**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система управління людськими ресурсами на базі технологій ASP.NET Core MVC»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_____ Денис ХОДАК
*(підпис)*

Виконав: здобувач(ка) вищої освіти групи ТЦР-43
Денис ХОДАК

Керівник: Ждан ЧЕРНЯВСЬКИЙ
*ст. вклд.*

Рецензент: _____

**Київ 2026**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ходак Денис Павлович

1. Тема кваліфікаційної роботи: «Інформаційна система управління людськими ресурсами на базі технологій ASP.NET Core MVC»
керівник кваліфікаційної роботи ст. вклд. Ждан ЧЕРНЯВСЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про архітектурні патерни та технології розробки web-застосунків, технічна документація ASP.NET Core MVC, ASP.NET Core Identity та Entity Framework Core, специфікації протоколів аутентифікації та авторизації, опис методів проектування реляційних баз даних та моделей управління доступом на основі ролей (RBAC).
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)https://meet.google.com/pec-imot-yed

1. Проаналізувати існуючі програмні аналоги систем управління людськими ресурсами, зокрема SAP SuccessFactors, BambooHR, OrangeHRM та Zoho People, дослідити їхні можливості та обмеження, а також обґрунтувати вибір технологічного стеку на основі отриманих результатів.

2. Спроєктувати вимоги до програмного забезпечення шляхом визначення ролей і потреб користувачів, сформулювати функціональні та нефункціональні вимоги до системи управління людськими ресурсами і задокументувати їх у структурованому вигляді.
3. Дослідити та порівняти технології ASP.NET Core MVC, ASP.NET Core Identity, Entity Framework Core, SQLite та SQL Server, оцінити їхню сумісність і відповідність сформованим вимогам, після чого обґрунтувати доцільність використання обраного стеку.
4. Спроєктувати та розробити програмне забезпечення, визначивши архітектуру системи, структури даних і схему бази даних, реалізувати функції CRUD і рольову модель доступу, провести тестування компонентів та перевірити відповідність системи встановленим вимогам.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Діаграма варіантів використання.
   5. Алгоритм роботи застосунку.
   6. Рольова модель доступу
   7. Екранні форми.
   8. Апробація результатів дослідження

6. Дата видачі завдання «20» лютого 2026 року.

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів кваліфікаційної роботи | Строк виконання | Примітка |
|---|---|---|---|
| 1 | Дата видачі завдання | 20.02.2026 | Виконано |
| 1 | Підбір та аналіз науково-технічної літератури з тематики HRMS та web-розробки | 20.02-27.02.2026 | Виконано |
| 2 | Аналіз існуючих програмних рішень для управління людськими ресурсами та визначення вимог | 28.02-10.03.2026 | Виконано |
| 3 | Вибір та обґрунтування технологічного стеку (ASP.NET Core MVC, Identity, Entity Framework Core, SQLit) | 08.03-15.03.2026 | Виконано |
| 4 | Проектування вимог та архітектури інформаційної системи управління людськими ресурсами | 16.03-25.03.2026 | Виконано |
| 5 | Програмна реалізація застосунку: CRUD-модулі, рольова модель доступу (RBAC), автентифікація | 26.03-18.04.2026 | Виконано |
| 6 | Тестування застосунку | 9.04-26.04.2026 | Виконано |
| 7 | Оформлення роботи: вступ, висновки, реферат | 27.04-03.05.2026 | Виконано |
| 8 | Розробка демонстраційних матеріалів | 04.05-06.05.2026 | Виконано |
| 9 | Попередній захист роботи | 07.05-11.05.2026 | Виконано |

Здобувач(ка) вищої освіти _____ Денис ХОДАК
*(підпис)*

Керівник
кваліфікаційної роботи _____ Ждан ЧЕРНЯВСЬКИЙ
*(підпис)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 9 табл., 16 рис., 15 джерел.

*Мета роботи* – розробка web-орієнтованої інформаційної системи управління людськими ресурсами, що забезпечує автоматизацію HR-процесів, розмежування доступу за ролями та зручний інтерфейс для всіх категорій користувачів.

*Об'єкт дослідження* – процес автоматизації управління людськими ресурсами в організації.

*Предмет дослідження* – інформаційна система управління людськими ресурсами, розроблена на базі технологій ASP.NET Core MVC.

*Короткий зміст роботи:* У роботі проаналізовано предметну галузь управління людськими ресурсами та досліджено існуючі програмні аналоги: SAP SuccessFactors, BambooHR, OrangeHRM, Zoho People. Сформовано функціональні та нефункціональні вимоги до системи. Проведено огляд та порівняльний аналіз технологічного стеку: ASP.NET Core MVC, ASP.NET Core Identity, Entity Framework Core, SQLite та SQL Server. Спроектовано архітектуру системи за патерном MVC, розроблено структуру реляційної бази даних. Реалізовано ключові модулі: облік працівників з фільтрацією та пошуком, управління відділами та посадами, подання й обробка заяв на відпустку, рольова модель доступу з чотирма рівнями (Адміністратор, HR-менеджер, Керівник, Працівник), автентифікація через використання інтегрованих засобів безпеки та сучасних алгоритмів криптографічного захисту персональних даних. Розроблено інтерфейс користувача на основі Razor Views та Bootstrap. Проведено функціональне тестування системи на основі реальних сценаріїв використання.

Сферою використання розробленої системи є малі та середні організації, які потребують автоматизації кадрових процесів без придбання дорогого комерційного програмного забезпечення.

КЛЮЧОВІ СЛОВА: HRMS, ASP.NET CORE MVC, ENTITY FRAMEWORK CORE, РОЛЬОВА МОДЕЛЬ ДОСТУПУ, RBAC, АВТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, WEB-ЗАСТОСУНОК, УПРАВЛІННЯ ПЕРСОНАЛОМ, MVC.

**Зміст**

# АПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HRMS – Human Resource Management System

MVC – Model-View-Controller

ORM – Object-Relational Mapping

RBAC – Role-Based Access ControlA

EF Core – Entity Framework Core

CRUD – Create, Read, Update, Delete

UI/UX – User Interface / User Experience

HTTP – HyperText Transfer Protocol

SQL – Structured Query Language

CSRF – Cross-Site Request Forgery

API – Application Programming Interface

JWT – JSON Web Token

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

DI – Dependency Injection

# ВСТУП

Актуальність та основні характеристики дослідження. У сучасних умовах стрімкого розвитку інформаційних технологій управління людськими ресурсами (HR) перетворилось на один із ключових чинників конкурентоспроможності підприємств будь-якого масштабу. Цифрова трансформація бізнес-процесів охоплює дедалі більше сфер діяльності організацій, і кадрове адміністрування не є винятком. Ефективна організація HR-процесів – ведення обліку працівників, управління організаційною структурою, обробка заяв на відпустку, чітке розмежування прав доступу між різними категоріями користувачів -безпосередньо впливає на продуктивність праці, якість управлінських рішень і загальну операційну ефективність підприємства. Автоматизація цих процесів дозволяє суттєво скоротити витрати робочого часу на рутинні адміністративні операції, мінімізувати ризик людських помилок та забезпечити прозорість кадрової документації.

Разом з тим переважна більшість малих і середніх організацій досі покладається на застарілі засоби обліку персоналу: електронні таблиці, паперові журнали або розрізнені локальні програми. Такі рішення не лише не забезпечують належного рівня автоматизації та безпеки, але й суттєво ускладнюють спільну роботу різних підрозділів, унеможливлюють централізований контроль доступу до чутливих кадрових даних і не відповідають сучасним вимогам до зручності використання. Впровадження ж готових комерційних HR-систем часто виявляється економічно недоцільним для організацій з обмеженим бюджетом.

Об'єктом дослідження є процес автоматизації управління людськими ресурсами в організації.

Предметом дослідження є інформаційна система управління людськими ресурсами, розроблена на базі технологій ASP.NET Core MVC.

Мета роботи покращення ефективності адміністрування ключових HR-процесів через створення web-орієнтованої інформаційної системи з гнучким розмежуванням прав доступу та інтуїтивним інтерфейсом, придатної до швидкої адаптації без значних витрат.

Методи дослідження охоплюють декілька взаємопов'язаних підходів: аналіз предметної галузі та вивчення існуючих програмних аналогів на ринку HR-рішень; порівняльний аналіз їхніх функціональних можливостей і виявлення ключових недоліків; концептуальне та фізичне проектування бази даних із урахуванням вимог до цілісності та масштабованості; проектування багаторівневої архітектури системи; практична реалізація з використанням ASP.NET Core MVC, ASP.NET Core Identity та Entity Framework Core як основного стеку технологій; функціональне тестування на основі реальних сценаріїв використання з перевіркою коректності розмежування прав доступу.

Наукова новизна роботи полягає у комплексній реалізації механізму рольового управління доступом (RBAC) на базі розширеного ASP.NET Core Identity у поєднанні з повним забезпечення повного циклу управління інформаційними активами персоналу системи. Запропоноване рішення забезпечує гнучке та чітко структуроване розмежування прав між чотирма рольовими рівнями – адміністраторами, HR-менеджерами, керівниками підрозділів та рядовими працівниками – при збереженні єдиної кодової бази та уніфікованого інтерфейсу. Такий підхід, на відміну від більшості аналогічних рішень початкового рівня, поєднує простоту розгортання з достатньою гнучкістю для адаптації до організаційної структури підприємства різного типу.

Практична значущість результатів дослідження полягає у можливості безпосереднього впровадження розробленої системи в організаціях різного масштабу та галузевої приналежності. Система забезпечує автоматизацію обліку персоналу, управління організаційною структурою підприємства та повний цикл обробки заяв на відпустку – від подачі працівником до затвердження керівником – без необхідності придбання дорогого комерційного програмного забезпечення. Це робить розроблене рішення особливо актуальним для бюджетних установ, некомерційних організацій та підприємств малого і середнього бізнесу, що прагнуть підвищити ефективність кадрового адміністрування за мінімальних фінансових витрат.

1. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Аналіз методів та технологій для розробки інформаційних систем управління людськими ресурсами
2. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Проектування та реалізація інформаційної системи управління людськими ресурсами на базі ASP.NET Core MVC

За підсумками апробації було оприлюднено тези доповідей [1-2].

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Характеристика системи управління людськими ресурсами

Управління людськими ресурсами (Human Resource Management, HRM) являє собою комплекс управлінських процесів, що охоплюють планування, набір, розвиток, мотивацію та утримання персоналу організації. Інформаційна система управління людськими ресурсами (HRMS -Human Resource Management System) – це програмне забезпечення, що автоматизує зазначені процеси, забезпечує централізоване зберігання даних про персонал і надає інструменти для ефективного управління кадровими операціями [3].

Сучасні HRMS виконують такі ключові функції: облік персональних даних працівників; управління організаційною структурою підприємства (відділи, посади, ієрархія); нарахування заробітної плати та контроль зайнятості; управління відпустками та відсутностями; контроль доступу на основі ролей; формування звітів та аналітики. Ці функції безпосередньо пов'язані між собою і утворюють єдину інформаційну екосистему, що відображає поточний стан кадрового потенціалу організації.

Розроблювана система орієнтована на середні та малі підприємства і реалізує такі модулі: облік працівників з можливістю фільтрації та пошуку; управління відділами та посадами; подання та обробка заяв на відпустку; рольова модель доступу з чотирма рівнями – Адміністратор, HR-менеджер, Керівник та Працівник; автентифікація та авторизація через ASP.NET Core Identity.

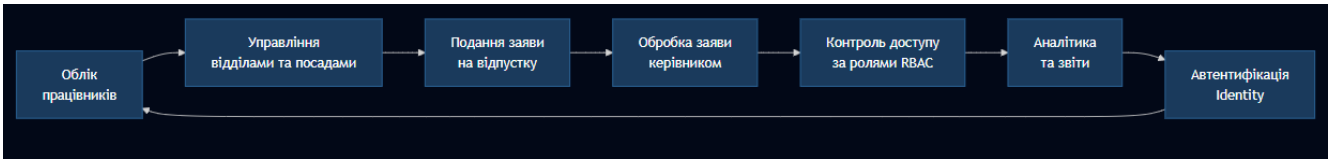



Рис. 1.1. Загальний життєвий цикл управління людськими ресурсами

З точки зору інженерної реалізації web-застосунку для управління людськими ресурсами важливою є фаза моніторингу та контролю, що забезпечує стабільність, безпеку і відповідність системи вимогам організації. Ключовим завданням цього етапу є забезпечення цілісності даних про персонал та прозорості кадрових процесів. Система повинна реєструвати всі зміни (журнал аудиту), підтримувати паролі з хешуванням PBKDF2, а доступ до функцій розмежовувати на основі ролей та Claims.

У системі передбачено чотири рівні користувачів:

1. Адміністратор – повний доступ до всіх модулів, включно з управлінням користувачами та ролями, видаленням відділів і посад.
2. HR-менеджер – перегляд усіх даних, CRUD-операції з працівниками, відділами та посадами, розгляд заяв на відпустку.
3. Керівник – перегляд працівників свого відділу, перегляд структури організації.
4. Працівник – перегляд власного профілю, подання заяв на відпустку.

Такий підхід забезпечує принцип найменших привілеїв, що є критичним для захисту персональних даних та запобігання несанкціонованим змінам у кадрових записах.

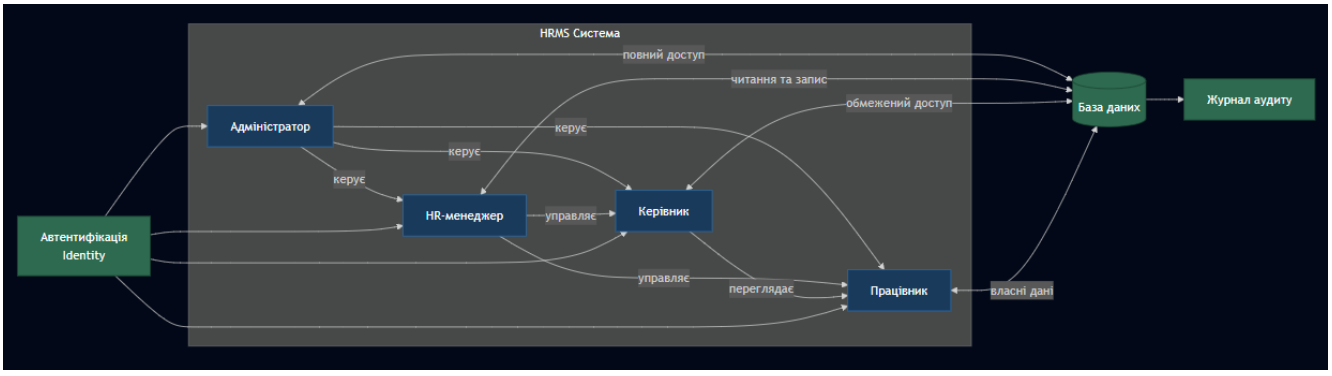



Рис. 1.2. Діаграма ролей та їх взаємодії в HRMS

## 1.2 Аналіз аналогів

На сучасному ринку програмного забезпечення представлено надзвичайно широкий спектр інструментів для управління людськими ресурсами – від масштабних монолітних корпоративних ERP-систем, що охоплюють увесь спектр бізнес-процесів організації, до легких хмарних сервісів, орієнтованих на потреби малого та середнього бізнесу. Стрімкий розвиток технологій та зростаючі вимоги до цифровізації кадрового адміністрування спричинили появу великої кількості конкуруючих рішень із принципово різними архітектурними підходами, моделями ліцензування та цільовими аудиторіями. Для обґрунтованого вибору технологічного стеку та формування чітких функціональних вимог до власної розробки доцільно здійснити порівняльний аналіз провідних аналогів, розглянувши їхні функціональні характеристики, архітектурні особливості, переваги та обмеження.

У цьому підрозділі представлено огляд чотирьох поширених HRMS-платформ, що мають різне позиціонування на ринку та різні підходи до вирішення задач кадрового менеджменту. Вибір саме цих систем зумовлений їхньою широкою розповсюдженістю у практиці вітчизняних та зарубіжних підприємств, а також репрезентативністю з точки зору охоплення різних сегментів ринку – від великих корпоративних рішень до доступних інструментів для невеликих команд. Особливу увагу в ході аналізу приділено таким критеріям оцінювання: рівень функціональності та повнота охоплення HR-процесів; гнучкість налаштування під специфіку конкретної організації; реалізація рольової моделі розмежування доступу; складність розгортання та подальшого супроводу системи; загальна вартість володіння рішенням, включаючи витрати на ліцензування, впровадження та підтримку.

Проведений огляд дозволяє не лише визначити сильні та слабкі сторони кожного з аналізованих рішень, але й виявити функціональні прогалини, які є спільними для більшості існуючих продуктів і можуть бути усунені в рамках власної розробки. Зокрема, характерною проблемою комерційних систем є

надмірна складність і надлишковість функціоналу для організацій із невеликим штатом, тоді як безкоштовні альтернативи нерідко поступаються у якості реалізації рольового контролю доступу та зручності користувацького інтерфейсу. Отримані в результаті аналізу висновки безпосередньо впливають на формування переліку функціональних вимог до розроблюваної системи та обґрунтування обраного технологічного стеку.

