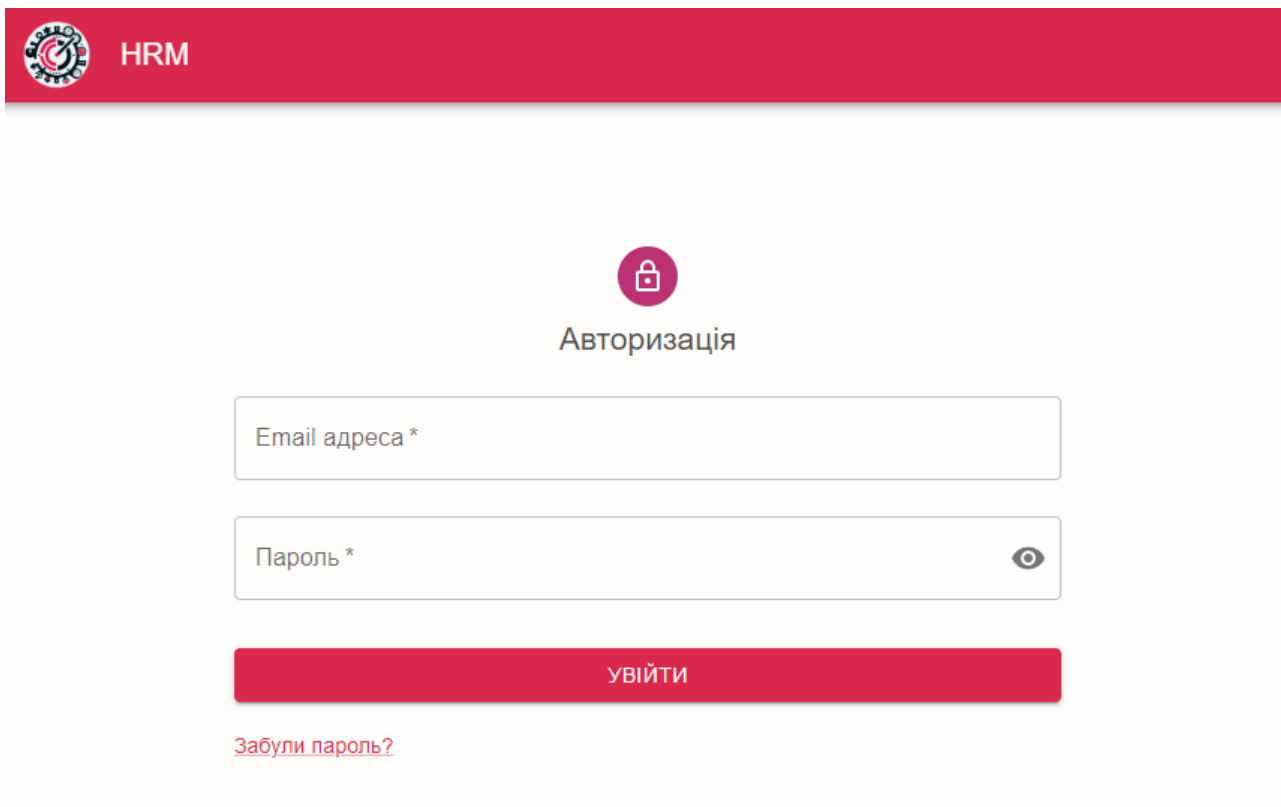
3.2. Структура інтерфейсу веб-орієнтованої системи керування робочим часом

Взаємодія користувачів із системою відбувається через вебзастосунок. Більшість функцій системи вимагають від користувача бути авторизованим, адже це платформа зав'язана на збиранні інформації для конкретної людини. Неавторизовані користувачі навіть не зможуть користуватися навігацією чи іншими елементами системи.

Для користувача, що не був авторизований в системі існує всього декілька сторінок, які так чи інакше пов'язані із авторизацією. До складу таких сторінок входять сторінка авторизації, сторінка відправки листа у випадку забутого паролю, встановлення паролю.

Якщо ж користувача було зареєстровано в системі і він отримав листа на пошту із посиланням на адресу реєстрації, то він отримає доступ до сторінки реєстрації, де зможе вказати про себе деякі додаткові дані, як номери телефону, додаткову пошту для зв'язку, дату народження і так далі. Але результатом завершення реєстрації є встановлення паролю і отримання доступу до системи.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		64



HRM

Авторизація

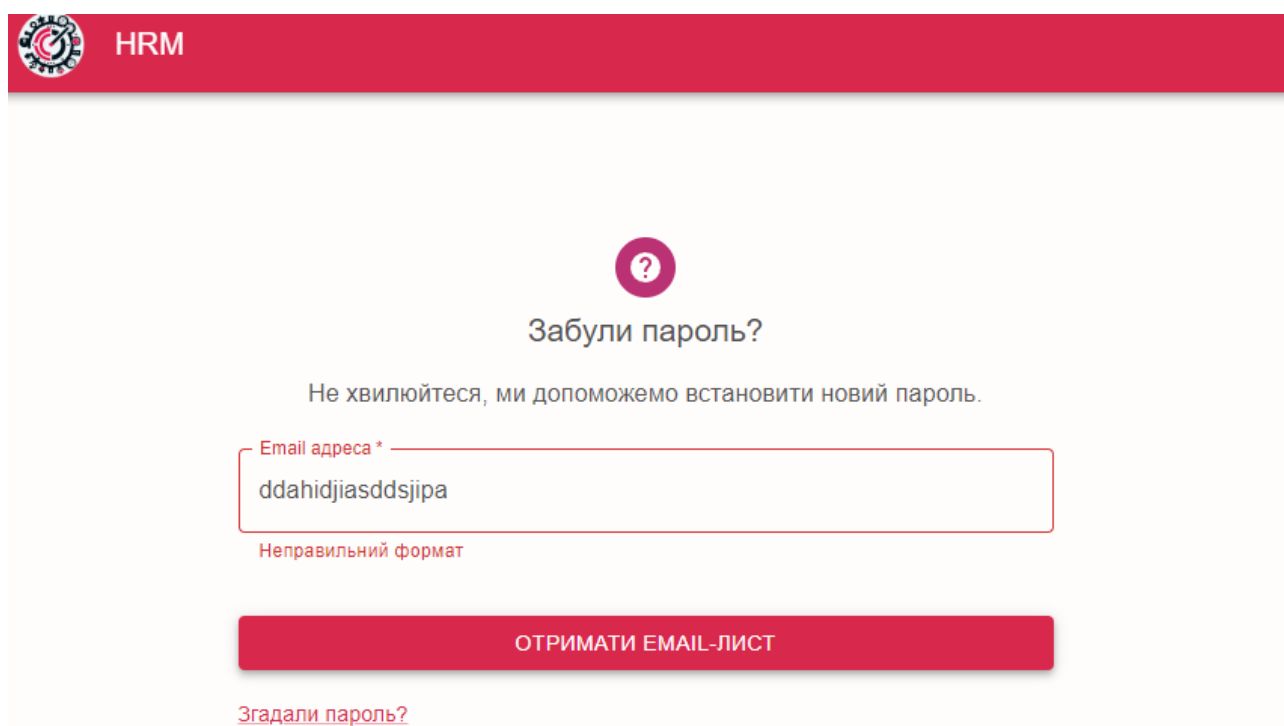
Email адреса *

Пароль *

УВІЙТИ

[Забули пароль?](#)

Рисунок 3.2 – Сторінка авторизації



HRM

Забули пароль?

Не хвилюйтеся, ми допоможемо встановити новий пароль.

Email адреса *

ddahidjiasddsjipa

Неправильний формат

ОТРИМАТИ EMAIL-ЛИСТ

[Згадали пароль?](#)

Рисунок 3.3 – Сторінка відновлення паролю

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		65



Chir, вітаємо на платформі управління персоналом

Вітаємо в <назва компанії>, ми з нетерпінням чекали вас в нашій команді!

Залишилося ще трошки перш ніж ви зможете перейти до повноцінної роботи.

Зараз вас чекає реєстрація на платформу управління персоналом. Тут ви зможете відслідковувати вашу кількість робочих годин, брати відпустки чи лікарняні, отримати інформацію та аналіз ваших дій в системі.

ДАЛІ

Рисунок 3.4 – Перший крок при завершенні реєстрації нового користувача у вебзастосунку

Якщо користувач авторизований, то отримує доступ до більшості елементів системи. Проте деякі сторінки чи її елементи можуть бути недоступними через відсутність потрібного дозволу у користувача.

Головна сторінка це перше що бачить користувач коли заходить до вебсайту. Саме ця сторінка формує настільки важливе перше враження, тому дуже важливо зробити її привабливою. Проте важливо зробити її не тільки гарною але зручною, наприклад додати пряме посилання на сторінки, які можуть бути не доступні на пряму через головне меню навігації. Також було прийнято рішення додати пошук користувача для швидкої навігації до профілю.

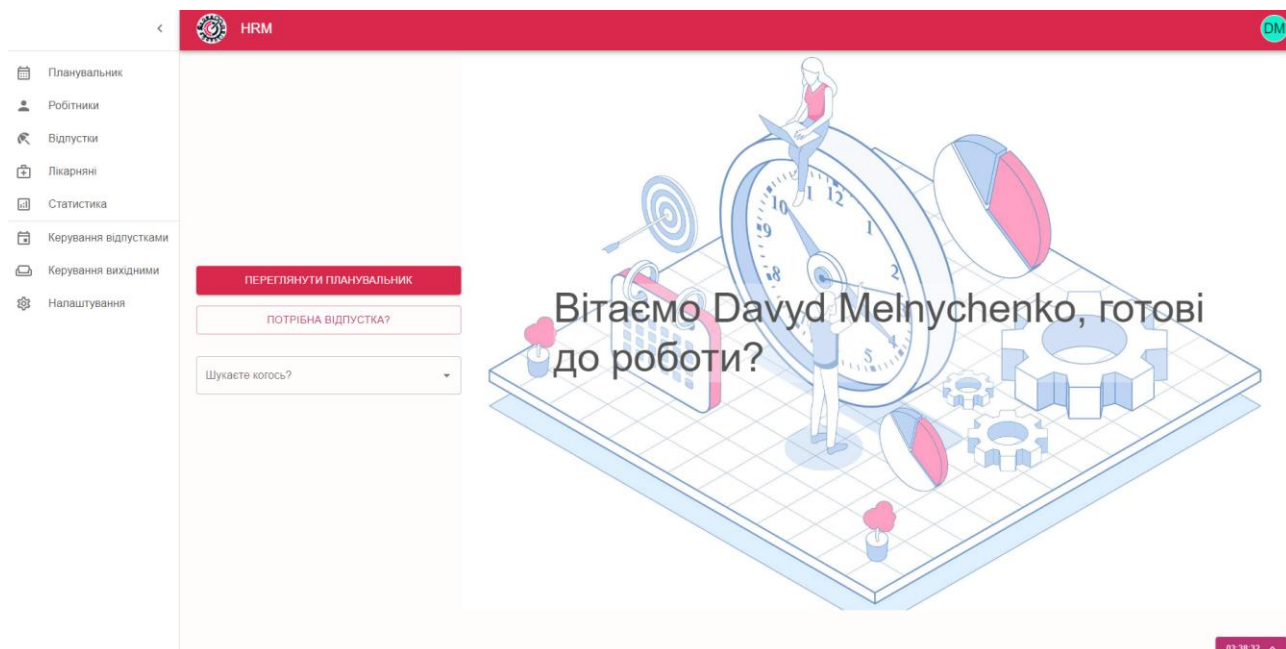


Рисунок 3.5 – Головна сторінка

Одним із найважливіших елементів системи є трекер, який надає можливість користувачу засікати час, який потім буде збережений в системі. Це один із ключових елементів, і доступ до нього має бути присутній завжди, тому було прийнято рішення розмістити його у вигляді кнопки, що завжди зафіксована на дні сторінки.