### 1.2.1 SAP SuccessFactors

SAP SuccessFactors – це провідна хмарна платформа управління персоналом від компанії SAP, орієнтована на великі корпорації та міжнародні організації [4]. Система охоплює повний цикл HR-процесів: від підбору персоналу та адаптації новачків до управління продуктивністю, навчанням і компенсацією. Платформа вирізняється глибокою інтеграцією з іншими продуктами SAP (ERP, CRM, Finance) та підтримкою понад 40 мов.

На сьогодні можна виділити такі особливості системи SAP SuccessFactors:

- повний модульний набір HR-функцій (Core HR, рекрутинг, продуктивність, навчання, компенсація);
- глибока аналітика та звітність з використанням машинного навчання;
- підтримка мобільного доступу через нативні застосунки;
- відповідність вимогам локального законодавства у понад 100 країнах.

Перевагами системи є її комплексність, що дозволяє охопити всі аспекти HR без інтеграції сторонніх рішень, надійна масштабованість для організацій із десятками тисяч працівників, а також потужні аналітичні інструменти.

Недоліками є висока вартість ліцензування та впровадження, що обмежує доступність для малого та середнього бізнесу, складність налаштування із потребою залучення спеціалізованих партнерів SAP, а також надмірна функціональність для невеликих організацій.

### 1.2.2 BambooHR

BambooHR – популярна хмарна HR-платформа, розроблена спеціально для малого та середнього бізнесу [5]. Сервіс позиціонується як простий у використанні інструмент з мінімальним порогом входження, що дозволяє навіть невеликим компаніям ефективно управляти персоналом без залучення IT-спеціалістів.

Особливості BambooHR:

- зручний інтуїтивний інтерфейс з мінімалістичним дизайном;
- функціонал самообслуговування для працівників (заяви на відпустку, перегляд власних даних);
- інтеграція з популярними ATS-системами та сервісами нарахування заробітної плати;
- мобільний застосунок для iOS та Android.

Перевагами системи є швидке впровадження, що дозволяє розпочати роботу протягом кількох годин, доступна ціна для малого бізнесу, а також висока оцінка задоволеності користувачів.

Недоліками є обмежені можливості кастомізації звітності, відсутність розширеної модульної структури, зокрема модулів навчання або компенсації в базовому плані, а також непридатність для великих організацій із комплексними HR-процесами.

### 1.2.3 OrangeHRM

OrangeHRM – відома відкрита HR-система з відкритим вихідним кодом, що розповсюджується за моделлю Open Source та надає базовий функціонал безкоштовно [6]. Система орієнтована на організації, що прагнуть отримати контроль над власними даними та не залежати від хмарних провайдерів.

Особливості OrangeHRM:

- безкоштовна Community Edition з базовими модулями (облік персоналу, відпустки, звітність);
- можливість самостійного розгортання на власному сервері;

- модульна архітектура з можливістю підключення платних розширень;
- підтримка понад 15 мов.

Перевагами системи є відсутність щомісячної абонентської плати у базовій версії, повний контроль над даними завдяки зберіганню на власній інфраструктурі, а також велика спільнота розробників.

Недоліками є застарілий інтерфейс у безкоштовній версії, складне початкове налаштування, що потребує технічних знань, а також обмежена технічна підтримка у безкоштовній версії.

### 1.2.4 Zoho People

Zoho People – хмарна HRMS-платформа від індійської компанії Zoho Corporation, що є частиною екосистеми Zoho із понад 50 бізнес-застосунків. Система орієнтована на малий та середній бізнес і пропонує гнучку цінову політику з багатьма рівнями підписки [7].

Особливості Zoho People:

- інтеграція з усіма продуктами екосистеми Zoho (CRM, Books, Recruit, Desk);
- модуль управління відпустками з автоматичним розрахунком залишку днів;
- портал самообслуговування для працівників;
- відстеження робочого часу та продуктивності.

Перевагами системи є гнучка цінова модель із безкоштовним планом для організацій до п’яти працівників, широка інтеграція в межах екосистеми Zoho, а також мобільний застосунок із відстеженням геолокації.

Недоліками є обмежена функціональність безкоштовного плану, залежність від екосистеми Zoho, що ускладнює міграцію, а також менш розвинена рольова модель порівняно з корпоративними рішеннями.

## 1.3 Порівняння аналогів

Зведені результати аналізу характеристик розглянутих систем наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати аналізу HRMS-платформ

| Параметр оцінювання | SAP Success Factors | BambooHR | OrangeHRM | Zoho People | Розроблювана система |
|---|---|---|---|---|---|
| Облік персоналу | + | + | + | + | +, повний CRUD |
| Рольова модель доступу | +, складна | +, базова | +, базова | +, обмежена | +, 4 ролі (RBAC) |
| Управління відпустками | + | + | + | + | +, 6 типів |
| Відкритий вихідний код | – | – | + | – | + |
| Самостійне розгортання | – | – | + | частково | + |
| Вартість | Висока | Середня | Безкоштовна | Низька | Безкоштовна |
| Адаптація під задачу | –, надлишкова | +, часткова | +, часткова | +, часткова | +, повна |
| Простота освоєння | –, складна | +, легка | –, середня | +, легка | +, легка |

На основі проведеного порівняльного аналізу характеристик п'яти програмних засобів для управління людськими ресурсами можна сформулювати

низку узагальнених висновків, що безпосередньо визначають вектор подальшої розробки власної системи та обґрунтовують доцільність її створення як самостійної альтернативи наявним рішенням на ринку.

По-перше, розглядаючи функціонал у сфері обліку персоналу, варто зазначити, що всі проаналізовані платформи забезпечують його підтримку на базовому або розширеному рівні. Однак ступінь адаптації під специфічні потреби конкретного підприємства суттєво відрізняється від системи до системи. SAP SuccessFactors вирізняється найбагатшим і найповнішим набором інструментів серед усіх розглянутих рішень, проте саме ця всеохопність стає її принциповим слабким місцем у контексті застосування в малому та середньому бізнесі. Висока складність налаштування, тривалий процес впровадження і значні фінансові витрати фактично унеможливлюють її практичне використання більшістю вітчизняних компаній. Рішення середнього класу – зокрема BambooHR та Zoho People – пропонують достатньо збалансований набір можливостей і є більш доступними, проте їхня гнучкість обмежена наперед визначеними шаблонами та типовими конфігураціями, які нерідко не збігаються з реальною організаційною структурою та бізнес-процесами конкретного замовника.

По-друге, проведений аналіз виявив значну неоднорідність підходів до реалізації рольового розмежування прав доступу в різних системах. Переважна більшість розглянутих платформ використовують спрощену дворівневу або трирівневу рольову модель, яка не відображає справжньої ієрархії прийняття рішень і розподілу відповідальності у типовій організації. Натомість розроблювана система запроваджує чотирирівневу RBAC-модель із чітко визначеними ролями Admin, HR-Manager, Manager та Employee. Така архітектура є оптимальним поєднанням гнучкості у налаштуванні прав та зручності повсякденного адміністрування, дозволяючи точно відтворити реальну управлінську вертикаль організації без надмірного ускладнення системи.

По-третє, усі розглянуті комерційні рішення функціонують виключно за хмарною SaaS-моделлю, що неминуче породжує стійку технологічну та операційну залежність від стороннього постачальника послуг. Окрім цього, така модель несе

потенційні ризики щодо конфіденційності та захисту чутливих кадрових даних, особливо в умовах мінливого регуляторного середовища. Для організацій із суворими законодавчими вимогами до обробки персональних даних або з внутрішньою корпоративною політикою локального зберігання інформації подібна модель є принципово неприйнятною. Розроблювана система, на відміну від аналогів, підтримує повністю автономне розгортання на власній інфраструктурі замовника без будь-якої залежності від зовнішніх хмарних сервісів. Це відкриває реальну можливість її застосування в державних органах, освітніх і наукових закладах, медичних установах та інших організаціях із підвищеними вимогами до безпеки та захисту персональних даних працівників.

Таким чином, розроблювана HRMS займає чітко визначену нішу між простими хмарними сервісами для малого бізнесу та масштабними і дорогими корпоративними платформами. Вона органічно поєднує достатній для більшості організацій функціонал, повну відсутність ліцензійних відрахувань і можливість самостійного розгортання на власних потужностях – саме ту комбінацію характеристик, якої критично бракує переважній більшості наявних ринкових аналогів і яка робить систему особливо привабливою для вітчизняного ринку.

# 2 ПРОЄКТУВАННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Визначення ролей та потреб користувачів

У сучасних умовах цифровізації управлінських процесів програмне забезпечення (ПЗ) відіграє ключову роль у забезпеченні зручності, ефективності та автоматизації роботи з персоналом. Зростаюча кількість організацій прагне замінити паперовий облік та розрізнені електронні таблиці на єдину інтегровану систему, здатну охопити весь спектр HR-операцій. У зв'язку з цим зростає потреба у створенні адаптивних програмних засобів, що задовольняють індивідуальні потреби різних категорій користувачів.

Проектування програмного забезпечення передбачає формалізацію вимог до його функціоналу, інтерфейсу, надійності, масштабованості та інших характеристик. Важливо враховувати не лише технічні параметри, а й контекст використання, особливості цільової аудиторії та можливі сценарії взаємодії користувача з системою. Вимоги до ПЗ класифікуються на дві основні категорії: функціональні та нефункціональні. Функціональні вимоги описують конкретні дії, що повинна виконувати система (наприклад, створити запис про працівника, фільтрувати список за відділом, подати заяву на відпустку). Нефункціональні вимоги встановлюють обмеження та стандарти якості виконання функцій: продуктивність, безпеку, масштабованість, зручність інтерфейсу тощо.

У контексті розробки HRMS особливого значення набувають вимоги до інтерфейсу користувача, розмежування прав доступу, захисту персональних даних та можливості масштабування системи під потреби організацій різного розміру. Таким чином, аналіз ролей і потреб є фундаментом для формалізації функціональних можливостей системи та побудови її архітектури.

Одним із вирішальних компонентів залучення аудиторії до використання системи є зручність користування. Сучасний HR-фахівець або адміністратор очікує інтуїтивно зрозумілий інтерфейс, швидке виконання типових операцій та

адаптивність до різних пристроїв. Саме тому проектування розпочинається з чіткого визначення ролей та їхніх потреб.

Одним із основних етапів проектування будь-якої інформаційної системи є ідентифікація основних категорій користувачів, які взаємодіятимуть із розробленим програмним забезпеченням, та аналіз їхніх потреб. Чітке розуміння ролей дозволяє сформулювати якісніші вимоги до функціональності, забезпечити релевантні інтерфейсні рішення та гарантувати зручність використання. У межах розробленої HRMS виокремлюють такі основні ролі:

Адміністратор – це технічний або управлінський персонал, відповідальний за конфігурацію системи, управління обліковими записами та загальний контроль над її функціонуванням. Ця роль передбачає повний доступ до управління обліковими записами користувачів, включаючи їх створення, блокування та призначення ролей, а також CRUD-доступ до всіх ключових сутностей системи – працівників, відділів і посад. Адміністратор має можливість переглядати журнали дій та статистику системи, а також керувати ролями та правами доступу через вбудований Identity-механізм. Для задоволення цих потреб система надає адміністративну панель із повним набором інструментів управління та можливістю делегування окремих функцій ролі HR-менеджера.

HR-менеджер є основним оперативним користувачем системи, що безпосередньо виконує кадрові операції в межах своїх повноважень. До функціональних потреб цієї ролі належать реєстрація нових працівників та редагування їхніх профілів, управління організаційною структурою підприємства – відділами, посадами та призначеннями, – а також перегляд усіх поданих заяв на відпустку і прийняття відповідних рішень щодо них. Крім того, HR-менеджер має доступ до інструментів пошуку і фільтрації персоналу за різними критеріями, а також може формувати аналітичні огляди та звіти про поточний стан кадрового складу організації.

Керівник підрозділу (Manager) – це посадова особа, яка має доступ виключно до даних свого відділу та працює з інформацією, що стосується безпосередньо підпорядкованого йому персоналу. В межах своїх повноважень керівник може

переглядати список підлеглих працівників і їхні профілі, опрацьовувати заяви на відпустку, подані співробітниками відділу, а також ознайомлюватися із загальною організаційною структурою підприємства для розуміння місця свого підрозділу в ієрархії.

Працівник (Employee) – це рядовий користувач системи, функціональні можливості якого обмежені операціями з власними персональними даними та заявами. Зокрема, працівник має право переглядати власний профіль та особисту інформацію, подавати заяви на відпустку різних типів і відстежувати статус уже поданих заяв, а також самостійно змінювати пароль та редагувати контактні дані у своєму обліковому записі.

Таким чином, чітке окреслення ролей дозволяє сформувати функціональні та нефункціональні вимоги з урахуванням різних сценаріїв використання системи.


Рис. 2.1. Діаграма потреб основних ролей системи HRMS

Прецедент UC-01 описує процес подачі заяви на відпустку, де актором виступає працівник з активним обліковим записом, який ініціює процес натисканням кнопки «Подати заяву» у відповідному розділі. Система відображає форму, в якій працівник обирає тип відпустки – щорічну, лікарняну, за власний рахунок, навчальну тощо – та вказує дати початку і закінчення, після чого система

автоматично розраховує кількість днів і показує наявний залишок, а за бажанням працівник може додати коментар. Після натискання кнопки «Надіслати» система реєструє заяву з унікальним номером і надсилає сповіщення відповідальному HR-менеджеру або керівнику підрозділу. Якщо запитувана кількість днів перевищує залишок, система показує попередження і блокує надсилання, а якщо обраний період збігається із затвердженою відпусткою або святковим днем – повідомляє про конфлікт і пропонує змінити діапазон дат. Підсумком успішного проходження сценарію є збереження заяви зі статусом «Очікує розгляду», її відображення у розділі «Мої заяви» особистого кабінету працівника та надходження до черги опрацювання HR-менеджера або керівника підрозділу.

Прецедент UC-02 описує процес розгляду та затвердження заяви на відпустку, де актором виступає автентифікований HR-менеджер із відповідними правами, який розпочинає роботу після отримання сповіщення або самостійного переходу до розділу «Управління заявами» за наявності щонайменше однієї заяви зі статусом «Очікує розгляду». Система відображає відсортований список заяв із фільтрацією за відділом, типом, датою та статусом, після чого HR-менеджер відкриває потрібну заяву і переглядає її деталі: ПІБ працівника, посаду, відділ, запитуваний період, залишок днів і коментар. Залежно від прийнятого рішення HR-менеджер обирає «Затвердити» або «Відхилити»: у разі затвердження статус заяви змінюється, використані дні списуються з балансу і вноситься відповідний запис до журналу, а у разі відхилення обов'язково вводиться причина такого рішення. Якщо заява потребує уточнень або додаткових документів, HR-менеджер встановлює статус «Потребує уточнення» і надсилає працівнику відповідне повідомлення, після чого заява автоматично повертається до черги опрацювання зі збереженням усього листування. Підсумком розгляду є оновлення статусу заяви, а у випадку затвердження – зменшення залишку відпускних днів, внесення запису до журналу відпусток і надсилання працівнику email-сповіщення з результатом розгляду.

Прецедент UC-03 описує процес реєстрації нового працівника в системі, де актором виступає автентифікований HR-менеджер із правами на створення

облікових записів, який ініціює процес натисканням кнопки «Додати працівника» у розділі «Персонал» за умови, що відповідні відділ і посада вже існують у системі. Система відображає багатокрокову форму реєстрації, яка складається з трьох кроків: на першому вносяться особисті дані працівника – ПІБ, дата народження, стать, телефон та email; на другому визначається організаційне положення – підрозділ, посада і безпосередній керівник; на третьому вказуються умови роботи – дата прийому, тип зайнятості та початковий баланс відпускних днів. Після збереження форми система автоматично перевіряє унікальність email, коректність форматів і наявність усіх обов'язкових полів, після чого створює профіль працівника, генерує тимчасовий пароль і надсилає лист-запрошення з інструкцією для першого входу. Якщо введений email вже зареєстровано в системі, відображається повідомлення про дублювання і форма не зберігається, а незаповнені або некоректні обов'язкові поля підсвічуються з блокуванням збереження до їх виправлення. Підсумком успішної реєстрації є збереження профілю та активація облікового запису, надсилання працівнику email-запрошення з тимчасовими даними для входу, негайна доступність інформації для HR-менеджера й адміністратора, а також відображення нового працівника в організаційній схемі підрозділу та загальному реєстрі персоналу.