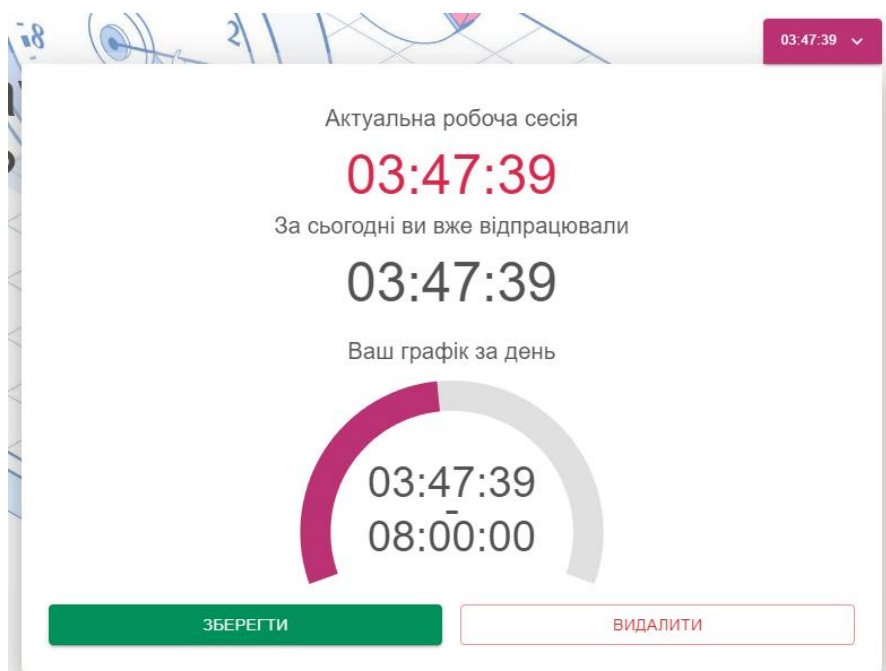


Рисунок 3.6 – Відображення відкритої зафіксованої панелі трекера

При нажиманні кнопки відбудеться відображення додаткової панелі, де буде відображені такі елементи як актуальна робоча сесія, кількість

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

відпрацьованого часу за сьогодні, шкала для демонстрації потрібного відпрацьованого часу за сьогодні.



Рисунок 3.7 - Відображення закритої зафіксованої панелі трекера в активному та не активному варіантах

Для користувача з усім дозволами, є можливість налаштовувати систему відповідно до власних потреб. Хоч і кількість налаштувань в теперішній реалізації невелика, проте сторінка вже є повністю функціонуючою. Для того щоб мати чіткий розподіл налаштувань, було додано вкладки, що також полегшує навігацію по сторінці налаштувань.

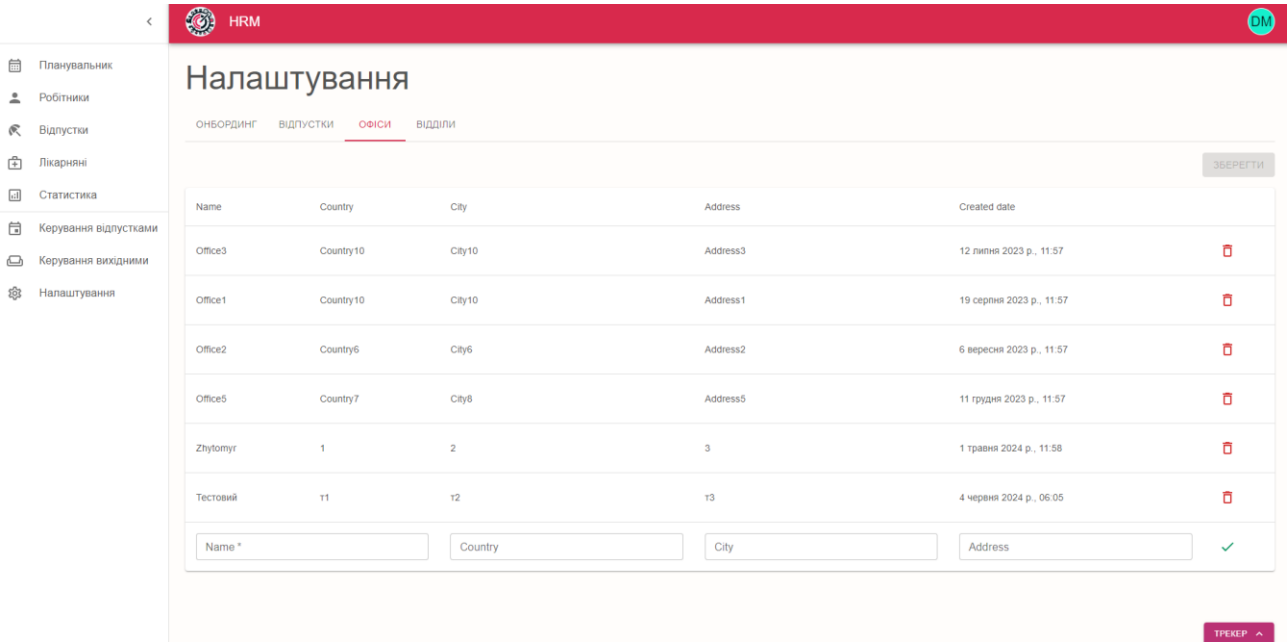


Рисунок 3.8 – Сторінка налаштувань.

Однією із найвідвідуваніших сторінок неодмінно буде сторінка із переліком усіх працівників. Тому дуже важливо було створити усі умови, щоб користувач почував себе комфортно на цій сторінці. Для відображення даних на цій та подібних цій сторінці використовували таблиці, для яких було додано велика кількість елементів для зручності, серед них пагінація, зміна кількості елементів на сторінці, сортування за колонками, зміна розмірів колонки, приховування колонок, зміна висоти рядка, пошук (в даному випадку по імені та email-адресі), кнопка авторозміру сторінки та розширені фільтри. Цих елементів

куди більш достатньо аби забезпечити зручне користування таблицею із великою кількістю даних.

HRM

DM

Наші робітники

ДОДАТИ ПРАЦІВНИКА

Пошук

СТОВПЦІ

ВИСОТА РЯДКА

РОЗШИРЕНІ ФІЛЬТРИ

ІЗ ДЕАКТИВОВАНИМИ

СКИНУТИ ФІЛЬТРИ

Ім'я	Пошта	Посада	Офіс	Відділ	Статус	Тип працевлаштування	Дії
User115	user115@example.com	JobTitle115	Office2	Department5	Працює	Не повний робочий день (0.32год.)	
User116	user116@example.com	JobTitle116	Office5	Department1	Працює	Не повний робочий день (5.8год.)	
User117	user117@example.com	JobTitle117	Office1	Department8	Працює	Не повний робочий день (5.4год.)	
User118	user118@example.com	JobTitle118	Office1	Department4	Працює	Не повний робочий день (7.36год.)	
User119	user119@example.com	JobTitle119	Office1	Department7	Працює	Не повний робочий день (2.08год.)	
User12	user12@example.com	JobTitle12	Office4	Department2	Працює	Не повний робочий день (3.92год.)	
User120	user120@example.com	JobTitle120	Zhytomyr	Department7	Працює	Не повний робочий день (4.64год.)	
User121	user121@example.com	JobTitle121	Office2	Department1	Працює	Не повний робочий день (6.72год.)	
User122	user122@example.com	JobTitle122	Office2	Department7	Працює	Не повний робочий день (0.72год.)	
User123	user123@example.com	JobTitle123	Office3	Department2	Працює	Не повний робочий день (3.44год.)	
User124	user124@example.com	JobTitle124	Zhytomyr	Department7	Працює	Не повний робочий день (5.6год.)	

Рядів на сторінці: 25 26-50 з 308 1 2 3 4 5 ... 13 > АВТОРОЗМІР

Рисунок 3.9 – Сторінка з переліком робітників у вигляді таблиці

Розширені фільтри

Офіс

Без офісу

Відділ

Без відділу

Тип працевлаштування

Нічого

Звітує до

Статус користувача

☐ Тільки менеджери

ОЧИСТИТИ ФІЛЬТРИ

ДОДАТИ

Рисунок 3.10 – Розширені фільтри для сторінки з переліком робітників

Для додавання нового користувача в систему теж є окрема сторінка, яка містить форму для введення усіх полів із відповідною валідацією та іншими елементами. Наприклад, якщо форму заповнює користувач із усіма дозволами, то на форму будуть присутні додаткові поля для встановлення чи редагування дозволів користувача що створюється чи редагується.

Рисунок 3. 11 – Сторінка для створення нового користувача або його редагування

Розглянемо функціонал відпусток, яку можна грубо поділити на звичайну сторінку та адміністративну. Сторінки виконані з використанням компонента таблиці, аналогічного до того, який знаходиться на сторінці користувачів. На сторінці відпусток користувач має змогу побачити власні запити на відпустку та їх статус, а також існує можливість відмінити запит поки менеджер не погодив відповідь на нього. На сторінці також вказується кількість часу відпустки, яку вдалося накопичити протягом усього часу, а також кількість зарезервованих днів, у випадку якщо є погоджені відпустки, час яких ще не настав.

Початок	Кінець	Коментар	Затверджувач	Коментар затверджувача	Статус	Дія
20 вересня 2024 р.	25 вересня 2024 р.		Давід Шеєдин davyd.melnichenko@gmail.com	Pending	⌘	⋮
10 червня 2024 р.	15 червня 2024 р.	Taking a break to recharge.	Davyd Melnychenko ipz204_mdv@student.znu.edu.ua	I'm asking :)	✓	⋮

Рисунок 3.12 – Сторінка відпусток

Сторінка створення запиту на відпустку має аналогічний елемент інформації про зарезервований і накопичений час. А також додаткові поля валідація запиту, наприклад форма буде відображати попередження якщо

користувач створює запит на відпустку, де кількість вказаних днів перевищує кількість накопичених днів.

Створити запит на відпустку

Інформація про відпустку

Набраний час відпустки: декілька секунд

Із них вже заброньовано: 12 днів

Початок *

12.06.2024

—

Кінець *

02.07.2024

Коментар

Запит на відпустку має бути створений хоча б за два тижні

Запит перевищує набрану кількість часу для відпустки

ВІДМІНИТИ

СТВОРИТИ

Рисунок 3.13 – Форма створення відпустки

Для затвердження відпусток існує окрема сторінка, доступ до якої мають лише користувачі із відповідним дозволом, сторінка також представлена у вигляді таблиці.

Для лікарняних була використана аналогічна схема з компонентом-таблицею із відмінністю у тому, що присутня лише одна сторінка, де користувачі без дозволу можуть лише переглядати інформацію, а з дозволом - редагувати, створювати та видаляти.

У випадку із святами відмінність полягає у наявності форми для створення та редагування на самій сторінці поряд з таблицею. У випадках коли форма і таблиця не будуть вміщатися на сторінку, форма буде відображена під сторінкою, а для покращення навігації додаються додаткові компоненти, які надають можливість автоматично переміщуватися між таблицею та формою. Ці зміни зумовлені тим, що користувачі найвірогідніше будуть додавати свята

одразу великими кількостями, а наявність додаткової сторінки для створення свята лише буде додавати зайвих кроків.

The screenshot shows the 'Створити вихідний' (Create Holiday) form in the HRM system. The interface includes a red top header with a menu icon, a logo, the text 'HRM', and a 'DM' button. A left sidebar contains icons for calendar, user, search, add, list, calendar, folder, and settings. The main content area displays the form with the following fields: 'Назва *' (Name), 'Дата *' (Date) with a calendar icon, 'Кінець (якщо > 24год)' (End (if > 24h)) with a calendar icon, 'Тип вихідного' (Holiday type) dropdown menu with 'Свято' (Holiday) selected, and a checkbox 'Сповістити робітників' (Notify workers). At the bottom, there are buttons '↑', 'ВІДМІНИТИ' (Cancel), 'ПІДТВЕРДИТИ' (Confirm), and a 'ТРЕКЕР' (Tracker) button with an upward arrow. A pagination bar at the top of the form shows 'Рядків на сторінці: 10', '1-10 з 21', and page numbers 1, 2, 3, with a red circle around the number 1. A button 'АВТОРОЗМІР' (Auto-size) is also present.

Рисунок 3.14 – Фрагмент сторінки для керування вихідними.

Сторінка профілю містить дані про користувача у вигляді карток, поділених на групи для зручності. Меню керування, до складу якого входить наприклад редагування чи деактивація працівника. А також графік, який демонструє початок і кінець робочих сесій користувача протягом місяця і приблизний прогнозований час коли його найвірогідніше можна зустріти на роботі.

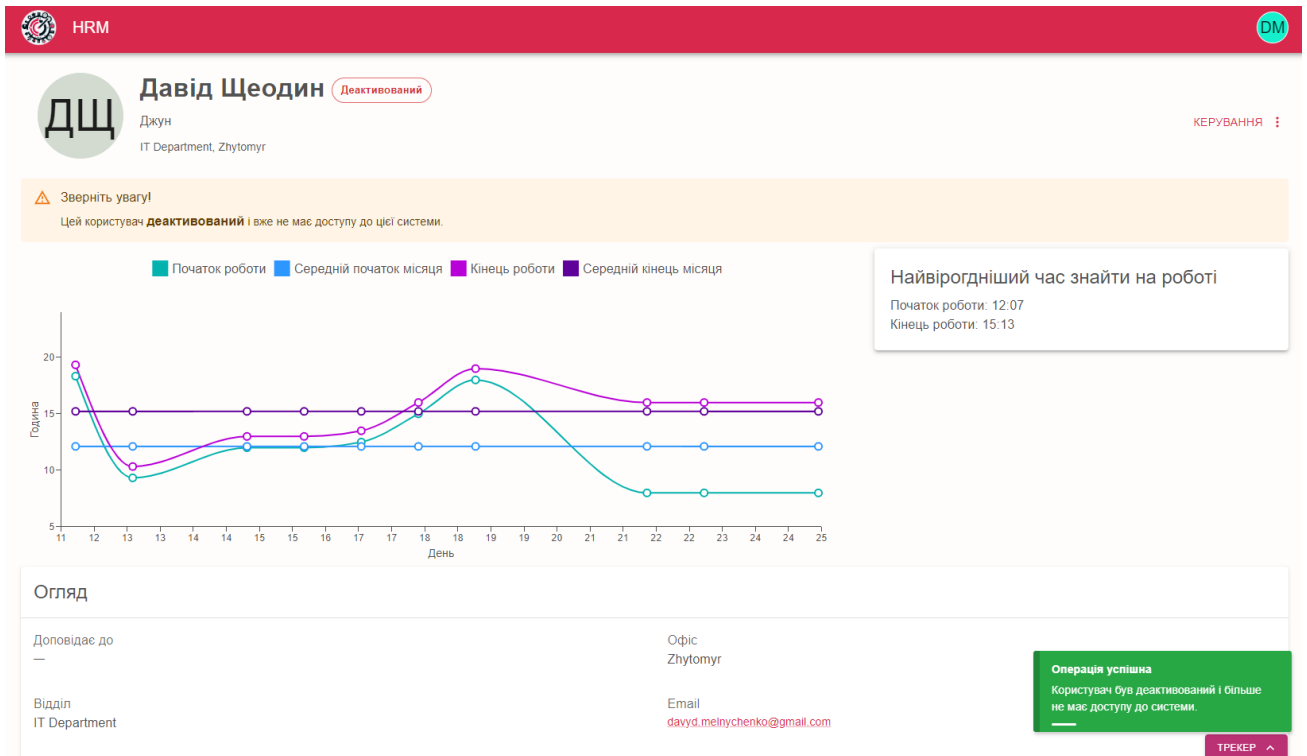


Рисунок 3.15 – Сторінка профілю працівника

Наступною розглянемо сторінку статистики, яка збирає статистичну інформацію про відпрацьовані години для працівників. Сторінка теж використовує компонент таблиці і деякі додаткові елементи, наприклад вибір місяця, загальна інформація про кількість робочих днів у місяці та вихідні в ньому. Зверху присутня кнопка завантаження звіту в вигляді ексел-файлу.

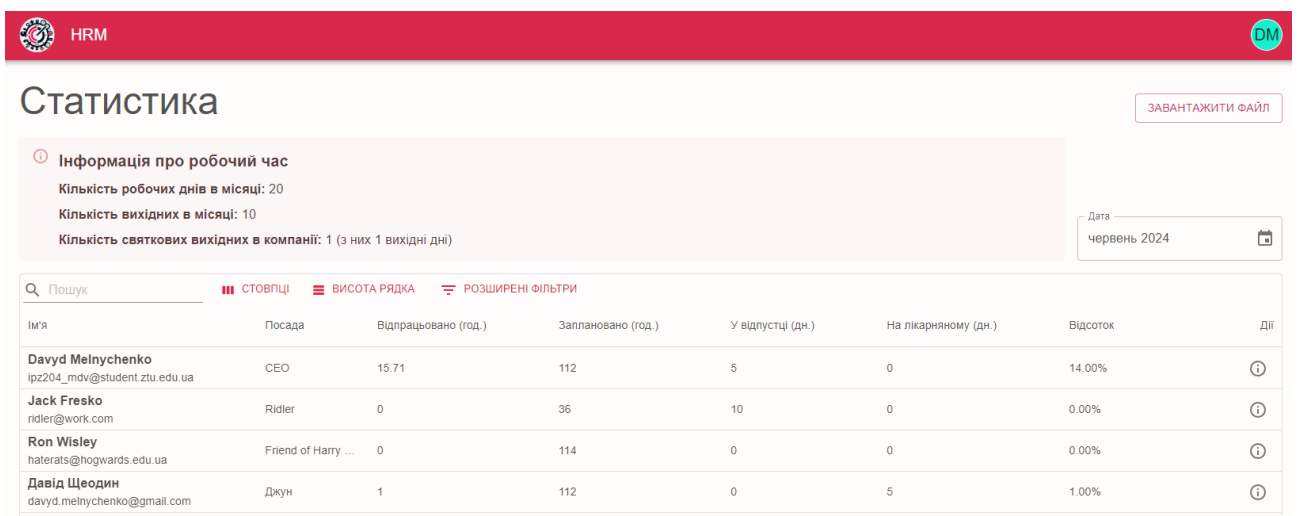


Рисунок 3.16 – Сторінка статистики

По кожному користувачу також можна переглянути детальну інформацію, яка подається у вигляді графіків та детального опису проведених обрахунків.

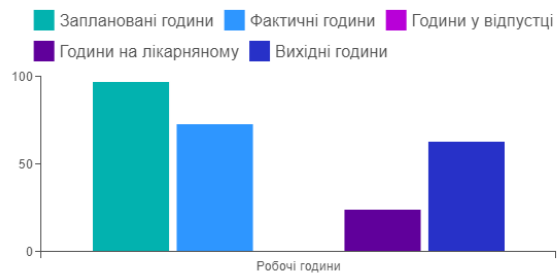
Деталі



Інформація про відпрацьовані години користувача:



Davyd Melnychenko
CEO
lpz204_mdv@student.ztu.edu.ua



Кількість робочих днів в місяці: 23
Робоча ставка: 8 год.
Заплановані години: 23 дн. x 8 год. = 184 год.
Дні у відпустці: 0 дн. (184 год. - 0 дн. x 8 = 184 год.)
Дні на лікарняному: 3 дн. (184 год. - 3 дн. x 8 = 160 год.)

ЗАКРИТИ

Рисунок 3.17 – Детальна статистика для одного користувача

Якщо користувач бажає отримати інформацію про робочий час, редагувати його чи звірити її із кимось іншим, він може скористатися сторінкою планувальника. Планувальник представляє собою календар, на якому відображають проведені робочі сесії, свята, відпустки та лікарняні. Календар підтримує перегляд окремого дня, тижня чи місяця, а також режим порядку денного (Agenda).

Для зручності календар також підтримує зображення інших користувачів на календарі, що дає змогу порівнювати їх робочі сесії. Також користувачів можна відображувати на різному календарі, використовуючи розділений режим. При потребі користувач може сховати заплановані робочі сесії.