## 2.2 Функціональні вимоги

У межах цього проєкту функціональні вимоги подано як перелік основних можливостей системи, і кожна вимога супроводжується детальним поясненням її ролі та значення у загальній архітектурі HRMS. Такий підхід дозволяє не лише зафіксувати перелік реалізованих функцій, а й обґрунтувати архітектурні рішення, що лежать в їх основі.

Реєстрація та автентифікація користувачів є однією з ключових функціональних складових системи, що забезпечує повний цикл управління обліковими записами – від їх початкового створення до безпечної авторизації. Для входу в систему використовується комбінація електронної пошти та пароля,

реалізована засобами ASP.NET Core Identity. Паролі користувачів не зберігаються у відкритому вигляді: для їх захисту застосовується алгоритм хешування PBKDF2 [9], який забезпечує стійкість до атак методом перебору та злому через таблиці передобчислених значень.

Після успішної автентифікації для кожного користувача формується сесія, до якої прив'язуються Claims – структуровані атрибути, що містять інформацію про роль користувача в системі та його унікальний ідентифікатор. Ці дані використовуються в подальшому для авторизації доступу до окремих функцій і ресурсів HRMS відповідно до рольової моделі.

Управління обліковими записами та ролями передбачає, що адміністратор може створювати нові облікові записи з автоматичною генерацією тимчасового пароля, призначати ролі (Admin, HR, Manager, Employee), блокувати або розблоковувати доступ, при цьому система підтримує рольову модель на основі Claims через ASP.NET Core Identity з атрибутами [Authorize(Roles = "...")], що є критичною вимогою для реалізації принципу найменших привілеїв та захисту конфіденційних кадрових даних.

Управління картками працівників (CRUD) передбачає, що система забезпечує повний цикл операцій із картками працівників, включаючи створення, перегляд, редагування та видалення записів, при цьому картка містить ПІБ, по батькові, дату народження, адресу, табельний номер, зарплату, тип зайнятості (повна, часткова або контракт), статус (активний, у відпустці або звільнений), посаду, відділ і безпосереднього керівника, а також передбачено фільтрацію за відділом, статусом і текстовий пошук за ПІБ, що разом забезпечує точний облік і зручну навігацію.

Управління відділами та посадами передбачає, що HR-менеджер та адміністратор можуть створювати, редагувати та видаляти відділи і посади, при цьому відділ містить назву, код, опис і посилання на керівника, а посада містить назву, опис, мінімальну та максимальну зарплату та прив’язку до відділу, що формує ієрархічну структуру, яка відображає реальну організаційну схему підприємства і забезпечує коректне призначення працівників.

Система управління заявами на відпустку надає працівнику можливість подати заяву одного з шести доступних типів: щорічна, лікарняна, навчальна, без збереження заробітної плати, декретна або додаткова відпустка. Після подання заява отримує статус «На розгляді», після чого HR-менеджер опрацьовує її та переводить у статус «Схвалено» або «Відхилено», а за потреби заява може бути переведена у статус «Скасовано». При прийнятті рішення HR-менеджер має можливість залишити коментар із поясненням або уточненням. Окрім цього, система автоматично розраховує загальну кількість використаних днів відпустки та відображає актуальні дані безпосередньо у профілі відповідного працівника.

Система забезпечує адміністраторів та HR-менеджерів комплексним зведеним дашбордом, що відображає ключові показники діяльності організації – загальну кількість працівників, детальний розподіл за їхніми статусами, поточну кількість активних заяв на відпустку та перелік нових надходжень до системи, виступаючи головною точкою доступу одразу після успішного входу й забезпечуючи користувачам швидке та зрозуміле уявлення про актуальний стан HR-процесів без необхідності переходу до інших розділів.

Кожен автентифікований користувач системи має повноцінний доступ до власного персонального профілю, де може самостійно редагувати особисті дані та за потреби змінювати пароль із обов'язковою перевіркою поточного пароля для підтвердження особи, а вся ця функціональність реалізована через спеціалізований AccountController із відповідними методами Profile та ChangePassword, які спираються на можливості компонента UserManager<ApplicationUser> зі стандартної бібліотеки ASP.NET Core Identity.

Система містить окрему, суворо розмежовану панель управління, призначену виключно для адміністраторів і повністю недоступну рядовим користувачам, у межах якої уповноважені особи можуть переглядати всі наявні облікові записи, тимчасово блокувати доступ окремим користувачам, гнучко призначати та змінювати ролі й видаляти застарілі або некоректні дані, що забезпечує централізований і прозорий контроль над усією системою, суттєво підвищуючи її стабільність, керованість та загальний рівень безпеки.

Зведений перелік функціональних вимог з прив'язкою до ролей наведено у таблиці 2.1.

Таблиця 2.1

Функціональні вимоги до HRMS з прив'язкою до ролей

| Функціональна вимога | Admin | HR | Manager | Employee |
|---|---|---|---|---|
| Реєстрація та автентифікація | + | + | + | + |
| Управління обліковими записами та ролями | + | - | - | - |
| CRUD-операції зі працівниками | + | + | - | - |
| Перегляд власного профілю | + | + | + | + |
| Управління відділами та посадами | + | + | - | - |
| Перегляд структури організації | + | + | + | - |
| Подання заяви на відпустку | + | + | - | + |
| Перегляд та обробка заяв на відпустку | + | + | частково | власні |
| Дашборд із кадровою статистикою | + | + | - | - |
| Зміна пароля та редагування профілю | + | + | + | + |
| Блокування облікових записів | + | - | - | - |

## 2.3 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні якісні характеристики системи, що не залежать безпосередньо від конкретної функціональності, але суттєво впливають на досвід користування, надійність та безпеку застосунку. На відміну від функціональних вимог, які описують що саме система повинна

виконувати, нефункціональні вимоги відповідають на питання -наскільки добре і в яких умовах вона це робить. Нижче подано основні нефункціональні вимоги до розроблюваної HRMS із детальним обґрунтуванням їх важливості та описом конкретних технічних засобів реалізації кожної з них.

Зручність користування (Usability) означає, що застосунок має інтуїтивно зрозумілий інтерфейс, який дозволяє HR-фахівцям, керівникам підрозділів і рядовим працівникам без спеціальної технічної підготовки самостійно виконувати основні кадрові операції без звернення до документації чи служби підтримки. Інтерфейс розроблено з урахуванням принципів мінімалізму та когнітивного розвантаження користувача, де навігаційні елементи логічно згруповані за функціональними зонами, а всі дії супроводжуються інформативними повідомленнями про результат із чітким поясненням причин помилки або підтвердженням успіху. Валідація даних реалізована як на стороні клієнта для забезпечення миттєвого зворотного зв’язку, так і на стороні сервера для гарантування цілісності інформації. Критичні незворотні дії, зокрема видалення працівника або скасування заяви, супроводжуються обов’язковим діалогом підтвердження для запобігання випадковим помилкам оператора. Такий підхід безпосередньо впливає на швидкість адаптації нових користувачів і суттєво знижує кількість помилок під час роботи з чутливими кадровими даними.

Продуктивність (Performance) передбачає, що система повинна забезпечувати швидку обробку запитів навіть при роботі з великою кількістю кадрових записів, при цьому час відгуку сторінок не має перевищувати двох секунд для типових операцій читання. Досягнення високої продуктивності забезпечується використанням асинхронних викликів бази даних через патерн async/await у контролерах, оптимізованими LINQ-запитами із застосуванням методу Include() для eager loading пов’язаних сутностей, а також обов’язковою пагінацією при виведенні списків працівників і заяв на відпустку. Повільна робота системи обліку персоналу негативно впливає на ефективність HR-відділу та може спричиняти незадоволення серед працівників.

Надійність (Reliability) полягає в тому, що застосунок має стабільно функціонувати без збоїв і забезпечувати цілісність кадрових даних за будь-яких умов використання, включаючи обробку некоректних вхідних даних, паралельні запити від кількох користувачів і можливі переривання мережевого з'єднання. Цілісність даних гарантується на рівні бази даних за допомогою зовнішніх ключів, каскадного видалення пов'язаних записів та обмежень Required і MaxLength, які унеможливлюють збереження неповних або некоректних даних. На рівні застосунку застосовуються транзакції Entity Framework Core для атомарного виконання складених операцій, що забезпечує або повне застосування змін, або їх повне скасування без ризику часткової фіксації пошкоджених або суперечливих даних. Оскільки персональні дані працівників і кадрова документація мають критичне значення для організації з юридичної та операційної точок зору, втрата або пошкодження будь-якої інформації є абсолютно неприйнятними.

Система реалізує комплексний багаторівневий захист, що охоплює автентифікацію, авторизацію, захист від поширених мережевих атак і контроль спроб несанкціонованого входу, при цьому сукупність реалізованих заходів відповідає базовим вимогам стандарту OWASP Top 10 для захисту вебзастосунків [8]. Автентифікація базується на механізмі Cookie Authentication з PBKDF2-хешуванням паролів через ASP.NET Core Identity, завдяки чому оригінальний пароль неможливо відновити навіть у разі повного витоку бази даних, оскільки зберігається лише односпрямований хеш із сіллю. Авторизація здійснюється через атрибути [Authorize(Roles = "...")] на рівні контролерів і окремих методів дії, що забезпечує гранульований контроль доступу відповідно до чотирирівневої RBAC-моделі – кожна роль має строго визначений перелік дозволених операцій, а спроба отримати доступ поза межами своїх повноважень автоматично перенаправляє користувача на сторінку відмови. Захист від міжсайтових підроблених запитів (CSRF) реалізовано через атрибут [ValidateAntiForgeryToken] у всіх POST-методах, що унеможливлює виконання будь-яких дій від імені автентифікованого користувача зі стороннього сайту, а автоматичне блокування облікового запису

після п'яти послідовних невдалих спроб входу надійно захищає систему від атак перебору паролів.

Масштабованість системи забезпечується завдяки закладеним архітектурним рішенням, що дозволяють зберігати повну працездатність при поступовому збільшенні кількості працівників, відділів і заяв без необхідності суттєвого рефакторингу коду чи перепроектування архітектури. Патерн MVC із чітким розподілом відповідальностей між рівнями презентації, бізнес-логіки та доступу до даних, а також механізм Dependency Injection забезпечують легке підключення нових модулів і розширення існуючих без порушення наявної функціональності. Перехід з SQLite, що використовується у середовищі розробки, на повноцінний SQL Server у промисловому середовищі потребує зміни лише одного рядка у файлі конфігурації, оскільки бізнес-логіка та всі LINQ-запити залишаються повністю незмінними завдяки абстракції Entity Framework Core.

Супроводжуваність системи безпосередньо визначає вартість її підтримки у довгостроковій перспективі, тому код повинен бути структурованим, добре читабельним і легко розширюваним. Досягається це завдяки чіткому розподілу на контролери, ViewModels, сервіси та Views, що забезпечує логічну ізоляцію відповідальностей і спрощує навігацію по кодовій базі. Використання строго типізованих ViewModels з анотаціями DataAnnotations дозволяє централізувати валідацію вхідних даних і уникнути її дублювання у різних частинах застосунку. Коментування ключових методів і нетривіальних алгоритмічних рішень додатково знижує поріг входження для розробників, які долучаються до подальшої підтримки та розвитку системи.

Кросплатформеність системи забезпечує коректне відображення застосунку у всіх сучасних браузерах – Chrome, Firefox, Edge і Safari – а також на пристроях з різними роздільними здатностями екранів, від смартфонів до широкоформатних моніторів. Адаптивний дизайн реалізовано засобами власного CSS без залежності від зовнішніх UI-фреймворків, що знижує загальне навантаження на клієнт і прискорює початкове завантаження сторінок. Крім того, платформа ASP.NET Core підтримує розгортання на Windows, Linux і macOS, що надає організації повну

свободу у виборі серверної інфраструктури відповідно до наявних ресурсів і корпоративних стандартів.

У результаті проведеного аналізу функціональних та нефункціональних вимог до розроблюваного програмного забезпечення визначено повний перелік ключових можливостей системи та сформульовано загальні вимоги до її якісного функціонування. Сукупність зазначених вимог формує цілісну архітектуру програмного продукту, орієнтованого на якість, ефективність та безпечність взаємодії HR-фахівців з інформаційним середовищем управління персоналом. Важливо підкреслити, що кожна з визначених нефункціональних вимог не є абстрактним побажанням, а підкріплена конкретними технологічними рішеннями в межах обраного стеку. На рисунку 2.2 зображено принципову різницю між функціональними та нефункціональними вимогами.

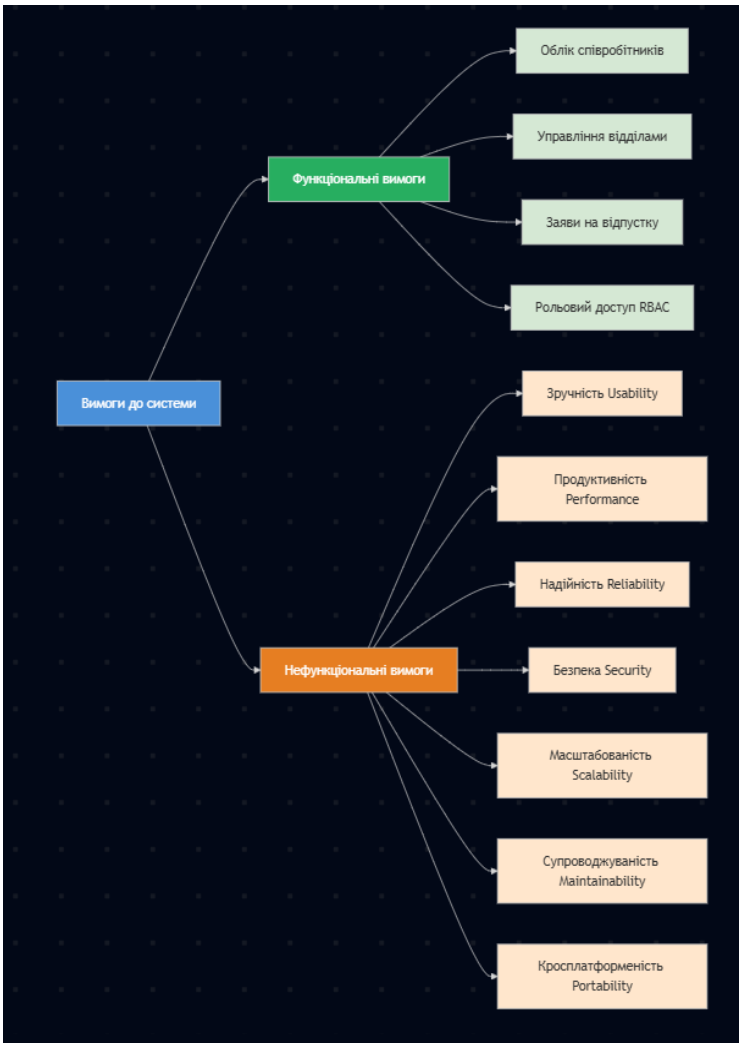



Рис. 2.2. Різниця між функціональними та нефункціональними вимогами

Таким чином, зведення вимог до єдиного узгодженого переліку забезпечує несуперечливість проектних рішень на всіх наступних етапах розробки -від проектування архітектури до тестування. Кожна нефункціональна вимога безпосередньо пов'язана з конкретними технічними рішеннями, що будуть детально розглянуті у розділах 3 та 4.

Зведений перелік нефункціональних вимог та засобів їх реалізації наведено у таблиці 2.2.

Таблиця 2.2

Нефункціональні вимоги та засоби їх реалізації

| № | Вимога | Засіб реалізації |
|---|---|---|
| 1 | Зручність користування | Валідація DataAnnotations, TempData-повідомлення, Razor Views |
| 2 | Продуктивність | async/await у контролерах, LINQ Include(), пагінація |
| 3 | Надійність | Foreign Keys, каскадне видалення, транзакції EF Core |
| 4 | Безпека | PBKDF2, [Authorize], [ValidateAntiForgeryToken], блокування |
| 5 | Масштабованість | MVC-архітектура, Dependency Injection, змінна БД |
| 6 | Супроводжуваність | ViewModels, коментарі, чіткий поділ контролерів |
| 7 | Кросплатформеність | ASP.NET Core (Win/Linux/macOS), адаптивний CSS |

# 3 ЗАСОБИ РЕАЛІЗАЦІЇ

## 3.1 Фреймворк для розробки вебзастосунків

ASP.NET Core MVC – це кросплатформовий, високопродуктивний веб-фреймворк з відкритим вихідним кодом, розроблений компанією Microsoft. Він побудований на основі шаблону проєктування Model-View-Controller (MVC), що дозволяє чітко розділити відповідальності між компонентами застосунку та забезпечити його підтримку і масштабованість [10].

Архітектура MVC передбачає три основні компоненти: модель (Model) – відповідає за бізнес-логіку та роботу з даними; представлення (View) – ідповідає за відображення інформації користувачеві; контролер (Controller) – обробляє запити користувача, взаємодіє з моделлю та формує відповідь у вигляді представлення.