Для додавання інших користувачів, можна скористатися бокової панеллю, яка відображається над календарем на менших екранах.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

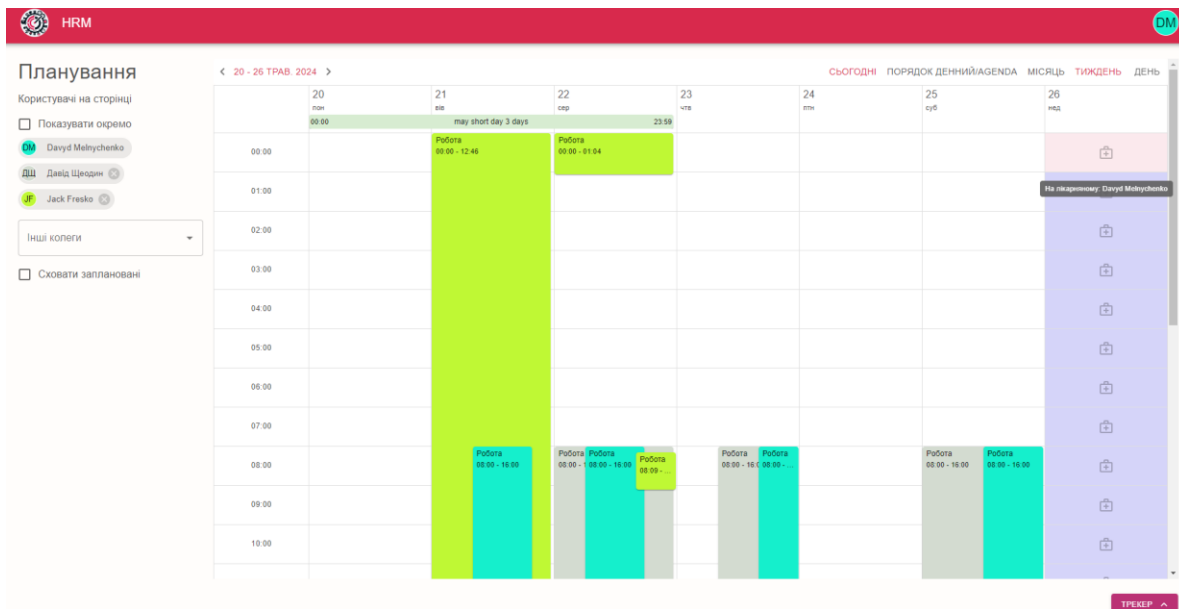


Рисунок 3.18 – Сторінка планувальника в режимі перегляду тижня

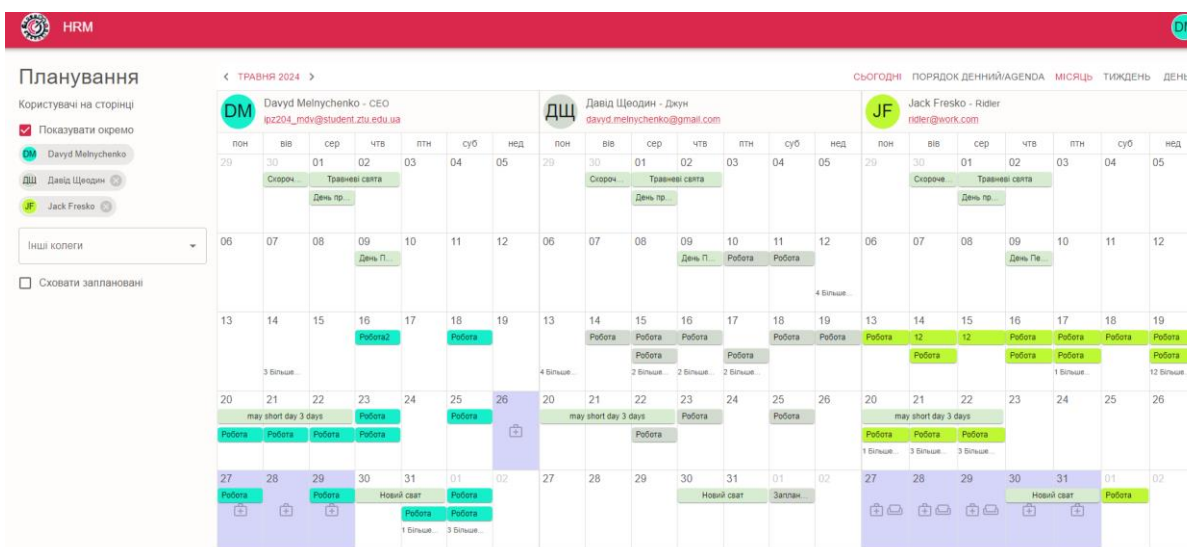


Рисунок 3.19 – Сторінка планувальника в режимі перегляду місяця

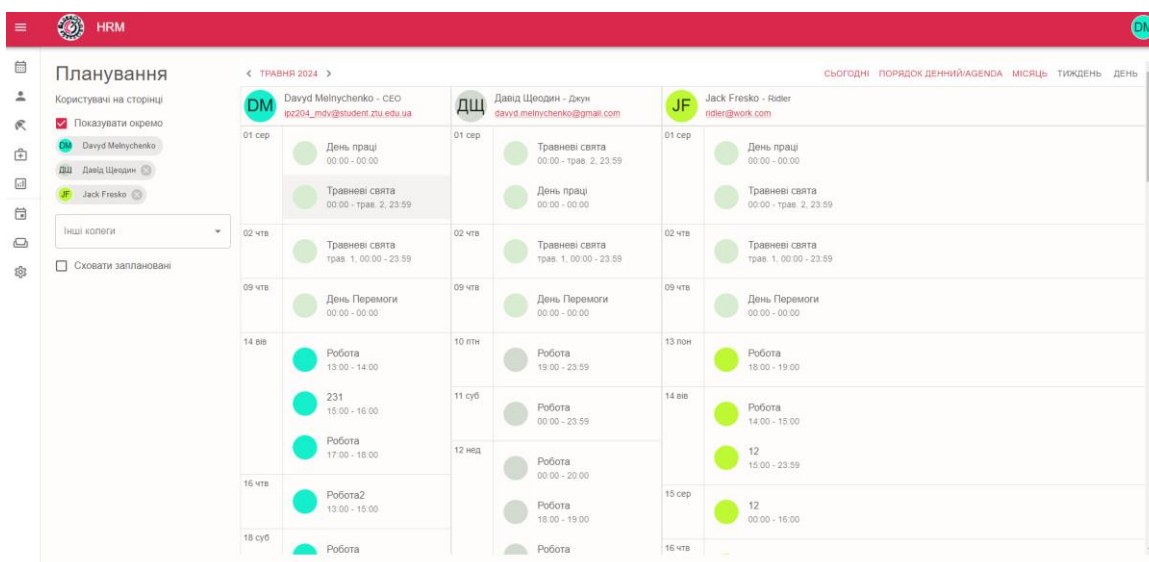


Рисунок 3.20 – Сторінка планувальника в режимі порядку денного

3.3. Тестування роботи програмного комплексу

Для забезпечення якості програмного продукту дуже важливо провести його тестування. Тестування дозволяє збільшити шанс знайти помилки на ранніх етапах розробки, що зможе надати користувачам більш приємний досвід користування вебзастосунком.

Вебзастосунок для керування робочим часом також не є виключенням, тому під час роботи було проведено багато мануального тестування. Воно є основним способом перевірки працездатності системи, адже дозволяє на власні очі побачити результат роботи програми.

Основними перевагами мануального тестування є його гнучкість, швидкість та простота. Вони майже завжди допоможуть там де інші види тестування не справляються. Крім того воно не вимагає наявності будь-якого додаткового програмного забезпечення.

Під час виконання мануального тестування було знайдено величезну кількість помилок, наприклад:

- відображення даних трекера старого користувача після повторної авторизації через інший акаунт користувача;
- активація обробника подій до того як завершилася початкове завантаження, що відображало сторінку до того як усі потрібні дані були встановлені в стан;
- некоректне спрацьовування валідації полів форми;
- алгоритм підраховування відпрацьованих годин віднімав час, який вже був віднятий коли були відкинуті усі вихідні (субота та неділя) в місяці;
- планувальник не правильно визначав потрібну дату через що запит до серверу надсилався декілька разів.

Цей список ще можна продовжувати, адже мануальне тестування допомогло вирішити значно більше помилок ніж можна описати. Але все ж мануальне тестування не дає в повній змозі оцінити деякі окремі елементи системи, наприклад продуктивність чи швидкість завантаження.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

Також для тестування окремих елементів програмного продукту було застосовано unit тести. Вони запускаються автоматично та ізольовано від усієї системи. Завдяки ним розробники можуть бути певні, що частина коду працює правильно, і якщо в майбутньому хтось порушить роботу механізму, то unit тести дадуть йому про це знати.

У власній розробці unit тести були використані для перевірки роботи класу для обрахунку відпрацьованих годин. На даний момент тести перевіряють декілька сценаріїв, серед яких:

- валідацію введеного періоду часу;
- перевірка коректного спрацьовування методу обрахування із використанням відпусток, лікарняних та вихідних;
- перевірка коректного спрацьовування методу при використанні користувача із не повною ставкою і скорочених днів.

Для прикладу розглянемо один із написаних тестів (повний лістинг класу наведено в додатках).

```
[Fact]
public async Task CalculateHoursSummaryPageAsync_ReturnsCorrectHoursSummaryPage()
{
    var userPageInput = new UserPageInput();
    var from = new DateTime(2024, 6, 1);
    var to = new DateTime(2024, 6, 30);

    var users = new List<User> { new() { Id = Guid.NewGuid(), EmploymentRate = 100 } };
    userRepository.Setup(r => r.GetPageAsync(userPageInput)).ReturnsAsync(new
    Page<User>(1, users));

    var holidays = new List<Holiday> { new() { Date = new DateTime(2024, 6, 12), Type
    = HolidayType.DayOff } };
    holidayRepository.Setup(r => r.GetByDateRangeAsync(from,
    to)).ReturnsAsync(holidays);

    var vacations = new List<VacationRequest> { new() { UserId = users[0].Id, Start =
    new DateTime(2024, 6, 2), End = new DateTime(2024, 6, 3) } };
    vacationRepository.Setup(r =>
    r.GetUsersVacationsInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from,
    to)).ReturnsAsync(vacations);

    var sickLeaves = new List<SickLeave> { new() { UserId = users[0].Id, Start = new
    DateTime(2024, 6, 4), End = new DateTime(2024, 6, 5) } };
    sickLeaveRepository.Setup(r =>
    r.GetSickLeaveInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from,
    to)).ReturnsAsync(sickLeaves);

    var workSessions = new List<WorkSession> { new() { UserId = users[0].Id, Start =
    new DateTime(2024, 6, 6, 9, 0, 0), End = new DateTime(2024, 6, 6, 17, 0, 0) } };
    workSessionRepository.Setup(r => r.GetInRangeAsync(It.IsAny<IEnumerable<Guid>>(),
    from, to, true)).ReturnsAsync(workSessions);
```

```

var result = await target.CalculateHoursSummaryPageAsync(userPageInput, from, to);

Assert.NotNull(result);
Assert.Equal(1, result.Page.TotalAmount);
Assert.Equal(20, result.WorkingDays);
Assert.Equal(1, result.Holidays);
Assert.Equal(1, result.DayOffs);

var summary = result.Page.Items.First();
Assert.Equal(users[0], summary.User);
Assert.Equal(128, summary.ExpectedHours);
Assert.Equal(8, summary.ActualHours);
Assert.Equal(0.062, summary.Percent, 3);
Assert.Equal(1, summary.DaysInVacation);
Assert.Equal(2, summary.DaysInSickLeave);
}

```

Опис роботи цього алгоритму вже було наведено в розділі 2.4., тому не будемо зупинятися на його повторному поясненні. Звернемо увагу на те як unit-тест готує дані *mocked* дані для відтворення основного методу, в чому йому допомагає NuGet-пакет Moq. Для написання самих unit-тестів використовували інший пакет під назвою XUnit.

Крім того тестування продукту також здійснювалося за допомогою такого інструменту як Lighthouse. Lighthouse — це автоматизований інструмент із відкритим кодом для покращення якості веб-сторінок. Доступ до нього може отримати будь-хто просто використовуючи панель розробника.

Цей інструмент має низку метрик якості вебсторінки, такі як продуктивність, доступність, прогресивність, SEO тощо. Після проведення аналізу сторінки Lighthouse генерує звіт про ефективність сторінки. Звіт у дуже зручному форматі показує ваші показники, а також корисні поради для покращення сторінки. Кожен аудит має довідковий документ, у якому пояснюється, чому аудит важливий, а також як це можна виправити.

Тож запусимо Lighthouse на сторінках нашої системи:

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

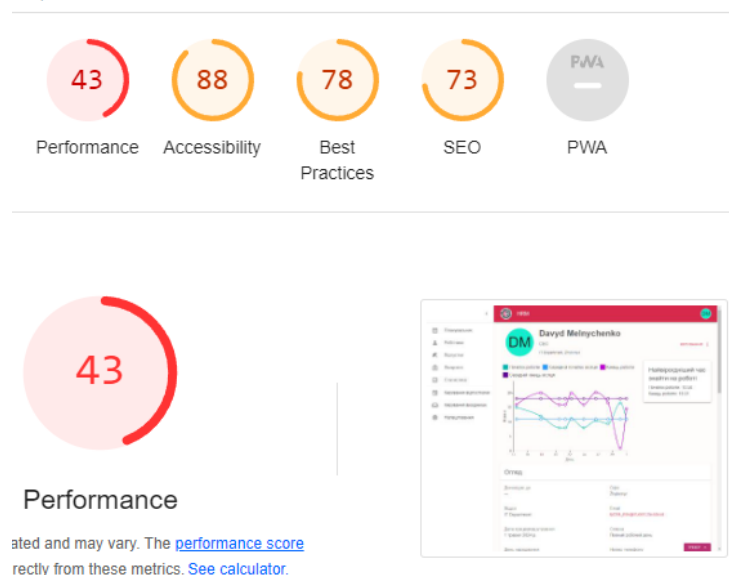


Рисунок 3.21 – Запуск звіт Lighthouse на сторінці профілю

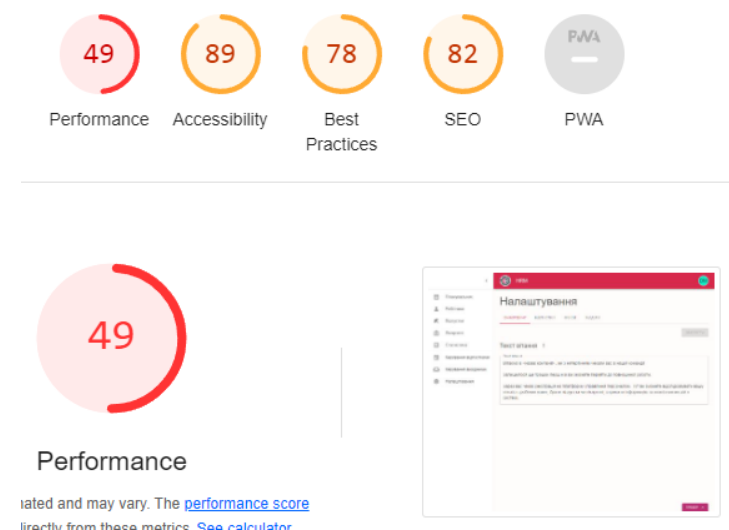


Рисунок 3.22 – Запуск звіт Lighthouse на сторінці налаштувань

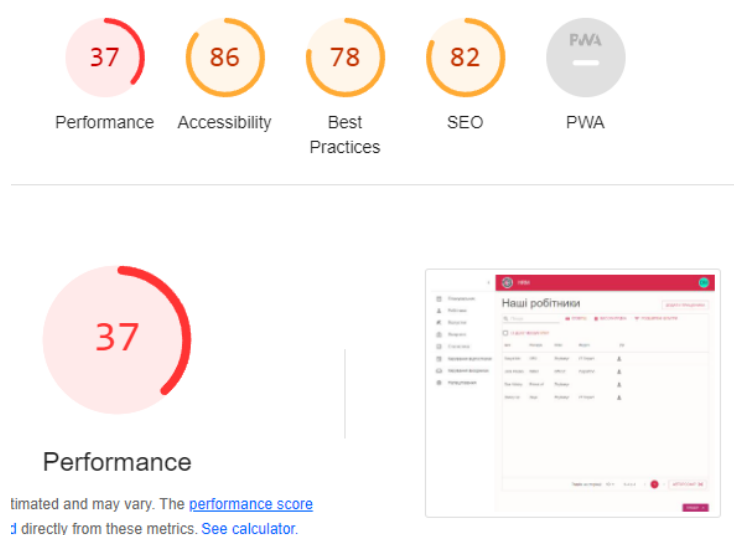


Рисунок 3.23 – Запуск звіт Lighthouse на користувачів

Як ми можемо побачити, більшість показників Lighthouse мають хороший рівень якості, до таких входять доступність, найкращі практики, SEO. Але основна увагу варто звернути на продуктивність сторінки, яка, нажаль, тримається на доволі низькому рівні. Це означає що код на сторінці повільніший ніж міг би бути. І хоч на даному етапі цього повністю вистачає аби забезпечити усі поставлені задачі, все ж це дуже важливий елемент якості веб сторінки, який варто покращити як тільки буде така можливість.

У своєму звіті Lighthouse надає наступні рекомендації:

- уникати великих зсувів макета;
- застосувати стиснення тексту;
- мінімізувати JavaScript;
- якщо є, зменшити присутню кількість коду JavaScript;
- видалити повторювальні методи в пакетах JavaScript, якщо такі є.

Отже, засоби мануального та unit-тестування допомогли нам забезпечити кращу якість програмного застосунку, що зможе значно покращити досвід користування вебзастосунком. А інструмент Lighthouse допоміг отримати чіткий звіт із метриками, які оцінюють стан власної розробки, а також список рекомендацій для покращення оцінки вебсайту.

Висновки до розділу 3

У ході виконання третього розділу дипломної роботи було описано порядок встановлення та налаштування параметрів системи, включаючи такі його елементи як SMTP сервер, сервер бази даних, сервер вебзастосунку та пристрій користувача. Для кращого розуміння було наведено діаграму розгортання продукту.

Було продемонстровано структуру інтерфейсу веб-орієнтованої системи керування робочим часом. Було зображено основні елементи інтерфейсу програми та описано багато прийнятих рішень стосовно створення дизайну вебінтерфейсу застосунку, включаючи діаграми, як елементу графічного відображення даних.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

Останнім етапом стало тестування роботи програмного комплексу в ході якого було використано мануальне тестування вебзастосунку, покриття коду unit-тестами. Для отримання метрик стосовно продуктивності вебсайту було використано інструмент Lighthouse, який вказав на недоліки, а також надав рекомендації для покращення оцінки вебсайту.

В результаті виконання розділу було створено і описано інтерфейс та порядок роботи з веб-орієнтованою системою керування робочим часом з елементами системи управління персоналом.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

ВИСНОВКИ

В ході виконання дипломної роботи було проведено детальний аналіз напрямків застосування інформаційних технологій для реалізації вебзастосунок для керування часом з елементами системи управління персоналом. Було поставлені задачі до продукту, проведено детальний аналіз аналогічних розробок. Також було спроектовано і обрано архітектурні шаблони вебзастосунок, які змогли б в подальшому розширити цей застосунок до більших масштабів. Було проведено дослідження з можливих технологій для вебзастосунок, та обрано технології та інструменти, які змогли б досягти бажаного результату.

Важливим етапом також було визначення варіантів використання системи та створення об'єктно-орієнтованої структури системи. Формування стабільної і зручної бази даних також відіграло важливу роль, так як платформа орієнтована на зберігання великих обсягів інформації. Попередні кроки дозволили спроектувати основні алгоритми роботи системи, а також реалізувати їх враховуючи усі сформовані потреби з використанням вибраних архітектурних шаблонів та технологій.

Після реалізації основних функцій системи, було створено механізм розгортання вебзастосунок, враховуючи усі його компоненти. Також було проведено важливу роботу із проектування інтерфейсів і дизайну вебзастосунок. Наостанок було проведено тестування програмного застосунок засобами мануального та unit-тестування, а також отримано оцінку системи засобами такого інструменту як Lighthouse.

Результатом виконання дипломної роботи є розроблений і готовий до використання вебзастосунок для управління робочим часом з елементами системи управління персоналом, який забезпечує автоматизацію, оптимізацію та спрощення процесів обліку та планування робочого часу.

Було досягнуто усіх поставлених завдань включаючи створення масштабованої, потужної і надійної архітектури, яка дозволить проекту і надалі

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

еволюціонувати у напрямку вебзастосунку для управління людським персоналом.

Розроблений вебзастосунок може бути впроваджений у діяльність підприємств різних галузей для автоматизації обліку та планування робочого часу. Це дозволить підвищити ефективність управління робочими процесами, знизити витрати часу на адміністративні процедури та покращити продуктивність працівників. Система також забезпечує точність даних, що сприяє оптимізації внутрішніх процесів компаній та створенню основи для подальшого розвитку повноцінної системи управління персоналом. Вебзастосунок може забезпечити централізоване місце для усієї інформації про робочий час, що полегшує доступ до цієї інформації та її аналіз.