Для розробки HRMS обрано ASP.NET Core 8 LTS як найбільш актуальну версію з підтримкою до 2026 року. Ця версія включає вбудовану підтримку Dependency Injection (DI), Middleware Pipeline, Routing та Razor Views – усіх необхідних механізмів для реалізації інформаційної системи.

Основними перевагами ASP.NET Core MVC для даного проєкту є: кросплатформеність – робота на Windows, Linux та macOS; висока продуктивність – оптимізоване оброблення запитів Kestrel-сервером; вбудована підтримка авторизації та аутентифікації через Identity; повна інтеграція з Entity Framework Core; підтримка Razor Views для генерації HTML на стороні сервера; активна спільнота та офіційна документація Microsoft.

Середовищем розробки обрано Microsoft Visual Studio 2022, яке забезпечує повноцінну підтримку C# 12, зручне налагодження, інтеграцію з Git та вбудовані інструменти для роботи з .NET проєктами.

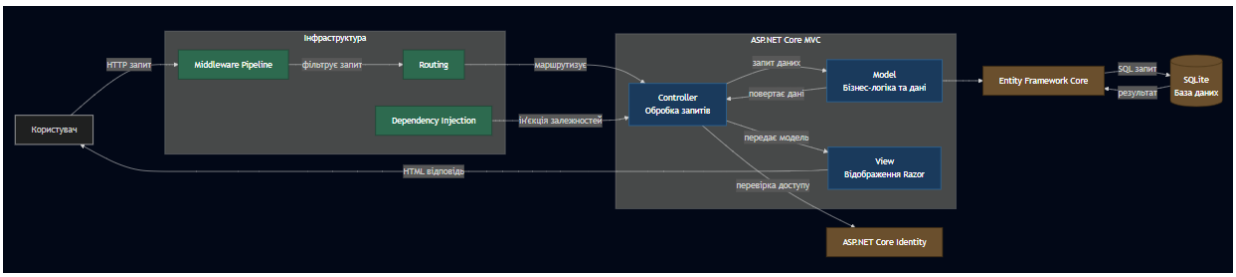



Рис. 3.1. Архітектура ASP.NET Core MVC

Структура проєкту HRMS містить такі ключові каталоги: Controllers/ – 6 контролерів (AccountController, EmployeesController, DepartmentsController, PositionsController, LeaveRequestsController, DashboardController); Models/ – 5 моделей сутностей; ViewModels/ – 8 строго типізованих моделей представлення; Views/ – 22 Razor-шаблони; wwwroot/ – статичні ресурси (CSS, JS, зображення).

Маршрутизація у проєкті налаштована за замовчуванням у вигляді {controller}/{action}/{id?}, що забезпечує зрозумілу URL-структуру. Наприклад: /Employees/Index – список працівників, /LeaveRequests/Create – форма подачі заяви на відпустку, /Dashboard/Index – головна панель користувача.

## 3.2 Система автентифікації та управління користувачами

ASP.NET Core Identity – це система управління членством (membership system), яка вбудована у фреймворк ASP.NET Core і забезпечує функції реєстрації, аутентифікації та авторизації користувачів. Вона зберігає дані про користувачів у реляційній базі даних через Entity Framework Core [11].

У проєкті HRMS клас ApplicationUser розширює стандартний IdentityUser та містить додаткові властивості, специфічні для системи управління персоналом: LastName (прізвище), FirstName (ім'я), MiddleName (по батькові), Address (адреса проживання), BirthDate (дата народження), PhoneNumber (телефон). Завдяки цьому кожен запис у таблиці AspNetUsers безпосередньо пов'язаний із записом Employee, а вся персональна інформація зберігається у двох пов'язаних таблицях.

Рольова модель реалізована через IdentityRole та містить чотири рівні доступу: Administrator – повний доступ до всіх функцій системи та адміністративної панелі, HR – управління персоналом, відділами, посадами, розгляд заяв, Manager – перегляд підлеглих свого відділу, Employee – перегляд власного профілю, подача заяв на відпустку.

Авторизація доступу до дій контролерів реалізується за допомогою атрибутів [Authorize] та [Authorize(Roles = "Admin,HR")], що дозволяє декларативно обмежувати доступ. Налаштування Identity у Program.cs включає такі параметри безпеки: мінімальна довжина пароля – 8 символів; обов'язкові великі та малі літери, цифри та спеціальні символи; блокування після 5 невдалих спроб входу на 15 хвилин; унікальна електронна адреса для кожного користувача.

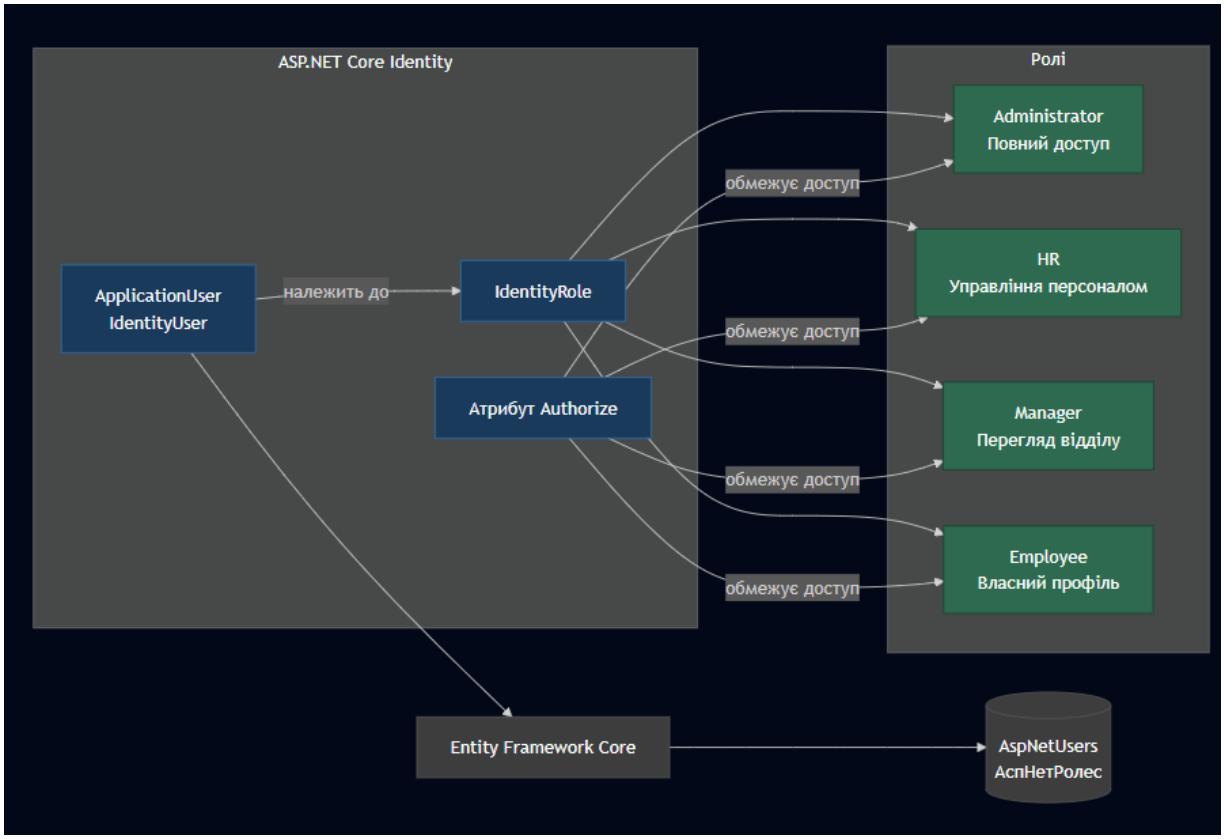



Рис. 3.2. Схема ролей та прав доступу в системі HRMS

Генерація тимчасових паролів для нових працівників реалізована через метод GenerateRandomPassword(), який формує рядок із заданими вимогами Identity. При першому вході система пропонує змінити тимчасовий пароль. Зберігання паролів

здійснюється виключно у хешованому вигляді за алгоритмом PBKDF2 з випадковою сіллю (salt), що забезпечує захист від атак за словником та веселкових таблиць.

### 3.3 Інструмент об'єктно-реляційного відображення

Entity Framework Core (EF Core) – це сучасний, легкий, розширюваний і кросплатформовий ORM (Object-Relational Mapper) для платформи .NET, що є офіційно рекомендованим інструментом доступу до даних у екосистемі Microsoft. Головна ідея ORM полягає у тому, щоб усунути так звану «об'єктно-реляційну невідповідність» – принципову різницю між тим, як дані представляються у реляційній базі даних у вигляді таблиць і рядків, і тим, як вони моделюються у застосунку у вигляді об'єктів і класів. EF Core дозволяє розробникам працювати з базою даних виключно через об'єкти C#, не пишучи SQL-запитів вручну: усі операції читання, запису, оновлення та видалення транслюються у відповідні SQL-команди автоматично [12]. Це суттєво прискорює розробку, знижує ризик синтаксичних помилок у запитах і робить код незалежним від конкретного діалекту SQL, що використовується обраною СУБД.

У проєкті HRMS застосовується підхід Code-First, за якого схема бази даних генерується автоматично на основі опису класів-моделей та їхніх відносин, визначених безпосередньо у коді C#. Це означає, що розробник працює лише з класами та властивостями, а EF Core самостійно транслює їх у відповідні таблиці, стовпці, індекси та зовнішні ключі. Будь-які зміни у моделях фіксуються у вигляді міграцій – автоматично згенерованих скриптів, що інкрементально оновлюють схему бази даних без її повного перестворення і без втрати наявних даних.

Центральним елементом рівня доступу до даних є клас HrmsDbContext, який успадковується від IdentityDbContext<ApplicationUser> – спеціалізованого базового класу, що поєднує стандартний контекст EF Core з інфраструктурою ASP.NET Core Identity. Таке успадкування автоматично додає до схеми бази даних усі необхідні таблиці для зберігання облікових записів, ролей і їхніх зв'язків, які

Identity потребує для своєї роботи, не вимагаючи їх ручного визначення. Клас HrmsDbContext містить чотири DbSet<T> – властивості, кожна з яких відповідає окремій сутності предметної області: Employees для зберігання даних про працівників, Departments для відділів, Positions для посад і LeaveRequests для заяв на відпустку. Саме через ці властивості виконуються всі LINQ-запити у контролерах і сервісах застосунку – DbSet<T> є точкою входу для будь-якої операції з відповідною таблицею бази даних.

Основні відносини між сутностями налаштовані через Fluent API у методі OnModelCreating, що є більш гнучким і явним способом конфігурації порівняно з атрибутами DataAnnotations. Fluent API дозволяє описати складні відносини, каскадну поведінку та обмеження безпосередньо у коді конфігурації, відокремлюючи їх від класів-моделей і забезпечуючи кращу читабельність схеми. Відносини один-до-багатьох між Department та Employee відображають реальну організаційну структуру: один відділ може об'єднувати довільну кількість працівників, тоді як кожен працівник належить рівно до одного відділу. Аналогічна кардинальність реалізована між Position та Employee – одна посада може обіймати декількох працівників одночасно.

Особливої уваги заслуговують два специфічні зв'язки у схемі даних. Перший – це відносини самопосилання Employee → Employee, що моделюють ієрархію керівник-підлеглий: кожен працівник може мати посилання на іншого працівника як на свого безпосереднього керівника, що дозволяє відтворити довільно глибоку організаційну ієрархію без додаткових таблиць. Другий -відносини один-до-одного між ApplicationUser та Employee, що пов'язують обліковий запис системи автентифікації з відповідним кадровим записом працівника, забезпечуючи єдину точку ідентифікації користувача в обох підсистемах.

Використання Entity Framework Core з підходом Code-First і конфігурацією через Fluent API забезпечує чітке розмежування між логікою застосунку та деталями зберігання даних. Розробник оперує виключно об'єктами предметної області, тоді як усі аспекти взаємодії з базою даних залишаються прозорими та

керованими через єдиний контекст. Такий підхід суттєво спрощує як початкову розробку, так і подальший супровід системи при зміні вимог до структури даних.

Міграції EF Core дозволяють керувати змінами схеми БД у контрольованому порядку. При першому запуску застосунку метод context.Database.Migrate() у Program.cs автоматично застосовує всі наявні міграції та заповнює початкові дані (seed data): ролі Identity, тестовий адміністратор, тестовий HR-менеджер, кілька відділів та посад.

LINQ-запити використовуються для вибірки даних із підтримкою фільтрації, сортування та пагінації. Наприклад, отримання списку активних працівників відділу реалізовано за допомогою методів розширення «LINQ», які забезпечують фільтрацію за статусом, алфавітне сортування та «жадібне завантаження» (eager loading) пов'язаних сутностей відділу та посади з бази даних.

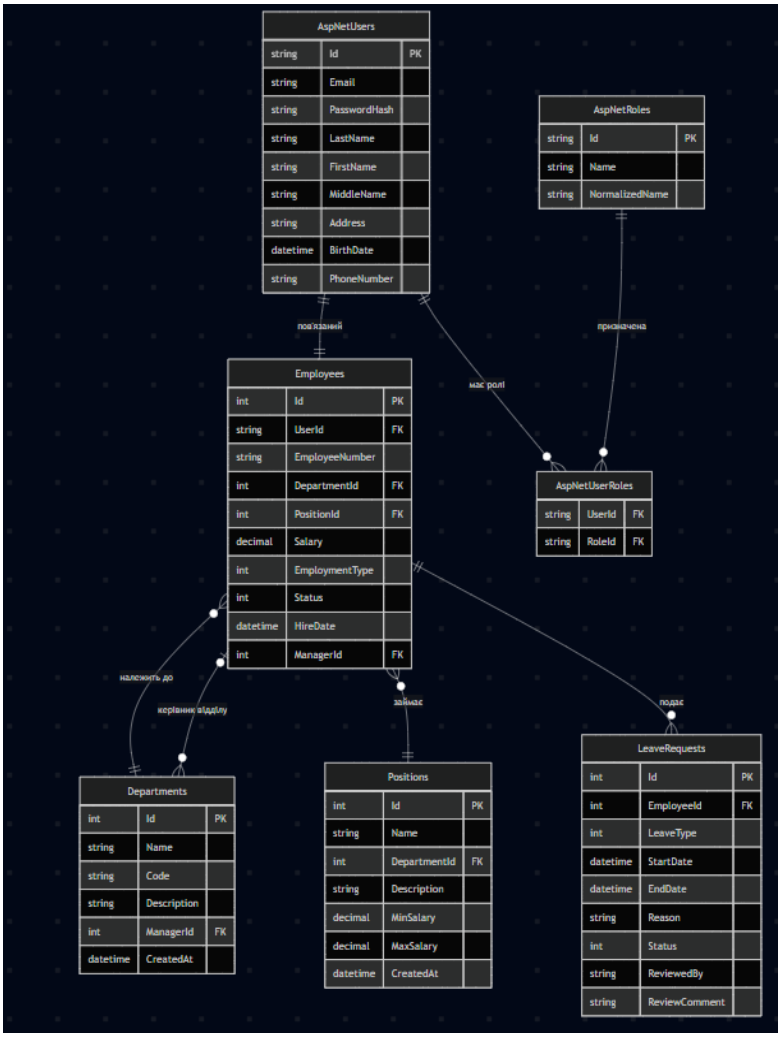



Рис. 3.3. Діаграма сутностей бази даних HRMS

## 3.4 Використані бази даних

Для зберігання даних у проєкті HRMS використовуються дві системи управління базами даних залежно від середовища розгортання. Такий підхід є поширеною практикою у сучасній web-розробці: він дозволяє максимально спростити процес локальної розробки та тестування, зберігаючи при цьому повну готовність до розгортання у промисловому середовищі з мінімальними змінами у конфігурації. Вибір конкретної СУБД визначається потребами середовища, а не архітектурою застосунку – завдяки абстракції Entity Framework Core обидва варіанти є повністю взаємозамінними.

На етапі розробки та тестування застосовується SQLite – легка файлова база даних, яка не потребує встановлення окремого серверного процесу та зберігає всі дані в єдиному файлі hrms.db [13]. Ця особливість робить SQLite винятково зручним інструментом для командної розробки: новий розробник може розгорнути повністю робоче середовище буквально за хвилини, без конфігурування сервера баз даних, створення облікових записів або налаштування мережевого доступу. Усі схемні зміни застосовуються через стандартний механізм міграцій EF Core, а сам файл бази даних за потреби можна просто скопіювати або видалити для повного скидання стану системи. Підтримка SQLite в Entity Framework Core реалізована через пакет Microsoft.EntityFrameworkCore.Sqlite і вимагає мінімальної конфігурації у файлі appsettings.json – фактично одного рядка із вказівкою шляху до файлу бази даних. Це робить SQLite ідеальним вибором не лише для прототипування, але й для демонстрації роботи системи замовнику без розгортання повноцінної серверної інфраструктури.