Хоч вебзастосунок вже має багато корисних функцій, присутній величезний простір для подальшого розвитку. Наприклад, можете розглянути можливість додавання функції автоматичного відстеження часу, яка би допомогла користувачам точніше відстежувати свій робочий час. Також можна розглянути інтеграцію з іншими системами, такими як системи управління проектами або календарі, щоб надати користувачам більше контексту про свою роботу. Крім того, можете розглянути можливість додавання аналітичних інструментів, які б допомогли користувачам краще розуміти, як вони витрачають свій час. Також можна розвивати налаштування системи для додавання гнучкості її елементам, а також додавати функціонал що має належить до системи управління персоналом.

Найбільшими складнощами під час створення вебзастосунку було створення потужної архітектури за допомогою складних інструментів, що були обрані для реалізації. Також велику кількість зусиль було витрачено на створення алгоритмів роботи системи.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Client-Server Architecture? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.simplilearn.com/what-is-client-server-architecture-article>.
2. What is a human resource information system (HRIS)? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.hibob.com/hr-glossary/hris/>.
3. Функції та завдання HRM системи управління персоналом [Електронний ресурс] – Режим доступу до ресурсу: <https://peopleforce.io/uk/blog/shho-take-hrm-i-navishho-potribna-optimizatsiya-hr-protseviv>.
4. HURMA [Електронний ресурс] – Режим доступу до ресурсу: <https://hurma.work/>.
5. Drawio. Step-by-Step Guides [Електронний ресурс] – Режим доступу до ресурсу: <https://drawio-app.com/tutorials/step-by-step-guides/>.
6. MDN Web Docs. Resources for Developers, by Developers [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/>.
7. Microservices vs. monolithic architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>.
8. Рефакторинг.Гуру [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk>.
9. N-tier architecture style [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>.
10. ASP.NET documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0>.
11. Policy-based authorization in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/policies?view=aspnetcore-8.0>.
12. Unit testing C# in .NET using dotnet test and xUnit [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test>.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

13. SQL Sever. User-defined functions [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-functions?view=sql-server-ver16>.
14. Fluent migrations framework for .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://fluentmigrator.github.io/>.
15. Learn Dapper [Електронний ресурс] – Режим доступу до ресурсу: <https://www.learndapper.com/>.
16. GraphQL .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql-dotnet.github.io/>.
17. GraphQL [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql.org/>.
18. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>.
19. Vite Next Generation Frontend Tooling [Електронний ресурс] – Режим доступу до ресурсу: <https://vitejs.dev/>.
20. Flux In-Depth Overview [Електронний ресурс] – Режим доступу до ресурсу: <https://facebookarchive.github.io/flux/docs/in-depth-overview/>.
21. Re-ducks. Building on the duck legacy [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/alexn/re-ducks>.
22. What is Component-Based Architecture? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mendix.com/blog/what-is-component-based-architecture/>.
23. Реактивне програмування: що це таке та його особливості [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/reaktyvne-prohramuvannia/>.
24. RxJS Reactive Extensions Library for JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://rxjs.dev/>.
25. redux-observable [Електронний ресурс] – Режим доступу до ресурсу: <https://redux-observable.js.org/>.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

26. Redux A JS library for predictable and maintainable global state management [Електронний ресурс] – Режим доступу до ресурсу: <https://redux.js.org/>.
27. Thinking in React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn/thinking-in-react>.
28. Material UI, an open-source React component library [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/material-ui/>.
29. Mailtrap. Sending Emails in ASP.NET Core [Електронний ресурс] / Mailtrap – Режим доступу до ресурсу: <https://mailtrap.io/blog/asp-net-core-send-email/>.
30. .Net, Asynchronous programming with async and await [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/>.
31. Мельниченко Д.В., Вакалюк Т.А. Вебзастосунок для керування робочим часом: огляд аналогів. Матеріали Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених «Тенденції розвитку технологій в автоматизації, приладобудуванні та робототехніці» (присвячена Дню науки), 13 травня 2024 року.