Принциповою перевагою обраного підходу є те, що заміна СУБД не торкається жодного рядка бізнес-логіки. Увесь код контролерів, сервісів і репозиторіїв працює однаково незалежно від того, яка база даних використовується під капотом – SQLite чи SQL Server. Це стало можливим завдяки тому, що весь доступ до даних здійснюється виключно через абстракцію DbContext і LINQ-запити, без жодного нативного SQL-коду прив'язаного до конкретного діалекту.

Для виробничого середовища передбачено можливість переходу на Microsoft SQL Server -повноцінну реляційну СУБД корпоративного рівня, що забезпечує розширені можливості у порівнянні з SQLite: підтримку конкурентних підключень від багатьох користувачів одночасно, розвинені механізми резервного копіювання та відновлення, детальне журналювання транзакцій, а також інструменти моніторингу продуктивності. SQL Server є стандартом де-факто для застосунків на платформі ASP.NET Core у корпоративному середовищі та повністю підтримується Entity Framework Core через пакет Microsoft.EntityFrameworkCore.SqlServer. Перехід з SQLite на SQL Server здійснюється двома змінами: заміною рядка підключення у файлі appsettings.json та заміною виклику UseSqlite() на UseSqlServer() у файлі Program.cs під час реєстрації контексту бази даних. Усі існуючі міграції при цьому залишаються повністю сумісними і застосовуються до нової бази даних без жодних модифікацій, що гарантує ідентичність схеми в обох середовищах.

Таблиця 3.1

Порівняльна характеристика СУБД

| Характеристика | SQLite | MS SQL Server | PostgreSQL |
|---|---|---|---|
| Тип | Вбудована файлова БД | Клієнт-серверна СУБД | Клієнт-серверна СУБД |
| Ліцензія | Public Domain | Комерційна / Free | PostgreSQL License (відкрита) |
| Масштабованість | Низька (файл) | Висока | Висока |
| Налаштування | Не потрібне | Потрібен сервер | Потрібен сервер |
| Розгортання | Дуже просте | Складне | Середнє |
| Підтримка EF Core | Так | Так | Так |
| Використання в проєкті | Розробка / тестування | Можливий перехід | Можливий перехід |

Таким чином, дворівнева стратегія вибору СУБД забезпечує оптимальний баланс між зручністю розробки та готовністю до промислового використання. Організація, що впроваджує систему, отримує можливість самостійно обрати

відповідну СУБД виходячи з наявної інфраструктури, кваліфікації адміністраторів і вимог до масштабованості – без будь-яких змін у коді застосунку.

Структура бази даних включає наступні основні таблиці: AspNetUsers – дані облікових записів (розширений IdentityUser); AspNetRoles, AspNetUserRoles – ролі та їх призначення; Employees – табельні номери, зарплата, тип зайнятості, дата прийому; Departments – відділи з кодами та керівниками; Positions – посади із зарплатними вилками; LeaveRequests – заяви на відпустку з типом, датами та статусом.

## 3.5 Порівняння технологічних стеків

Для обґрунтування вибору технологічного стеку проведено порівняльний аналіз ASP.NET Core MVC з трьома альтернативними веб-фреймворками: Django (Python), Laravel (PHP) та Spring Boot (Java). Вибір саме цих платформ зумовлений їхньою широкою розповсюдженістю у корпоративній веб-розробці та репрезентативністю з точки зору охоплення різних технологічних стеків. Аналіз виконувався за шістьма ключовими критеріями: продуктивність, вбудована підтримка автентифікації, зрілість ORM, кросплатформеність, складність входу та рівень підтримки з боку виробника. Кожен із розглянутих фреймворків має власну філософію проектування і типові сценарії застосування, тому вибір між ними потребує системного підходу, а не ґрунтується виключно на особистих уподобаннях розробника.

За результатами аналізу ASP.NET Core MVC продемонстрував найкращу сукупність характеристик для задач проєкту. Вбудована система ASP.NET Core Identity забезпечує нативну підтримку RBAC без залучення сторонніх бібліотек, Entity Framework Core з підходом Code-First спрощує роботу з базою даних, а висока продуктивність платформи підтверджена незалежними бенчмарками TechEmpower. Django і Laravel є простішими у початковому освоєнні, проте поступаються за гнучкістю рольової моделі, тоді як Spring Boot, попри порівнянну

потужність, має значно вищий поріг входу та потребує більшого обсягу шаблонного коду.

Таблиця 3.2

Порівняння технологічних стеків для розробки HRMS

| Критерій | ASP.NET Core MVC | Django (Python) | Laravel (PHP) | Spring Boot (Java) |
|---|---|---|---|---|
| Мова програмування | C# | Python | PHP | Java |
| Продуктивність | Висока | Середня | Середня | Висока |
| Вбудована авт-ція | ASP.NET Identity | Django Auth | Laravel Passport | Spring Security |
| ORM | EF Core | Django ORM | Eloquent | Hibernate |
| Підтримка СУБД | SQL Server, SQLite, PostgreSQL | PostgreSQL, MySQL, SQLite | MySQL, PostgreSQL | Oracle, MySQL, PostgreSQL |
| Шаблонізатор | Razor Views | Django Templates / Jinja2 | Blade | Thymeleaf / Freemarker |
| Ліцензія | MIT (відкрита) | BSD (відкрита) | MIT (відкрита) | Apache 2.0 (відкрита) |
| Кросплатформеність | Windows / Linux / macOS | Windows / Linux / macOS | Windows / Linux / macOS | Windows / Linux / macOS |
| Складність входу | Середня | Низька | Низька | Висока |
| Підтримка Microsoft | Так (офіційна) | Ні | Ні | Ні |

На підставі порівняльного аналізу, результати якого систематизовано в таблиці 3.На підставі порівняльного аналізу, результати якого систематизовано в таблиці 3.2, можна зробити однозначний висновок про те, що платформа ASP.NET Core MVC є найбільш обґрунтованим і доцільним вибором для практичної реалізації системи управління персоналом класу HRMS. До ключових факторів, що визначають перевагу цього рішення, належать такі аспекти: по-перше, вбудована і технологічно зріла підсистема автентифікації ASP.NET Core Identity, яка не потребує інтеграції жодних сторонніх бібліотек і забезпечує підтримку рольової моделі розмежування доступу безпосередньо «з коробки»; по-друге, щільна нативна інтеграція з Entity Framework Core, завдяки якій істотно скорочується час,

необхідний для розробки рівня доступу до даних; по-третє, наявність вичерпної офіційної документації та гарантована довгострокова підтримка з боку компанії Microsoft, що є критично важливим чинником для корпоративних проєктів; по-четверте, високий рівень продуктивності, що є цілком порівнянним із характеристиками рішень, побудованих на основі платформи Java; і нарешті, використання мови програмування C# із суворою статичною типізацією, що дозволяє виявляти значну частину потенційних помилок ще безпосередньо на етапі компіляції, не допускаючи їх прояву у середовищі виконання — runtime.

Якщо розглядати альтернативні платформи, то Django та Laravel демонструють дещо нижчий поріг входу для розробників-початківців, однак за показниками продуктивності та цілісності екосистеми вони поступаються ASP.NET Core MVC у контексті розробки корпоративних застосунків із розгалуженою та складною бізнес-логікою. Spring Boot є цілком зіставним конкурентом з точки зору продуктивності, однак суттєво відрізняється від розглянутого рішення за рядом важливих характеристик: він відзначається вищою складністю початкової конфігурації, значно більшим обсягом шаблонного коду, необхідного для реалізації типових сценаріїв, а також відсутністю такої органічної і глибокої інтеграції між ORM-інструментом і підсистемою автентифікації, яку природним чином забезпечує зв'язка Entity Framework Core та ASP.NET Core Identity в рамках єдиної екосистеми Microsoft.

# 4 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Архітектура програмного забезпечення

Архітектура інформаційної системи управління людськими ресурсами побудована за трирівневою моделлю: рівень представлення (Presentation Layer), рівень бізнес-логіки (Business Logic Layer) та рівень доступу до даних (Data Access Layer). Такий підхід забезпечує чітке розмежування відповідальностей між компонентами системи та спрощує її подальше обслуговування і масштабування.

Рівень представлення реалізовано засобами Razor Views (.cshtml) та власними CSS-стилями без використання зовнішніх UI-фреймворків. Кожна сторінка успадковує загальний шаблон _Layout.cshtml, що містить навігаційний сайдбар, заголовок та підвал. Це дозволяє забезпечити єдиний стиль інтерфейсу для всіх сторінок.

Рівень бізнес-логіки зосереджений у контролерах ASP.NET Core MVC. Кожен контролер відповідає за певну предметну область системи: управління обліковими записами, працівниками, відділами, посадами та заявами на відпустку. Контролери отримують дані від рівня доступу до даних, виконують перевірку прав доступу та формують відповідні представлення.

Ось виправлені абзаци:  Рівень доступу до даних реалізовано через Entity Framework Core та клас HrmsDbContext. Всі звернення до бази даних виконуються асинхронно через методи ToListAsync(), FirstOrDefaultAsync() та SaveChangesAsync(), що забезпечує ефективне використання ресурсів сервера та не блокує потоки під час очікування відповіді від СУБД.  Взаємодія між рівнями відбувається за такою схемою. Браузер надсилає HTTP-запит до Middleware Pipeline ASP.NET Core, який передає його маршрутизатору. Router визначає відповідний контролер і метод, після чого контролер виконує бізнес-логіку та

звертається до HrmsDbContext. Entity Framework Core транслює LINQ-запити у SQL-команди, виконує їх та повертає результат назад до контролера. Контролер формує ViewModel на основі отриманих даних і передає його у View, де Razor-движок генерує HTML-розмітку, яку сервер повертає у відповідь браузеру.

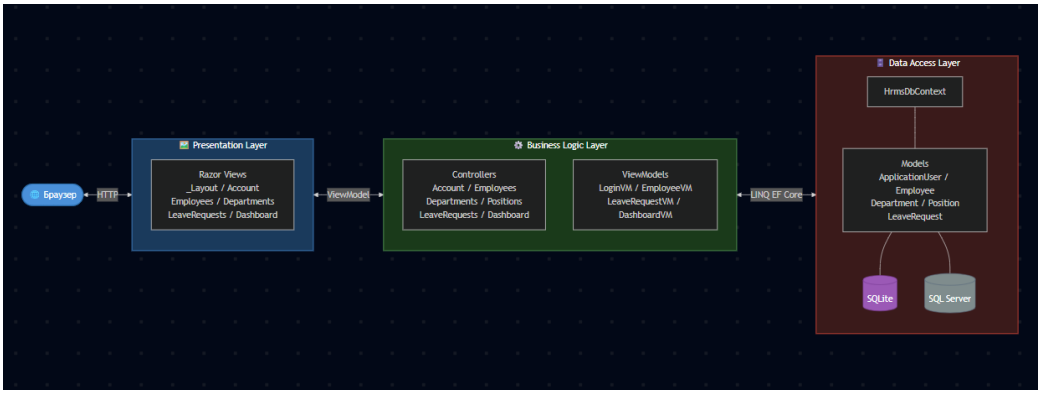



Рис. 4.1. Трирівнева архітектура HRMS

Структура проєкту організована відповідно до конвенцій ASP.NET Core MVC. У таблиці 4.1 наведено перелік контролерів із відповідними маршрутами та переліком дій.

Таблиця 4.1

Контролери системи HRMS та їх дії

| Контролер | Маршрут | Основні дії (Action-методи) |
|---|---|---|
| AccountController | /Account | Login, Register, Logout, Profile, ChangePassword |
| EmployeesController | /Employees | Index, Details, Create, Edit, Delete, Activate, Deactivate |
| DepartmentsController | /Departments | Index, Details, Create, Edit, Delete |
| PositionsController | /Positions | Index, Details, Create, Edit, Delete |
| LeaveRequestsController | /LeaveRequests | Index, Details, Create, Approve, Reject, Cancel |
| DashboardController | /Dashboard | Index (зведена статистика по ролях) |

Кожна дія контролера захищена відповідним атрибутом авторизації. HTTP GET-методи повертають відповідний View з даними, HTTP POST-методи

обробляють форми та виконують операції зі збереженням або зміною даних. Для захисту від CSRF-атак усі POST-форми використовують вбудований механізм Token Verification через AntiForgery-токени ASP.NET Core.

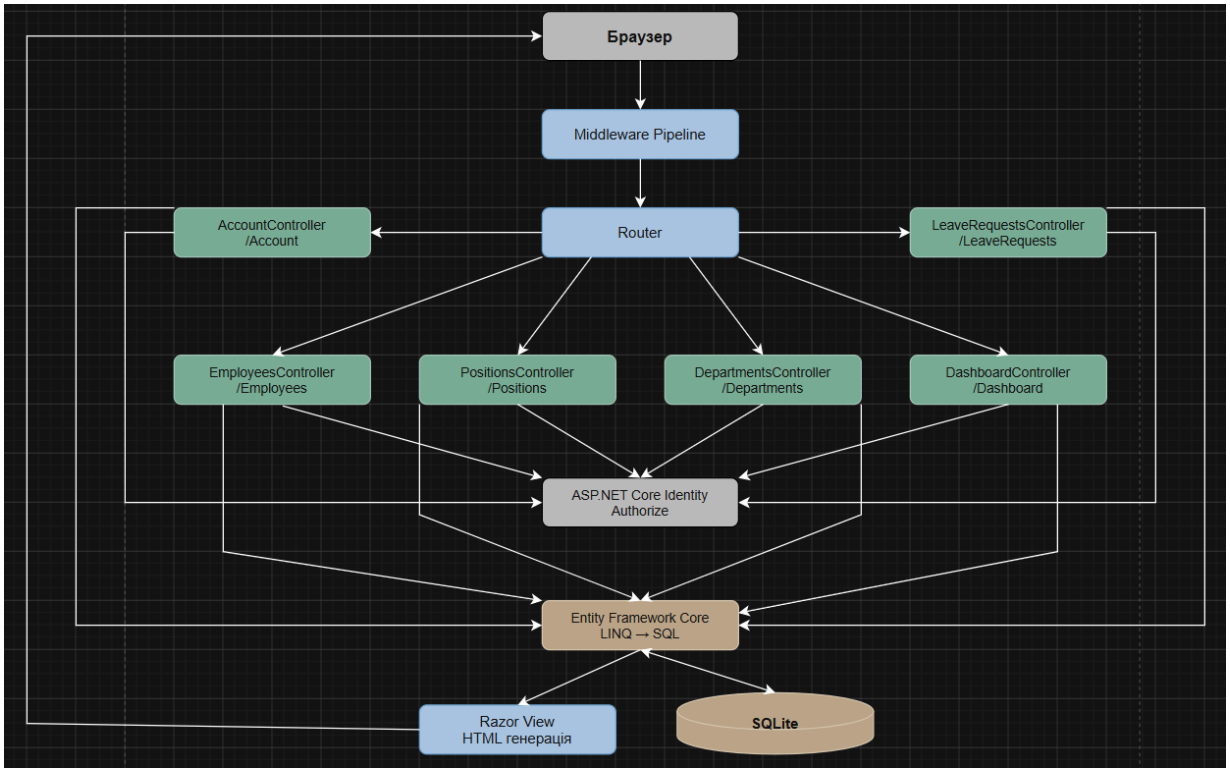



Рис. 4.2. Діаграма компонентів архітектури HRMS

## 4.2 Структури даних та бази даних

База даних HRMS спроєктована за принципами реляційної моделі та нормалізована до третьої нормальної форми (3NF). Це виключає надлишковість даних та забезпечує цілісність при виконанні операцій додавання, оновлення та видалення записів. Схема бази даних містить 7 основних таблиць, пов'язаних між собою зовнішніми ключами.

Таблиця AspNetUsers є розширенням стандартної Identity-таблиці та містить як системні поля (Id, Email, PasswordHash, SecurityStamp, ConcurrencyStamp), так і власні поля: LastName, FirstName, MiddleName (ПІБ працівника), Address (адреса), BirthDate (дата народження). Первинний ключ -рядковий GUID, що генерується автоматично при створенні.

Таблиця Employees зберігає кадрову інформацію та пов'язана з AspNetUsers відношенням один-до-одного через зовнішній ключ UserId. Додаткові поля: EmployeeNumber (унікальний табельний номер), DepartmentId (відділ), PositionId (посада), Salary (заробітна плата), EmploymentType (тип зайнятості: повна, часткова, контракт), Status (активний/неактивний), HireDate (дата прийому), ManagerId (керівник – самопосилання на Employee).

Таблиця LeaveRequests містить усі заяви на відпустку. Поле LeaveType визначає тип відпустки (щорічна, лікарняна, без збереження заробітної плати, декретна, навчальна, інше) у вигляді перечислення (enum). Поле Status відстежує стан заяви: Pending (на розгляді), Approved (схвалено), Rejected (відхилено), Cancelled (скасовано). Поле ReviewComment зберігає коментар HR при прийнятті рішення.