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

ДОДАТКИ

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

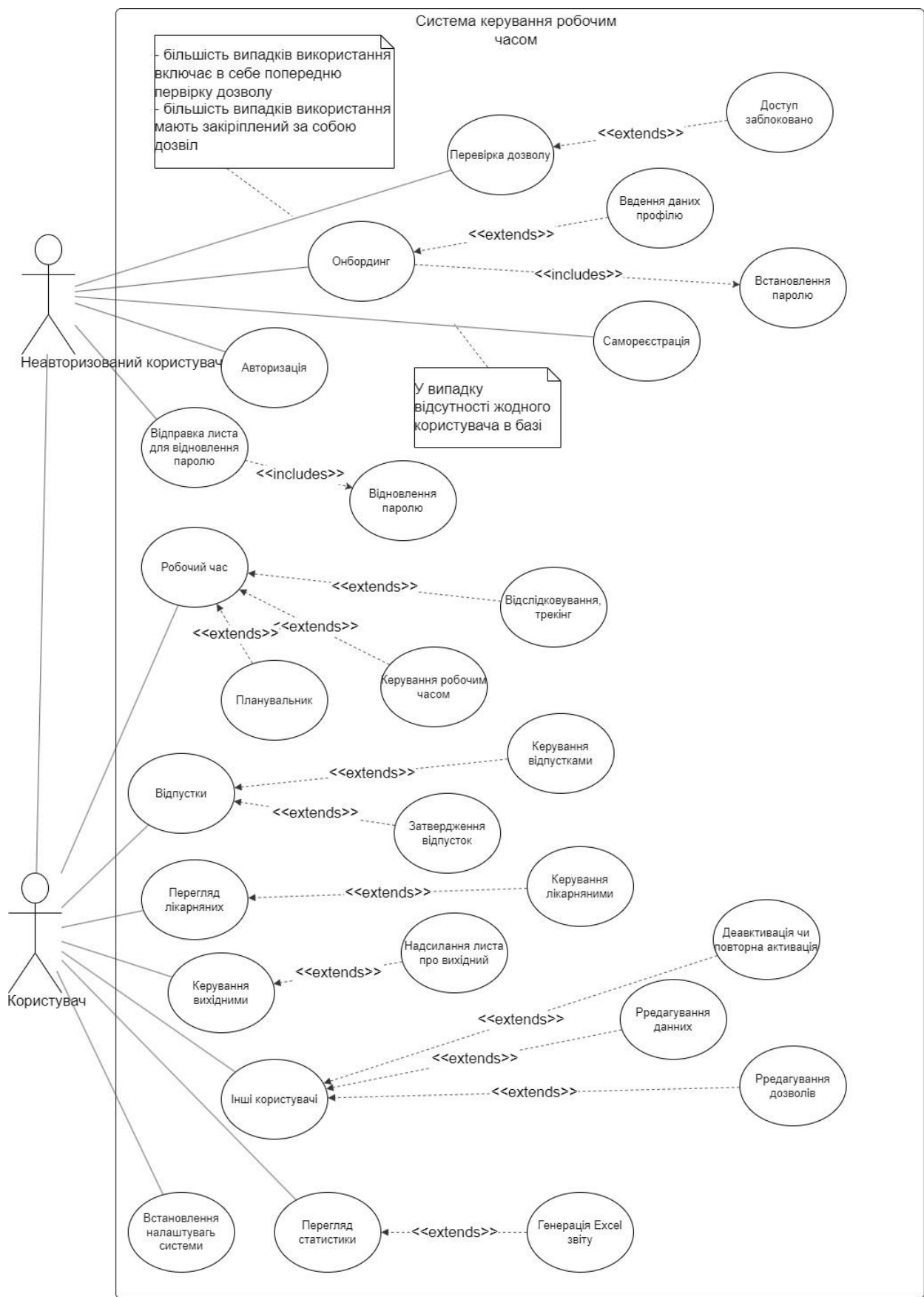


Рисунок А.1 – Діаграма варіантів використання

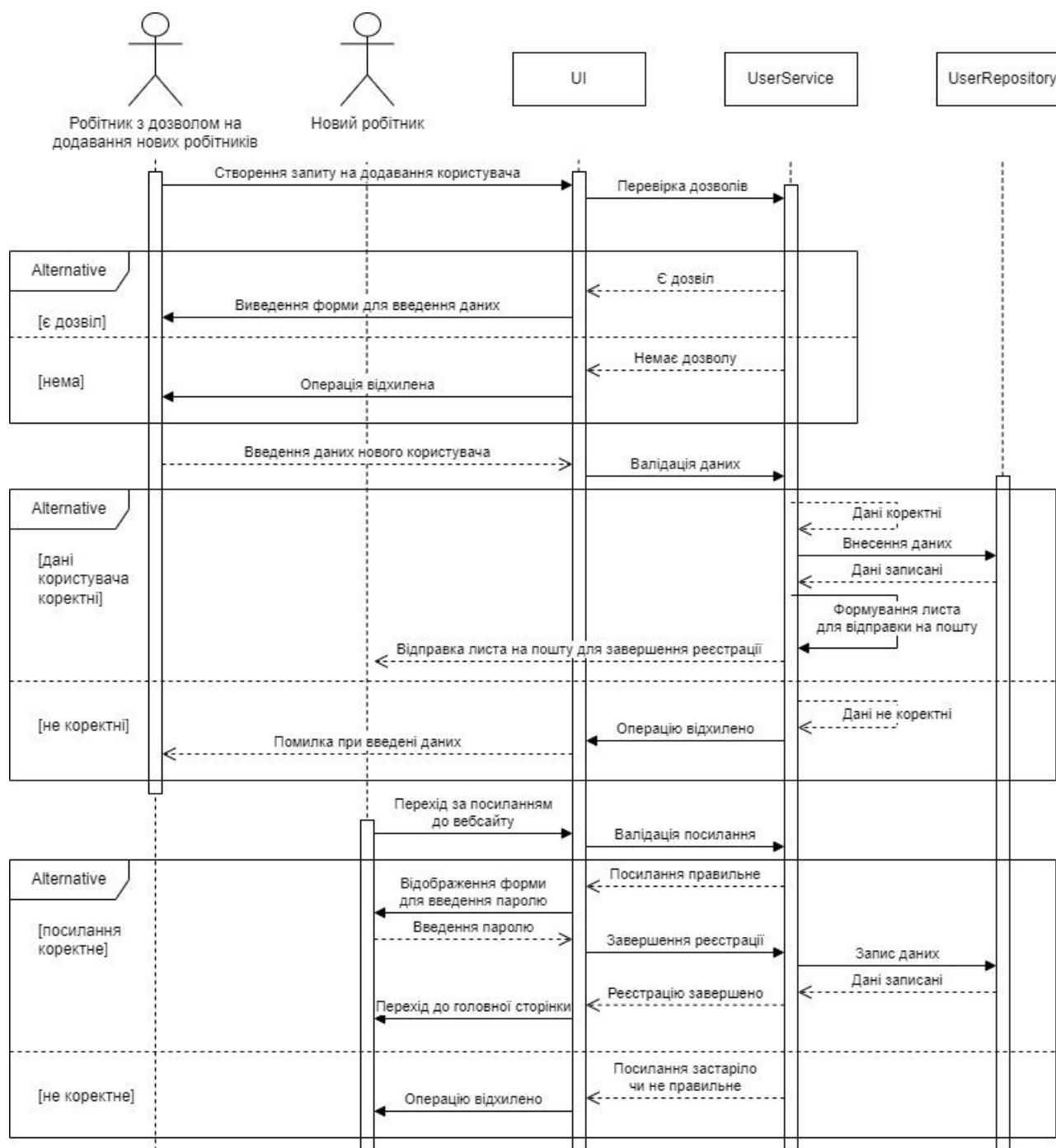


Рисунок А.2 – Діаграма послідовності для додавання нового користувача в систему

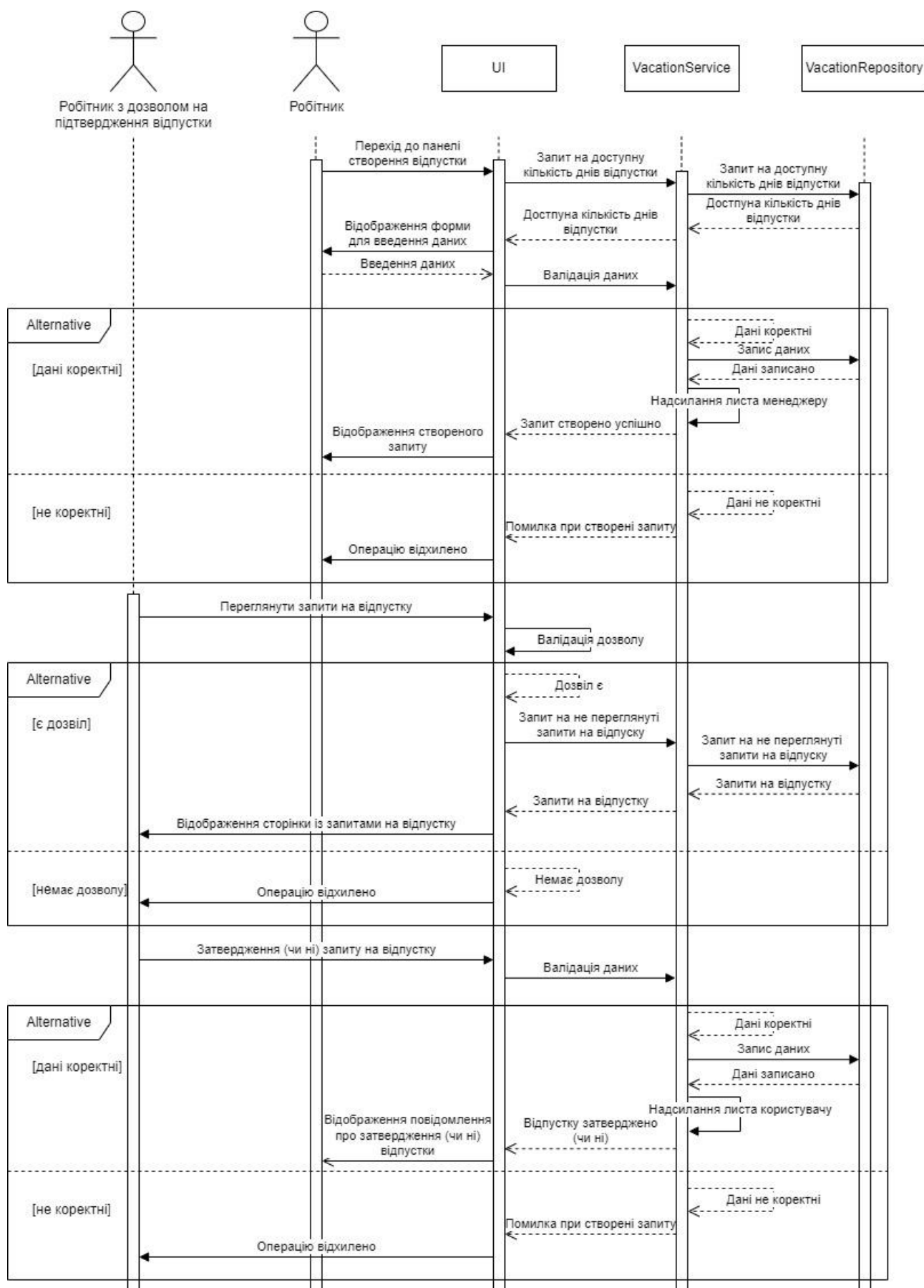


Рисунок А.3 – Діаграма послідовності для створення запиту на відпустку та його затвердження.

Клас HoursSummaryCalculator:

```

public class HoursSummaryCalculator(
    UserRepository userRepository,
    HolidayRepository holidayRepository,
    WorkSessionRepository workSessionRepository,
    VacationRequestRepository vacationRepository,
    SickLeaveRepository sickLeaveRepository,
    IConfiguration configuration)
{
    readonly UserRepository userRepository = userRepository;
    readonly HolidayRepository holidayRepository = holidayRepository;
    readonly WorkSessionRepository workSessionRepository = workSessionRepository;
    readonly VacationRequestRepository vacationRepository = vacationRepository;
    readonly SickLeaveRepository sickLeaveRepository = sickLeaveRepository;

    readonly int FullHoursPerDay = configuration.GetValue("WorkHours:FullDay", 8);

    public async Task<HoursSummaryPage> CalculateHoursSummaryPageAsync(UserPageInput
    userPageInput, DateTime from, DateTime to)
    {
        var userPage = await userRepository.GetPageAsync(userPageInput);

        var hoursSummaryResult = await CalculateHoursSummaryAsync(userPage.Items,
        from, to);

        var page = new Page<IndividualHoursSummary>(userPage.TotalAmount,
        hoursSummaryResult.IndividualHoursSummaries);

        return new(hoursSummaryResult.WorkingDays, hoursSummaryResult.Holidays,
        hoursSummaryResult.DayOffs, page);
    }

    public async Task<HoursSummary> CalculateHoursSummaryAsync(DateTime from, DateTime
    to)
    {
        var users = await userRepository.GetAllActiveAsync();

        var hoursSummaryResult = await CalculateHoursSummaryAsync(users, from, to);

        return new(
            hoursSummaryResult.WorkingDays,
            hoursSummaryResult.Holidays,
            hoursSummaryResult.DayOffs,
            hoursSummaryResult.IndividualHoursSummaries);
    }

    async Task<HoursSummaryResult> CalculateHoursSummaryAsync(IEnumerable<User> users,
    DateTime from, DateTime to)
    {
        var generalWorkingDays = GetWorkingDays(from, to);

        var holidays = await holidayRepository.GetByDateRangeAsync(from, to);
        var dayOffDays = GetDayOffDays(holidays, from, to);
        var shortDays = GetShortDays(holidays, from, to);

        var userIds = users.Select(u => u.Id);
        var vacations = await
        vacationRepository.GetUsersVacationsInRangeAsync(userIds, from, to);
        var sickLeaves = await sickLeaveRepository.GetSickLeaveInRangeAsync(userIds,
        from, to);
        var workSession = await workSessionRepository.GetInRangeAsync(userIds, from,
        to);
    }
}

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

var individualHoursSummaries = users.Select(u =>
{
    var hoursPerDay = FullHoursPerDay * (u.EmploymentRate / 100.0);
    var totalHoursToWork = generalWorkingDays * hoursPerDay;

    totalHoursToWork -= dayOffDays * hoursPerDay;
    totalHoursToWork -= GetShortDaysHours(holidays, from, to, hoursPerDay);

    var vacationDays = GetVacationDays(vacations, u.Id, from, to);
    totalHoursToWork -= vacationDays * hoursPerDay;

    var sickLeaveDays = GetSickLeaveDays(sickLeaves, u.Id, from, to);
    totalHoursToWork -= sickLeaveDays * hoursPerDay;

    var actualHours = GetActualHours(workSession, u.Id, from, to);
    var percent = actualHours / totalHoursToWork;

    return new IndividualHoursSummary(u, totalHoursToWork, actualHours,
percent, vacationDays, sickLeaveDays);
});

var generalHolidaysDays = dayOffDays + shortDays;

return new(generalWorkingDays, generalHolidaysDays, dayOffDays,
individualHoursSummaries);
}

static int GetDayOffDays(IEnumerable<Holiday> holidays, DateTime from, DateTime
to) => holidays
    .Where(h => h.Type is not HolidayType.ShortDay)
    .Select(h => CountDays(h, from, to, h => h.Date, h => h.EndDate ?? h.Date))
    .Sum();

static int GetShortDays(IEnumerable<Holiday> holidays, DateTime from, DateTime to)
=> holidays
    .Where(h => h.Type is HolidayType.ShortDay)
    .Select(h => CountDays(h, from, to, h => h.Date, h => h.EndDate ?? h.Date))
    .Sum();

double GetShortDaysHours(IEnumerable<Holiday> holidays, DateTime from, DateTime
to, double hoursPerDay) => holidays
    .Where(h => h.Type is HolidayType.ShortDay)
    .Select(h =>
    {
        var days = CountDays(h, from, to, h => h.Date, h => h.EndDate ?? h.Date);
        var shortenedHoursPerDay = h.HoursToWork ?? FullHoursPerDay;

        var hoursToWork = Math.Min(shortenedHoursPerDay, hoursPerDay);

        return days * (hoursPerDay - hoursToWork);
    })
    .Sum();

static int GetVacationDays(IEnumerable<VacationRequest> vacations, Guid userId,
DateTime from, DateTime to) =>
    vacations.Where(v => v.UserId == userId)
        .Select(v => CountDays(v, from, to, v => v.Start, v => v.End))
        .Sum();

static int GetSickLeaveDays(IEnumerable<SickLeave> sickLeaves, Guid userId,
DateTime from, DateTime to) =>
    sickLeaves.Where(sl => sl.UserId == userId)
        .Select(sl => CountDays(sl, from, to, sl => sl.Start, sl => sl.End))
        .Sum();

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        static double GetActualHours(IEnumerable<WorkSession> workSessions, Guid userId,
        DateTime from, DateTime to) =>
            workSessions.Where(ws => ws.UserId == userId)
                .Select(ws =>
                    {
                        var actualStart = ws.Start < from ? from : ws.Start;
                        var actualEnd = ws.End > to ? to : ws.End!.Value;

                        if (actualEnd > actualStart)
                            return (actualEnd - actualStart).TotalHours;

                        return 0.0;
                    })
                .Sum();

        static int GetWorkingDays(DateTime from, DateTime to)
        {
            if (to <= from)
                throw new ArgumentException("Cannot calculate working days with invalid
dates");

            var dayDifference = (int)to.Subtract(from).TotalDays + 1;
            return Enumerable
                .Range(0, dayDifference)
                .Select(d => from.AddDays(d))
                .Count(d => d.DayOfWeek != DayOfWeek.Saturday && d.DayOfWeek !=
DayOfWeek.Sunday);
        }

        static int CountDays<T>(
            T value,
            DateTime from,
            DateTime to,
            Func<T, DateTime> getStart,
            Func<T, DateTime> getEnd)
        {
            var originalStart = getStart(value);
            var originalEnd = getEnd(value);

            var start = originalStart < from ? from : originalStart;
            var end = originalEnd > to ? to : originalEnd;

            return Enumerable.Range(0, (int)(end - start).TotalDays + 1)
                .Select(d => start.AddDays(d))
                .Count(d => d.DayOfWeek != DayOfWeek.Saturday && d.DayOfWeek !=
DayOfWeek.Sunday);
        }

        record struct HoursSummaryResult(
            int WorkingDays,
            int Holidays,
            int DayOffs,
            IEnumerable<IndividualHoursSummary> IndividualHoursSummaries);
    }

```

Клас AverageWorkTimeCalculator:

```

public class AverageWorkTimeCalculator(
    UserRepository userRepository,
    WorkSessionRepository workSessionRepository,
    IConfiguration configuration)
{

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

readonly WorkSessionRepository workSessionRepository = workSessionRepository;

readonly int FullHoursPerDay = configuration.GetValue("WorkHours:FullDay", 8);

public async Task<WorkSessionAverageSummary?> Calculate(Guid userId)
{
    var user = await userRepository.GetOneAsync(userId)
        ?? throw new ArgumentException("Invalid user id", nameof(userId));

    var workSessions = await GetSuitableWorkSessions(user);

    if (!workSessions.Any())
        return null;

    var averageWorkHours = CalculateAverageWorkHoursPerDay(workSessions);

    var monthlyAverageStart = new TimeSpan((long)averageWorkHours.Average(t =>
t.AverageStart.Ticks));
    var monthlyAverageEnd = new TimeSpan((long)averageWorkHours.Average(t =>
t.AverageEnd.Ticks));

    return new(monthlyAverageStart, monthlyAverageEnd, averageWorkHours);
}

async Task<IEnumerable<WorkSession>> GetSuitableWorkSessions(User user)
{
    var end = DateTime.UtcNow.Date;
    var start = end.AddMonths(-1);

    var workSessions = await workSessionRepository.GetInRangeAsync([user.Id],
start, end, hidePlanned: true);

    var hoursPerDay = FullHoursPerDay / 100.0 * user.EmploymentRate;
    var minDurationOffset = hoursPerDay / 8;
    var maxDurationOffset = hoursPerDay * 1.25;

    return workSessions.Where(ws => ws.End.HasValue
        && (ws.End.Value - ws.Start).TotalHours >= minDurationOffset
        && (ws.End.Value - ws.Start).TotalHours <= maxDurationOffset);
}

static List<DailyAverageWorkHours>
CalculateAverageWorkHoursPerDay(IEnumerable<WorkSession> sessions)
{
    var dates = sessions.Select(ws => ws.Start.Date).Distinct().OrderBy(date =>
date);

    var averageWorkHoursPerDay = new List<DailyAverageWorkHours>();
    foreach (var date in dates)
    {
        var sessionsForDate = sessions.Where(ws => ws.Start.Date == date);
        var avgStartTime = new TimeSpan((long)sessionsForDate.Average(ws =>
ws.Start.TimeOfDay.Ticks));
        var avgEndTime = new TimeSpan((long)sessionsForDate.Average(ws =>
ws.End!.Value.TimeOfDay.Ticks));

        averageWorkHoursPerDay.Add(new(date, avgStartTime, avgEndTime));
    }

    return averageWorkHoursPerDay;
}

Unit-тест клас HoursSummaryCalculatorTests:

public class HoursSummaryCalculatorTests
{

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

readonly Mock<IUserRepository> userRepository;
readonly Mock<IHolidayRepository> holidayRepository;
readonly Mock<IWorkSessionRepository> workSessionRepository;
readonly Mock<IVacationRequestRepository> vacationRepository;
readonly Mock<ISickLeaveRepository> sickLeaveRepository;
readonly IConfiguration configuration;

readonly HoursSummaryCalculator target;

public HoursSummaryCalculatorTests()
{
    userRepository = new Mock<IUserRepository>();
    holidayRepository = new Mock<IHolidayRepository>();
    workSessionRepository = new Mock<IWorkSessionRepository>();
    vacationRepository = new Mock<IVacationRequestRepository>();
    sickLeaveRepository = new Mock<ISickLeaveRepository>();

    var inMemorySettings = new Dictionary<string, string> {
        {"WorkHours:FullDay", "8"},
    };

    configuration = new ConfigurationBuilder()
        .AddInMemoryCollection(inMemorySettings)
        .Build();

    target = new HoursSummaryCalculator(
        userRepository.Object,
        holidayRepository.Object,
        workSessionRepository.Object,
        vacationRepository.Object,
        sickLeaveRepository.Object,
        configuration);
}

[Fact]
public void GetWorkingDays_ThrowsArgumentException_WhenFromDateIsLaterThanToDate()
{
    var from = new DateTime(2024, 6, 2);
    var to = new DateTime(2024, 6, 1);

    userRepository.Setup(r => r.GetAllActiveAsync()).ReturnsAsync([]);

    Assert.ThrowsAsync<ArgumentException>(() =>
target.CalculateHoursSummaryAsync(from, to));
}

[Fact]
public async Task CalculateHoursSummaryPageAsync_ReturnsCorrectHoursSummaryPage()
{
    var userPageInput = new UserPageInput();
    var from = new DateTime(2024, 6, 1);
    var to = new DateTime(2024, 6, 30);

    var users = new List<User> { new() { Id = Guid.NewGuid(), EmploymentRate = 100
} };
    userRepository.Setup(r => r.GetPageAsync(userPageInput)).ReturnsAsync(new
Page<User>(1, users));

    var holidays = new List<Holiday> { new() { Date = new DateTime(2024, 6, 12),
Type = HolidayType.DayOff } };
    holidayRepository.Setup(r => r.GetByDateRangeAsync(from,
to)).ReturnsAsync(holidays);

    var vacations = new List<VacationRequest> { new() { UserId = users[0].Id,
Start = new DateTime(2024, 6, 2), End = new DateTime(2024, 6, 3) } };

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        vacationRepository.Setup(r =>
r.GetUsersVacationsInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from,
to)).ReturnsAsync(vacations);

        var sickLeaves = new List<SickLeave> { new() { UserId = users[0].Id, Start =
new DateTime(2024, 6, 4), End = new DateTime(2024, 6, 5) } };
        sickLeaveRepository.Setup(r =>
r.GetSickLeaveInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from,
to)).ReturnsAsync(sickLeaves);

        var workSessions = new List<WorkSession> { new() { UserId = users[0].Id, Start
= new DateTime(2024, 6, 6, 9, 0, 0), End = new DateTime(2024, 6, 6, 17, 0, 0) } };
        workSessionRepository.Setup(r =>
r.GetInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from, to,
true)).ReturnsAsync(workSessions);

        var result = await target.CalculateHoursSummaryPageAsync(userPageInput, from,
to);

        Assert.NotNull(result);
        Assert.Equal(1, result.Page.TotalAmount);
        Assert.Equal(20, result.WorkingDays);
        Assert.Equal(1, result.Holidays);
        Assert.Equal(1, result.DayOffs);

        var summary = result.Page.Items.First();
        Assert.Equal(users[0], summary.User);
        Assert.Equal(128, summary.ExpectedHours);
        Assert.Equal(8, summary.ActualHours);
        Assert.Equal(0.062, summary.Percent, 3);
        Assert.Equal(1, summary.DaysInVacation);
        Assert.Equal(2, summary.DaysInSickLeave);
    }

    [Fact]
    public async Task
CalculateHoursSummaryPageAsync_ExcludesShortDaysFromActualResult()
    {
        var userPageInput = new UserPageInput();
        var from = new DateTime(2024, 6, 1);
        var to = new DateTime(2024, 6, 30);

        var users = new List<User> { new() { Id = Guid.NewGuid(), EmploymentRate = 50
} };
        userRepository.Setup(r => r.GetPageAsync(userPageInput)).ReturnsAsync(new
Page<User>(1, users));

        var holidays = new List<Holiday> { new() {
            Date = new DateTime(2024, 6, 12),
            EndDate = new DateTime(2024, 6, 13),
            Type = HolidayType.ShortDay,
            HoursToWork = 3,
        } };
        holidayRepository.Setup(r => r.GetByDateRangeAsync(from,
to)).ReturnsAsync(holidays);

        var vacations = new List<VacationRequest>();
        vacationRepository.Setup(r =>
r.GetUsersVacationsInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from,
to)).ReturnsAsync(vacations);

        var sickLeaves = new List<SickLeave>();
        sickLeaveRepository.Setup(r =>
r.GetSickLeaveInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from,
to)).ReturnsAsync(sickLeaves);

```

```

        var workSessions = new List<WorkSession> {
            new() { UserId = users[0].Id, Start = new DateTime(2024, 6, 6, 9, 0, 0),
                End = new DateTime(2024, 6, 6, 17, 0, 0) },
            new() { UserId = users[0].Id, Start = new DateTime(2024, 6, 7, 9, 0, 0),
                End = new DateTime(2024, 6, 7, 17, 0, 0) },
            new() { UserId = users[0].Id, Start = new DateTime(2024, 6, 8, 9, 0, 0),
                End = new DateTime(2024, 6, 8, 17, 0, 0) },
        };
        workSessionRepository.Setup(r =>
            r.GetInRangeAsync(It.IsAny<IEnumerable<Guid>>(), from, to,
                true)).ReturnsAsync(workSessions);

        var result = await target.CalculateHoursSummaryPageAsync(userPageInput, from,
            to);

        Assert.NotNull(result);
        Assert.Equal(1, result.Page.TotalAmount);
        Assert.Equal(20, result.WorkingDays);
        Assert.Equal(2, result.Holidays);
        Assert.Equal(0, result.DayOffs);

        var summary = result.Page.Items.First();
        Assert.Equal(users[0], summary.User);
        Assert.Equal(78, summary.ExpectedHours);
        Assert.Equal(24, summary.ActualHours);
        Assert.Equal(0, summary.DaysInVacation);
        Assert.Equal(0, summary.DaysInSickLeave);
    }
}

```

		Мельниченко Д.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Вакалюк Т.А.				97
Змн.	Арк.	№ докум.	Підпис	Дата		