Таблиця 4.2

Сутності бази даних HRMS

| Сутність (таблиця) | Кількість полів | Ключові поля |
|---|---|---|
| AspNetUsers | 15 | Id, Email, PasswordHash, LastName, FirstName, MiddleName, Address, BirthDate |
| Employees | 11 | Id, EmployeeNumber, UserId, DepartmentId, PositionId, Salary, EmploymentType, Status, HireDate, ManagerId |
| Departments | 6 | Id, Name, Code, Description, ManagerId, CreatedAt |
| Positions | 7 | Id, Name, DepartmentId, Description, MinSalary, MaxSalary, CreatedAt |
| LeaveRequests | 9 | Id, EmployeeId, LeaveType, StartDate, EndDate, Reason, Status, ReviewedBy, ReviewComment |
| AspNetRoles | 3 | Id, Name, NormalizedName |
| AspNetUserRoles | 2 | UserId, RoleId |

Цілісність даних забезпечується на двох рівнях. На рівні бази даних -зовнішні ключі з каскадними правилами (RESTRICT для видалення, CASCADE для оновлення). На рівні застосунку – валідація DataAnnotations у моделях (обов'язкові поля [Required], обмеження довжини [StringLength], перевірка діапазонів [Range], формат email [EmailAddress]).

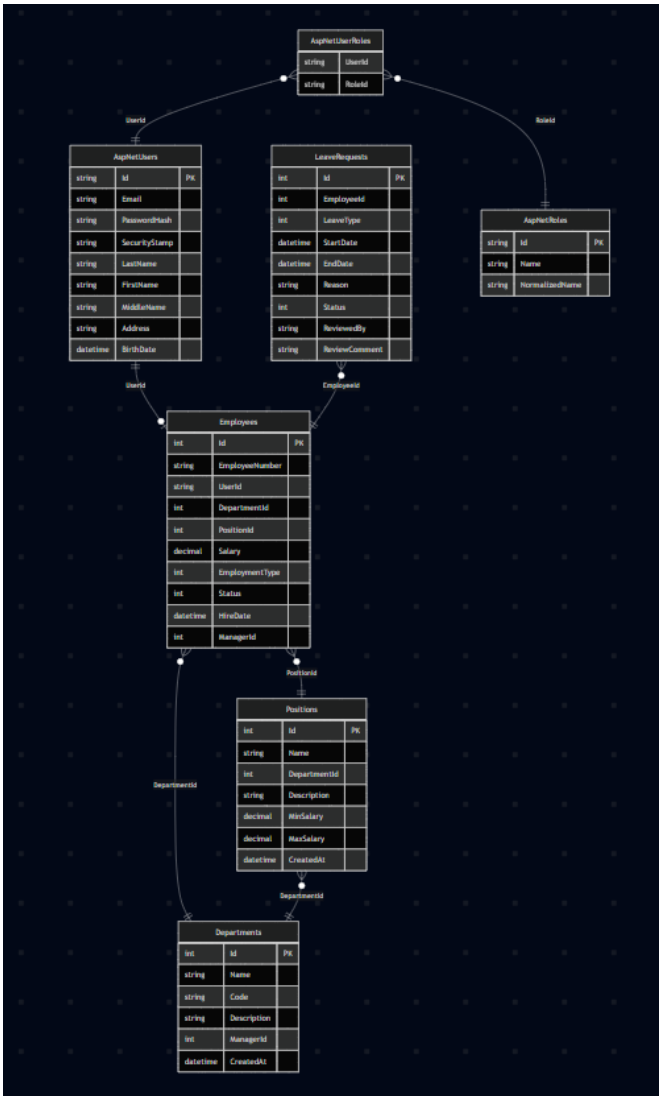



Рис. 4.3. ER-діаграма бази даних HRMS

Індексація таблиць виконана автоматично EF Core для первинних і зовнішніх ключів. Додатково створено унікальний індекс на поле EmployeeNumber для запобігання дублювання табельних номерів. Поле Email в AspNetUsers також має унікальний індекс, що забезпечується механізмом ASP.NET Core Identity.

## 4.3 Реалізація ключових функцій

Реалізація системи HRMS охоплює шість основних функціональних модулів: аутентифікація та управління сесіями, управління персоналом, організаційна структура, управління заявами на відпустку, дашборд та аналітика, профіль користувача.

Модуль аутентифікації реалізовано через AccountController. При вході система перевіряє облікові дані та встановлює автентифікаційний файл-маркер за допомогою інтегрованого сервісу «SignInManager». Під час реєстрації нового працівника відбувається створення облікового запису та призначення йому відповідної ролі засобами асинхронного менеджера користувачів «UserManager». Тимчасовий пароль відображається HR-менеджеру для передачі новому працівнику.

Модуль управління персоналом реалізовано в EmployeesController. Метод Index() формує список працівників з підтримкою фільтрації за відділом, статусом та пошуком за ім'ям через GET-параметри. Всі дані вибираються з використанням Include() для ледачого завантаження пов'язаних сутностей Department та Position. Метод Create() реалізує двоетапне збереження: спочатку створюється запис ApplicationUser через UserManager, потім пов'язаний запис Employee через DbContext.

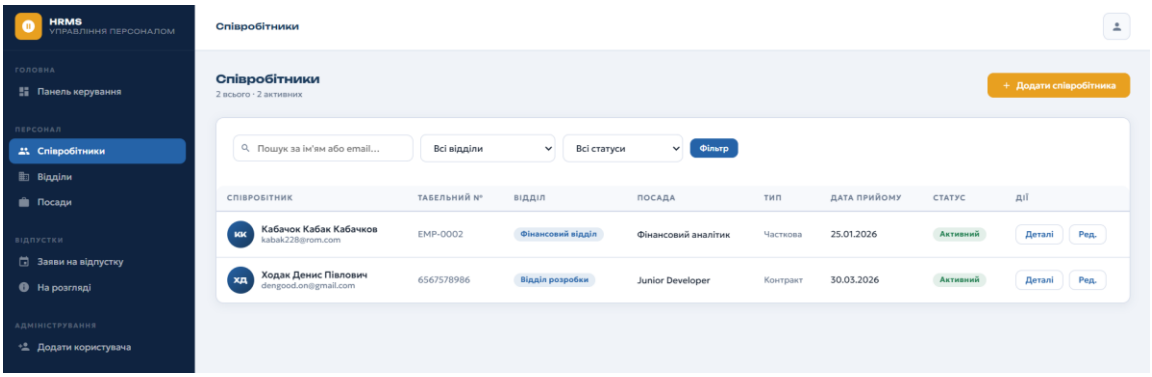



Рис. 4.4. Екран списку працівників із фільтрацією

Модуль управління заявами на відпустку реалізовано в LeaveRequestsController. Метод Create() доступний лише для ролі Employee – працівник вибирає тип відпустки, вказує дати початку та кінця, причину. Кількість днів розраховується автоматично через JavaScript на стороні клієнта. Після збереження заявка отримує статус Pending. Методи Approve() та Reject() доступні HR та Admin – вони змінюють статус заявки, записують ім'я рецензента (ReviewedBy) та необов'язковий коментар.

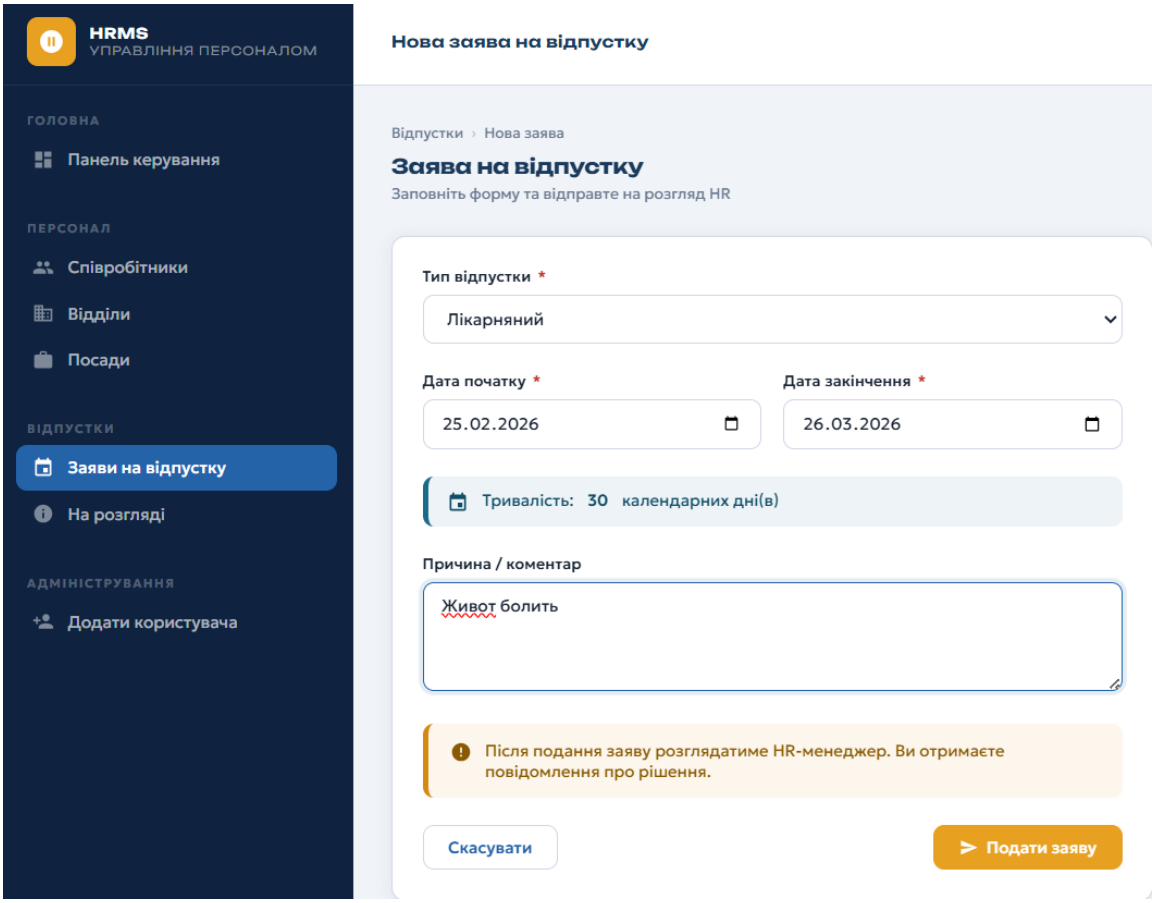



Рис. 4.5. Форма подачі заяви на відпустку

Дашборд реалізовано в DashboardController. Залежно від ролі поточного користувача формується різна статистика: для Admin та HR – загальна кількість працівників, розподіл по відділах, кількість активних/неактивних, нові працівники за місяць, відкриті заявки на відпустку; для Manager – кількість підлеглих, їх заявки; для Employee -статус власних заявок та наступна відпустка. Дані

агрегуються LINQ-запитами з підрахунком (Count()), групуванням (GroupBy()) та сортуванням.

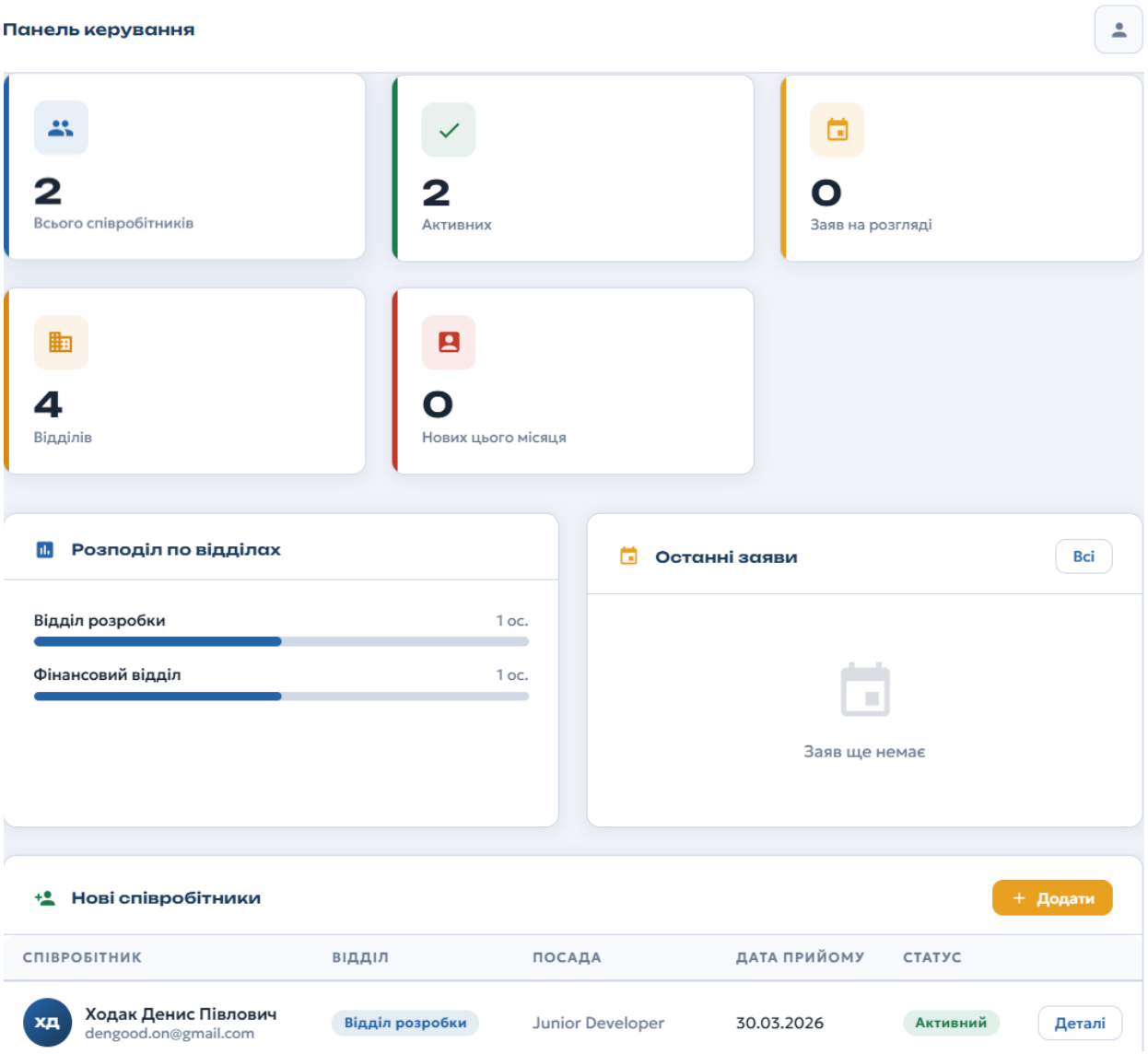



Рис. 4.6. Головна панель (Дашборд) для ролі HR

## 4.4 Рольова модель доступу

Система розмежування доступу в HRMS реалізована на основі RBAC (Role-Based Access Control) – рольового керування доступом. Цей підхід визначає права не для окремих користувачів, а для ролей, що спрощує адміністрування системи при масштабуванні [14].

В системі визначено чотири ролі, кожна з яких охоплює певне коло функцій. Роль Administrator призначена системному адміністратору та надає повний доступ до всіх функцій системи, включаючи видалення записів, управління ролями та перегляд системних журналів. Роль HR (Human Resources Manager) призначена для кадрового відділу та надає доступ до управління персоналом, відділами, посадами та розгляду заяв, але не дозволяє видаляти критичні записи. Роль Manager призначена для керівників відділів і обмежує перегляд лише своїми підлеглими. Роль Employee є базовою роллю для всіх працівників.

Таблиця 4.3

Матриця прав доступу HRMS

| Функціонал / Дія | Admin | HR | Manager | Employee |
|---|---|---|---|---|
| Перегляд списку всіх працівників | + | + | +- | - |
| Додавання нового працівника | + | + | - | - |
| Редагування даних працівника | + | + | - | - |
| Видалення / деактивація працівника | + | - | - | - |
| Управління відділами | + | + | - | - |
| Управління посадами | + | + | - | - |
| Подача заяви на відпустку | - | - | - | + |
| Розгляд заяв на відпустку (схвалення/відхилення) | + | + | - | - |
| Перегляд власного профілю | + | + | + | + |
| Редагування власного профілю | + | + | + | + |
| Доступ до адміністративної панелі | + | - | - | - |
| Дашборд із статистикою | + | + | + | + |

Технічна реалізація RBAC здійснюється на двох рівнях. На рівні контролерів атрибут [Authorize(Roles = "Admin")] забороняє доступ до всього контролера або окремих методів для користувачів без відповідної ролі. При спробі доступу без прав система перенаправляє на сторінку входу або повертає статус 403 Forbidden. На рівні користувацького інтерфейсу динамічне відображення елементів реалізовано через умовні конструкції шаблонізатора «Razor», які перевіряють наявність відповідної ролі у поточній сесії користувача.

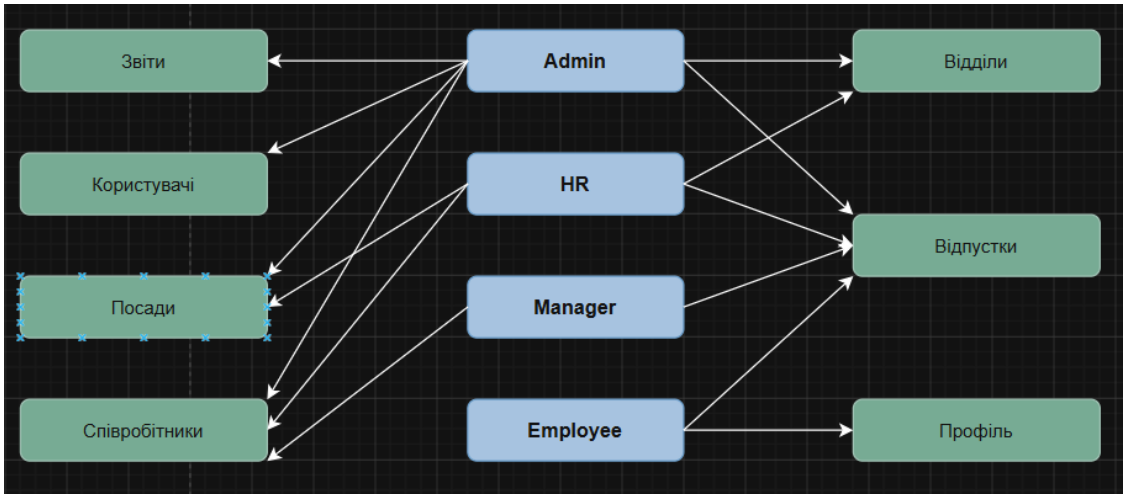



Рис. 4.7. Схема рольового доступу RBAC в системі HRMS

Призначення ролі відбувається при реєстрації нового працівника HR-менеджером. За замовчуванням кожен новий користувач отримує роль Employee. Якщо працівник є керівником відділу, йому призначається роль Manager. Роль HR та Admin призначаються лише адміністратором системи через адмін-панель. Один користувач може мати лише одну роль, що спрощує логіку перевірки прав і унеможливлює конфлікти дозволів.

## 4.5 Інтерфейс користувача

Інтерфейс системи HRMS розроблено без використання зовнішніх CSS-фреймворків (Bootstrap, Tailwind тощо) – повністю власна дизайн-система на базі чистого CSS3. Це дозволило повністю контролювати зовнішній вигляд та уникнути

зайвих залежностей. Дизайн орієнтований на корпоративний стиль: стримана колірна палітра, чіткі типографічні правила, мінімалістичний інтерфейс [15].

Таблиця 4.4

Елементи інтерфейсу та їх реалізація

| Елемент інтерфейсу | Опис реалізації | Технічне рішення |
|---|---|---|
| Навігаційний сайдбар | Фіксований бічний список з пунктами меню, активний пункт підсвічується | CSS position:fixed, класи active/inactive |
| Адаптивна верстка | Коректне відображення на десктопі та планшеті | CSS media queries, flexbox |
| Форми з валідацією | Серверна валідація DataAnnotations, підсвічування помилок | ModelState, asp-validation-for Tag Helpers |
| Таблиці з фільтрацією | Фільтр за відділом, статусом, пошук за ім'ям | GET-параметри, LINQ Where() |
| Статусні бейджі | Кольорові позначки статусу (активний/неактивний, схвалено/відхилено) | CSS-класи status-active, status-pending тощо |
| Сповіщення (Alert) | Повідомлення про успіх/помилку після дії | TempData, @if(TempData[...]) |
| Модальні підтвердження | Підтвердження видалення перед виконанням | JavaScript confirm() |

Шрифтова система використовує дві гарнітури: Unbounded – для заголовків та назви системи в сайдбарі, Geologica – для основного тексту, Fпараграфів та елементів форм. Обидва шрифти підключені через Google Fonts і відносяться до

категорії гротеск, що забезпечує їх читабельність на екранах різних розмірів та роздільної здатності.

Навігаційний сайдбар закріплено по лівому краю екрана за допомогою властивості position: fixed і містить кілька ключових елементів інтерфейсу: логотип системи, повне ім'я поточного авторизованого користувача, перелік пунктів меню, що формується відповідно до призначеної ролі, а також кнопку для виходу з облікового запису. Активний пункт меню візуально виділяється серед інших завдяки кольоровому лівому бордеру та зміні кольору фону відповідного елемента, що дозволяє користувачу одразу орієнтуватися у поточному розділі системи. На пристроях із розміром екрана, характерним для планшетів, сайдбар автоматично згортається у компактний вигляд – це реалізовано засобами адаптивної верстки через механізм CSS media queries, який відстежує ширину вікна перегляду та застосовує відповідні стил

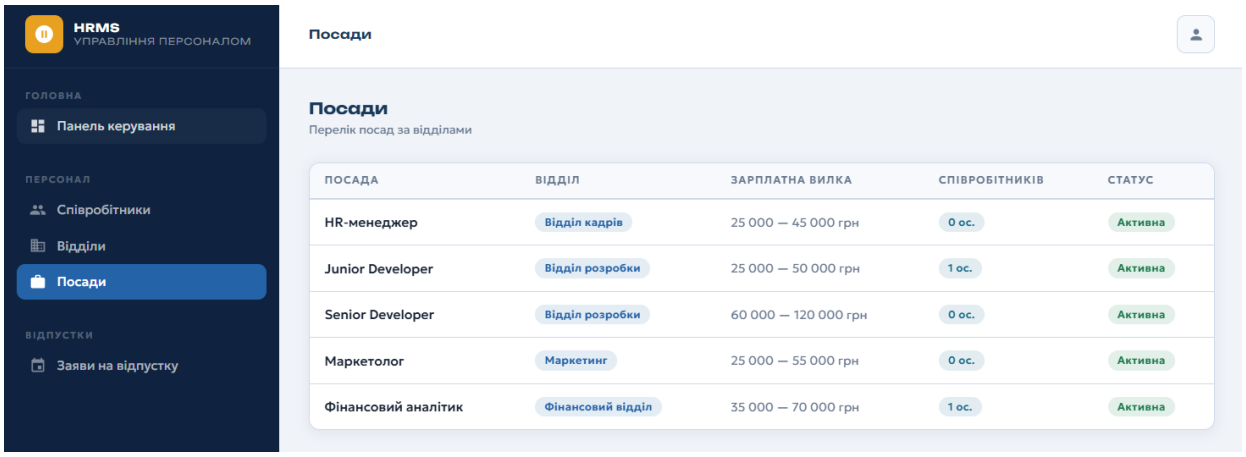



Рис. 4.8. Загальний вигляд інтерфейсу HRMS

Форми системи реалізовані з використанням Razor Tag Helpers (asp-for, asp-action, asp-controller, asp-validation-for), що забезпечує двосторонню прив'язку між полями форми та ViewModel. Валідація відбувається на стороні сервера через ModelState.IsValid. Помилки заповнення відображаються безпосередньо під відповідним полем форми у червоному кольорі.

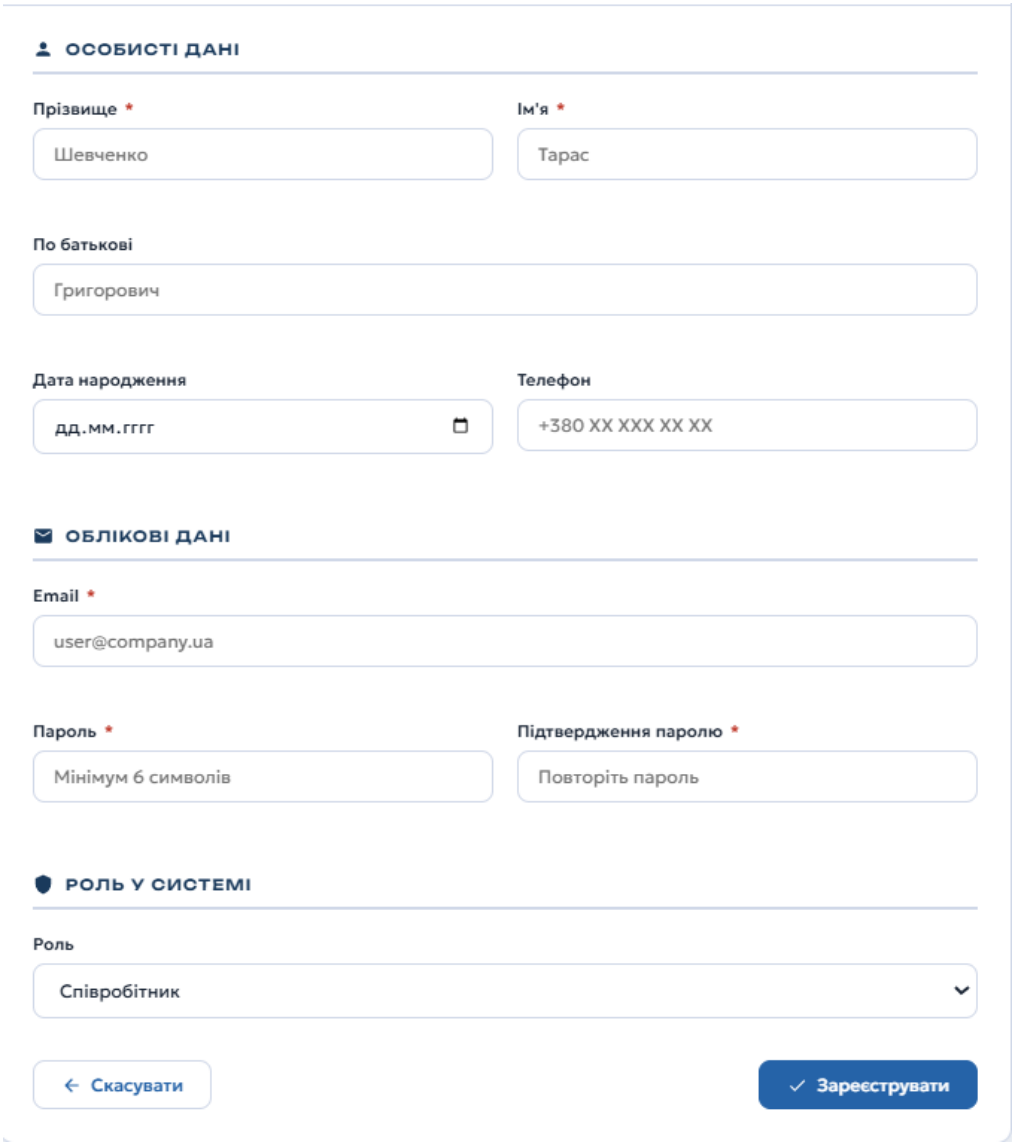



Рис. 4.9. Форма редагування профілю працівника

Повідомлення про результат виконання операцій реалізовані через TempData – механізм передачі одноразових повідомлень між редиректами. Після успішного збереження, видалення або розгляду заявки користувач бачить зелений блок-сповіщення з текстом результату. При помилці відображається червоний блок. Повідомлення автоматично зникають при наступному переході на іншу сторінку.

# ВИСНОВКИ

У кваліфікаційній роботі розроблено інформаційну систему управління людськими ресурсами на базі ASP.NET Core MVC, що забезпечує комплексну автоматизацію ключових HR-процесів для організацій різного масштабу.

1. Аналіз предметної області виявив, що більшість малих і середніх організацій досі використовує застарілі засоби кадрового обліку, тоді як впровадження комерційних рішень є економічно недоцільним. Це обґрунтувало необхідність розробки власної системи з балансом між функціональністю та простотою розгортання.
2. На основі сформульованих вимог визначено чотири ролі користувачів і спроєктовано трирівневу архітектуру системи. Обґрунтовано вибір стеку ASP.NET Core 8 MVC, Entity Framework Core та ASP.NET Core Identity як технологічної основи, що забезпечує надійність, масштабованість і зручність супроводу.
3. Реалізовано механізм рольового управління доступом (RBAC), який забезпечує чітке розмежування прав між адміністраторами, HR-менеджерами, керівниками підрозділів та рядовими працівниками в межах єдиної кодової бази. Це є ключовою відмінністю розробленого рішення від типових систем початкового рівня.
4. Розроблено повнофункціональні модулі обліку працівників, управління організаційною структурою та обробки заяв на відпустку – від подачі працівником до затвердження керівником. Усі модулі пройшли функціональне тестування на реальних сценаріях використання.

Практичне значення роботи полягає у готовому програмному рішенні, яке може бути розгорнуто в організаціях середнього розміру для автоматизації кадрового обліку та управління персоналом без придбання дорогого комерційного програмного забезпече.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Ходак Д.П., Чернявський Ж.А. Аналіз методів та технологій для розробки інформаційних систем управління людськими ресурсами // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в CRUDінформаційно-комунікаційних технологіях». Збірник тез. 23.04.2026, ДУІКТ, Київ, Україна. – подана до друку

2. Ходак Д.П., Чернявський Ж.А. Проектування та реалізація інформаційної системи управління людськими ресурсами на базі ASP.NET Core MVC «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 1-3.05.2026, ДУІКТ, Київ, Україна. – подана до друку.

3. Redirecting. *Home Page*. URL: https://doi.org/10.1016/j.hrmr.2006.11.002 (дата звернення: 26.04.2026).

4. SAP SuccessFactors Human Experience Management Suite. SAP. URL: https://www.sap.com/products/hcm.html(дата звернення: 26.04.2026).

5. BambooHR: The Complete HR Software for People, Payroll & Benefits [Електронний ресурс]. -Режим доступу: https://www.bamboohr.com/ (дата звернення: 26.04.2026).

6. OrangeHRM: All in One HR Software for Businesses [Електронний ресурс]. -Режим доступу: https://orangehrm.com/ (дата звернення: 26.04.2026).

7. Cloudbasierte HR-Software | HRMS-LÃ¶sung | Zoho People. *Zoho*. URL: https://www.zoho.com/people/ (дата звернення: 26.04.2026).

8. OWASP Top Ten Web Application Security Risks [Електронний ресурс] / OWASP Foundation. -Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення: 26.04.2026).

9. SP 800-132, Recommendation for Password-Based Key Derivation: Part 1: Storage Applications | CSRC. *NIST Computer Security Resource Center | CSRC*. URL: https://csrc.nist.gov/pubs/sp/800/132/final (дата звернення: 26.04.2026).

10. Overview of ASP.NET Core MVC. *Microsoft Learn: Build with answers in reach*. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-10.0 (дата звернення: 26.04.2026).

11. Introduction to Identity on ASP.NET Core. *Microsoft Learn: Build with answers in reach*. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-10.0&tabs=visual-studio (дата звернення: 26.04.2026).
12. Overview of Entity Framework Core - EF Core. *Microsoft Learn: Build with answers in reach*. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 26.04.2026).
13. SQLite Documentation. *SQLite Home Page*. URL: https://www.sqlite.org/docs.html (дата звернення: 26.04.2026).
14. Role Based Access Control | CSRC. *NIST Computer Security Resource Center | CSRC*. URL: https://csrc.nist.gov/projects/role-based-access-control (дата звернення: 26.04.2026).
15. Nielsen J. 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group*. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 26.04.2026).

# ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ


Інформаційна система управління людськими ресурсами на базі технологій ASP.NET Core MVC

Виконав студент 4 курсу
групи ТЦР-43
Ходак Денис Павлович
Керівник роботи
Старший викладач Чернявський Ждан Анатолійович

Київ – 2026

## МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Мета роботи**: Покращення процесів управління людськими ресурсами через розробку web-орієнтованої інформаційної системи з автоматизацією HR-операцій та рольовим доступом

**Об'єкт дослідження**: Процес автоматизації управління людськими ресурсами в організації.

**Предмет дослідження**: Інформаційна система управління людськими ресурсами, розроблена на базі технологій ASP.NET Core MVC.

# ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Огляд та аналіз предметної галузі HRMS, дослідження існуючих програмних аналогів.
2. Проектування функціональних та нефункціональних вимог до програмного забезпечення.
3. Огляд та порівняння засобів реалізації: ASP.NET Core MVC, Identity, Entity Framework Core.
4. Проектування архітектури системи та структури бази даних.
5. Програмна реалізація модулів та тестування застосунку.



# АНАЛІЗ АНАЛОГІВ

| Параметр | SAP SF | BambooHR | Zoho People | Наш застосунок |
|---|---|---|---|---|
| Рольова модель | + | + | + | **+ (4 ролі RBAC)** |
| Відкритий код | – | – | – | **+ (ASP.NET)** |
| Малий бізнес | – | + | + | **+** |
| Локальне розгорт. | – | – | – | **+** |
| Вартість | Висока | Середня | Середня | **Безкошт.** |

## ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### Функціональні вимоги:

1. Облік співробітників (CRUD, пошук, фільтрація)
2. Управління відділами та посадами
3. Подання та обробка заяв на відпустку
4. Рольова модель доступу (4 рівні)
5. Автентифікація через ASP.NET Core Identity
6. Аудит змін у системі

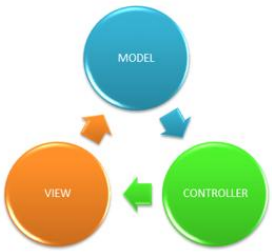
### Нефункціональні вимоги:

1. Зручність використання (Usability)
2. Продуктивність (Performance)
3. Надійність (Reliability)
4. Масштабованість (Scalability)
5. Безпека (Security - CSRF, XSS)



## ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



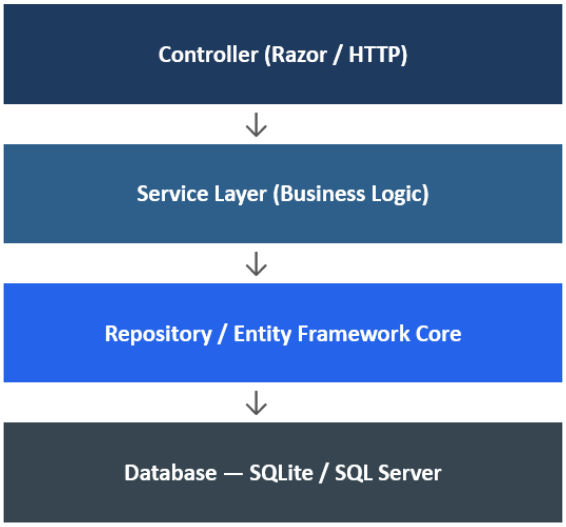





## Діаграма варіантів використання




# АРХІТЕКТУРА СИСТЕМИ ТА СТРУКТУРА ДАНИХ




**Ключові сутності БД**

**ApplicationUser**

Id, UserName, Email, DepartmentId, PositionId

**Department**

Id, Name, ManagerId

**Position**

Id, Title, DepartmentId

**LeaveRequest**

Id, UserId, StartDate, EndDate, Status

**AuditLog**

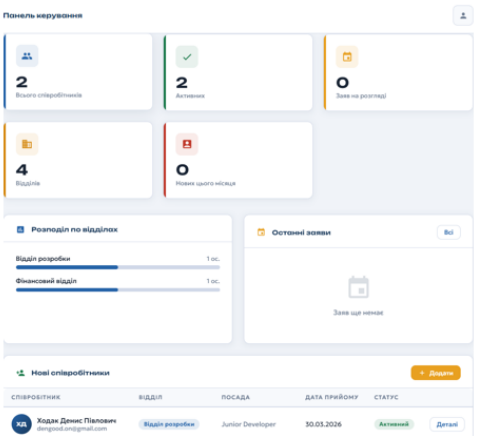
Id, UserId, Action, Entity, Timestamp

# РОЛЬОВА МОДЕЛЬ ДОСТУПУ (RBAC)

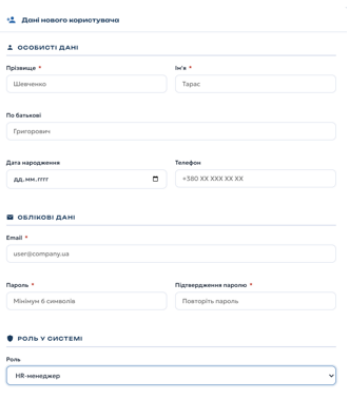
| Адміністратор | HR-менеджер | Менеджер | Співробітник |
|---|---|---|---|
| • Управління користувачами та ролями<br>• Видалення відділів і посад<br>• Повний доступ до всіх модулів<br>• Перегляд журналу аудиту | • CRUD для співробітників<br>• Управління відділами/посадами<br>• Розгляд заяв на відпустку<br>• Перегляд усіх даних | • Перегляд співробітників відділу<br>• Перегляд структури організації<br>• Розгляд заяв підлеглих | • Перегляд власного профілю<br>• Подання заяв на відпустку<br>• Перегляд статусу заявок |



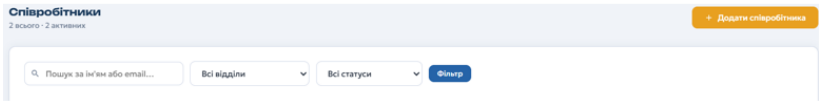
## Екранні форми




Головний дашборд з панелью керування




Форма додавання нового користувача




Поле пошуку та фільтрації співробітників

# АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Ходак Д.П., Чернявський Ж. Аналіз методів та технологій для розробки інформаційних систем управління людськими ресурсами. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», ДУІКТ, Київ, Україна. Подано до друку.

2. Ходак Д.П., Чернявський Ж. Проектування та реалізація інформаційної системи управління людськими ресурсами на базі ASP.NET Core MVC. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», ДУІКТ, Київ, Україна. Подано до друку.



# ВИСНОВКИ

1. Проаналізовано предметну галузь HRMS та досліджено аналоги SAP SuccessFactors, BambooHR, Zoho People; визначено вимоги до системи.
2. Обрано та обґрунтовано технологічний стек: ASP.NET Core MVC, Identity, Entity Framework Core, SQLite/SQL Server.
3. Спроектовано архітектуру за патерном MVC та структуру реляційної БД із 5 ключовими сутностями.
4. Реалізовано модулі обліку персоналу, управління відділами, заяви на відпустку та RBAC з 4 рівнями.
5. Проведено функціональне тестування - всі тест-кейси пройдено успішно.

# ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using HRSystem.Models;
using HRSystem.ViewModels;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace HRSystem.Controllers
{
   public class AccountController : Controller
   {
      private readonly UserManager<ApplicationUser>
_userManager;
      private readonly SignInManager<ApplicationUser>
_signInManager;
      private readonly RoleManager<IdentityRole>
_roleManager;

      public AccountController(
         UserManager<ApplicationUser> userManager,
         SignInManager<ApplicationUser> signInManager,
         RoleManager<IdentityRole> roleManager)
      {
         _userManager = userManager;
         _signInManager = signInManager;
         _roleManager = roleManager;
      }

      [HttpGet]
      public IActionResult Login(string? returnUrl = null)
      {
         ViewData["ReturnUrl"] = returnUrl;
         return View();
      }

      [HttpPost]
      [ValidateAntiForgeryToken]
      public async Task<IActionResult>
Login(LoginViewModel model, string? returnUrl = null)
      {
         ViewData["ReturnUrl"] = returnUrl;
         if (!ModelState.IsValid) return View(model);

         var result = await
_signInManager.PasswordSignInAsync(
            model.Email, model.Password, model.RememberMe,
lockoutOnFailure: true);

         if (result.Succeeded)
            return LocalRedirect(returnUrl ?? "/Dashboard");

         if (result.IsLockedOut)
         {
            ModelState.AddModelError("", "Обліковий запис
заблоковано. Спробуйте через 5 хвилин.");
            return View(model);
         }

         ModelState.AddModelError("", "Невірний email або
пароль.");
         return View(model);
      }

      [HttpGet]
      [Authorize(Roles = "Admin,HR")]
      public IActionResult Register()
      {
         return View(new RegisterViewModel());
      }

      [HttpPost]
      [Authorize(Roles = "Admin,HR")]
      [ValidateAntiForgeryToken]
      public async Task<IActionResult>
Register(RegisterViewModel model)
      {
         if (!ModelState.IsValid) return View(model);

         var existingUser = await
_userManager.FindByEmailAsync(model.Email);
         if (existingUser != null)
         {
            ModelState.AddModelError("Email", "Користувач з
таким email вже існує.");
            return View(model);
         }

         var user = new ApplicationUser
         {
            UserName = model.Email,
            Email = model.Email,
            FirstName = model.FirstName,
            LastName = model.LastName,
            MiddleName = model.MiddleName,
            PhoneNumber = model.PhoneNumber,
            DateOfBirth = model.DateOfBirth,
            EmailConfirmed = true
         };

         var result = await _userManager.CreateAsync(user,
model.Password);
         if (result.Succeeded)
         {
            // Ensure role exists
            if (!await
_roleManager.RoleExistsAsync(model.Role))
               await _roleManager.CreateAsync(new
IdentityRole(model.Role));

            await _userManager.AddToRoleAsync(user,
model.Role);

            TempData["Success"] = $"Користувача
{user.FullName} успішно зареєстровано.";
            return RedirectToAction("Index", "Employees");
         }

         foreach (var error in result.Errors)
            ModelState.AddModelError("", error.Description);

         return View(model);
      }

      [Authorize]
      [HttpPost]
      [ValidateAntiForgeryToken]
      public async Task<IActionResult> Logout()
      {
         await _signInManager.SignOutAsync();
         return RedirectToAction("Login");
      }

      [Authorize]
      [HttpGet]
      public async Task<IActionResult> Profile()
      {
         var user = await _userManager.GetUserAsync(User);
```

```
        if (user == null) return NotFound();

        var model = new EditProfileViewModel
        {
            Id = user.Id,
            FirstName = user.FirstName,
            LastName = user.LastName,
            MiddleName = user.MiddleName,
            Email = user.Email ?? "",
            PhoneNumber = user.PhoneNumber,
            DateOfBirth = user.DateOfBirth,
            Address = user.Address
        };
        return View(model);
    }

    [Authorize]
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Profile(EditProfileViewModel model)
    {
        if (!ModelState.IsValid) return View(model);

        var user = await _userManager.GetUserAsync(User);
        if (user == null) return NotFound();

        user.FirstName = model.FirstName;
        user.LastName = model.LastName;
        user.MiddleName = model.MiddleName;
        user.PhoneNumber = model.PhoneNumber;
        user.DateOfBirth = model.DateOfBirth;
        user.Address = model.Address;

        var result = await _userManager.UpdateAsync(user);
        if (result.Succeeded)
        {
            TempData["Success"] = "Профіль успішно
оновлено.";
            return RedirectToAction("Profile");
        }

        foreach (var error in result.Errors)
            ModelState.AddModelError("", error.Description);

        return View(model);
    }

    [Authorize]
    [HttpGet]
    public IActionResult ChangePassword()
    {
        return View(new ChangePasswordViewModel());
    }

    [Authorize]
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
ChangePassword(ChangePasswordViewModel model)
    {
        if (!ModelState.IsValid) return View(model);

        var user = await _userManager.GetUserAsync(User);
        if (user == null) return NotFound();

        var result = await
_userManager.ChangePasswordAsync(user,
model.CurrentPassword, model.NewPassword);
        if (result.Succeeded)
        {
            await _signInManager.RefreshSignInAsync(user);
            TempData["Success"] = "Пароль успішно
змінено.";
            return RedirectToAction("Profile");
        }

        foreach (var error in result.Errors)
            ModelState.AddModelError("", error.Description);

        return View(model);
    }

    public IActionResult AccessDenied()
    {
        return View();
    }
  }
}
using HRSystem.Data;
using HRSystem.Models;
using HRSystem.ViewModels;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace HRSystem.Controllers
{
  [Authorize]
  public class DashboardController : Controller
  {
    private readonly ApplicationDbContext _context;

    public DashboardController(ApplicationDbContext
context)
    {
      _context = context;
    }

    public async Task<IActionResult> Index()
    {
      var now = DateTime.Now;
      var startOfMonth = new DateTime(now.Year,
now.Month, 1);

      var employees = await _context.Employees
        .Include(e => e.Department)
        .Include(e => e.Position)
        .Include(e => e.User)
        .ToListAsync();

      var leaveRequests = await _context.LeaveRequests
        .Include(l => l.Employee).ThenInclude(e => e!.User)
        .Include(l => l.Employee).ThenInclude(e =>
e!.Department)
        .OrderByDescending(l => l.CreatedAt)
        .Take(10)
        .ToListAsync();

      var byDept = employees
        .Where(e => e.Department != null)
        .GroupBy(e => e.Department!.Name)
        .ToDictionary(g => g.Key, g => g.Count());

      var leavesByType = await _context.LeaveRequests
        .Where(l => l.Status == LeaveStatus.Approved)
        .GroupBy(l => l.LeaveType)
        .Select(g => new { Type = g.Key.ToString(), Count
= g.Count() })
        .ToDictionaryAsync(g => g.Type, g => g.Count);
```

```
            var vm = new DashboardViewModel
            {
                TotalEmployees = employees.Count,
                ActiveEmployees = employees.Count(e => e.Status
== EmployeeStatus.Active),
                Departments = await
_context.Departments.CountAsync(d => d.IsActive),
                PendingLeaveRequests = await
_context.LeaveRequests.CountAsync(l => l.Status ==
LeaveStatus.Pending),
                NewHiresThisMonth = employees.Count(e =>
e.HireDate >= startOfMonth),
                RecentLeaveRequests = leaveRequests,
                RecentEmployees =
employees.OrderByDescending(e => e.HireDate).Take(5),
                EmployeesByDepartment = byDept,
                LeavesByType = leavesByType
            };

            return View(vm);
        }
    }
}
using HRSystem.Data;
using HRSystem.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace HRSystem.Controllers
{
    [Authorize]
    public class DepartmentsController : Controller
    {
        private readonly ApplicationDbContext _context;

        public DepartmentsController(ApplicationDbContext
context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            var departments = await _context.Departments
                .Include(d => d.HeadEmployee).ThenInclude(e =>
e!.User)
                .Include(d => d.Employees)
                .Include(d => d.Positions)
                .ToListAsync();
            return View(departments);
        }

        public async Task<IActionResult> Details(int id)
        {
            var dept = await _context.Departments
                .Include(d => d.HeadEmployee).ThenInclude(e =>
e!.User)
                .Include(d => d.Employees).ThenInclude(e =>
e.User)
                .Include(d => d.Employees).ThenInclude(e =>
e.Position)
                .Include(d => d.Positions)
                .FirstOrDefaultAsync(d => d.Id == id);

            if (dept == null) return NotFound();
            return View(dept);
        }

        [Authorize(Roles = "Admin,HR")]
        public async Task<IActionResult> Create()
        {
            await PopulateManagers();
            return View(new Department());
        }

        [HttpPost]
        [Authorize(Roles = "Admin,HR")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(Department
model)
        {
            ModelState.Remove("HeadEmployee");
            ModelState.Remove("Employees");
            ModelState.Remove("Positions");

            if (!ModelState.IsValid)
            {
                await PopulateManagers();
                return View(model);
            }

            _context.Departments.Add(model);
            await _context.SaveChangesAsync();
            TempData["Success"] = $"Відділ «{model.Name}»
успішно створено.";
            return RedirectToAction(nameof(Index));
        }

        [Authorize(Roles = "Admin,HR")]
        public async Task<IActionResult> Edit(int id)
        {
            var dept = await _context.Departments.FindAsync(id);
            if (dept == null) return NotFound();
            await PopulateManagers(dept.HeadEmployeeId);
            return View(dept);
        }

        [HttpPost]
        [Authorize(Roles = "Admin,HR")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, Department
model)
        {
            if (id != model.Id) return BadRequest();
            ModelState.Remove("HeadEmployee");
            ModelState.Remove("Employees");
            ModelState.Remove("Positions");

            if (!ModelState.IsValid)
            {
                await PopulateManagers(model.HeadEmployeeId);
                return View(model);
            }

            _context.Update(model);
            await _context.SaveChangesAsync();
            TempData["Success"] = "Відділ оновлено.";
            return RedirectToAction(nameof(Index));
        }

        [Authorize(Roles = "Admin")]
        public async Task<IActionResult> Delete(int id)
        {
            var dept = await _context.Departments
                .Include(d => d.Employees)
                .FirstOrDefaultAsync(d => d.Id == id);
            if (dept == null) return NotFound();
```

```
            return View(dept);
        }

        [HttpPost, ActionName("Delete")]
        [Authorize(Roles = "Admin")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int
id)
        {
            var dept = await _context.Departments.Include(d =>
d.Employees).FirstOrDefaultAsync(d => d.Id == id);
            if (dept == null) return NotFound();

            if (dept.Employees.Any())
            {
                TempData["Error"] = "Неможливо видалити відділ,
в якому є працівники.";
                return RedirectToAction(nameof(Index));
            }

            _context.Departments.Remove(dept);
            await _context.SaveChangesAsync();
            TempData["Success"] = "Відділ видалено.";
            return RedirectToAction(nameof(Index));
        }

        private async Task PopulateManagers(int? selectedId =
null)
        {
            var managers = await _context.Employees
                .Include(e => e.User)
                .Where(e => e.Status == EmployeeStatus.Active)
                .ToListAsync();

            ViewBag.Managers = new SelectList(
                managers.Select(e => new { e.Id, Name =
e.User!.FullName }),
                "Id", "Name", selectedId);
        }
    }
}
```