

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО

«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД»

МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ

Факультет комп'ютерно-інформаційних технологій та дизайну

Кафедра комп'ютерних інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Тема: «Розробка програмного забезпечення для відділу кадрів»

Виконав: студент групи ІК-9-22-Б1ПЗ (4.0ддс)

Хоменко Р. М.

Керівник: д.т.н., професор

Дуднік А.С.

Рецензент:

Допущено до захисту

Завідувач кафедри _____ д.е.н., професор Кавун С.В.

(підпис)

Київ 2026

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 95 с., 38 рис., 1 табл., 3 додатки, 32 джерела.

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВІДДІЛ КАДРІВ, ENTITY FRAMEWORK CORE, BPM, CAMUNDA MODELER, ASP.NET CORE, SQL SERVER, DINKTOPDF, QRCODER.

Об'єктом дослідження роботи є процеси управління персоналом, що допомагають здійснювати адміністрування роботи відділу.

Метою роботи є проектування та розробка інформаційної системи для автоматизації процесів відділу кадрів завдяки екосистемі .NET з інтеграцією BPM-платформи Camunda.

Методи розробки базуються на .NET 8 з використанням BPM-платформи Camunda та бази даних MS SQL Server.

Результатом роботи є вебзастосунок, який переводить кадрові процеси в цифровий формат. Втілений функціонал охоплює повний операційний цикл — від обліку персоналу до погодження запитів на відпустки.

SOFTWARE DEVELOPMENT, HR DEPARTMENT, ENTITY FRAMEWORK CORE, BPM, CAMUNDA MODELER, ASP.NET CORE, SQL SERVER, DINKTOPDF, QRCODER.

The object of research of the work is personnel management processes that help to carry out the administration of the department's work.

The goal of the work is the design and development of an information system for the automation of HR department processes thanks to the .NET ecosystem with the integration of the Camunda BPM platform.

Development methods are based on .NET 8 with the use of the Camunda BPM platform and the MS SQL Server database.

The result of the work is a web application that transfers HR processes into a digital format. The implemented functionality covers the full operational cycle — from personnel accounting to the approval of requests for leaves.

ЗМІСТ

Список умовних скорочень	5
ВСТУП	7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СПЕЦИФІКИ HR-ПРОЦЕСІВ	9
1.1. Дослідження особливостей організації бізнес-процесів	9
1.2. Аналіз предметної області та життєвого циклу заяви	11
1.3. Формування вимог до програмного продукту	14
1.4. Висновки до першого розділу	15
2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ	17
2.1. Порівняльний аналіз програмних рішень-аналогів	17
2.2. Моделювання функціональних можливостей системи	19
2.3. Проєктування архітектури та інформаційного забезпечення	23
2.4. Обґрунтування вибору технологічного стеку та засобів розробки	27
2.5. Висновки до другого розділу	31
3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	32
3.1. Архітектура програмної реалізації	32
3.2. Реалізація бізнес-процесу засобами Camunda Modeler	36
3.3. Реалізація функціональних модулів системи	39
3.4. Реалізація бази даних та Entity Framework	43
3.4.3. Інтерфейсна реалізація маніпуляції даними	47
3.5. Перевірка та оцінка якості системи	53
3.6. Висновки до третього розділу	58
ВИСНОВКИ	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62

Список умовних скорочень

ACID (Atomicity, Consistency, Isolation, Durability) — набір властивостей, що гарантують надійність транзакцій бази даних.

API (Application Programming Interface) — програмний інтерфейс додатку для взаємодії компонентів.

BPM (Business Process Management) — методологія управління бізнес-процесами.

BPMN (Business Process Model and Notation) — стандарт моделювання та нотація бізнес-процесів.

CAPEX (Capital Expenditure) — капітальні витрати на імплементацію рішень.

DI (Dependency Injection) — впровадження залежностей для забезпечення слабкої зв'язності компонентів.

EF Core (Entity Framework Core) — технологія об'єктно-реляційного відображення для платформи .NET.

HR (Human Resources) — управління людськими ресурсами та кадрове адміністрування.

HTML (HyperText Markup Language) — мова розмітки гіпертексту, використана для створення шаблонів наказів.

HTTP (HyperText Transfer Protocol) — протокол передачі даних, на основі якого працює вебінтерфейс.

JWT (JSON Web Token) — стандарт для створення токенів доступу та забезпечення безпеки.

LINQ (Language Integrated Query) — мова запитів до різних джерел даних у середовищі .NET.

MVC (Model-View-Controller) — архітектурний патерн розробки програмного забезпечення.

PDF (Portable Document Format) — формат цифрових наказів, що генеруються системою.

QR (Quick Response) — матричний штрихкод для швидкої верифікації автентичності документів.

RBAC (Role-Based Access Control) — модель керування доступом до функцій системи на основі ролей.

REST (Representational State Transfer) — архітектурний стиль взаємодії між вебсервером та оркестратором Camunda.

SLA (Service Level Agreement) — регламент часових меж виконання завдань та контролю термінів погодження.

TCO (Total Cost of Ownership) — сумарна вартість володіння інформаційною системою. UI/UX (User Interface / User Experience) — інтерфейс та досвід взаємодії користувача з програмою.

UML (Unified Modeling Language) — уніфікована мова моделювання для проєктування архітектури.

UTC (Coordinated Universal Time) — всесвітній координований час для синхронізації подій та таймерів.

БД — База даних для збереження кадрової інформації.

КЗпП — Кодекс законів про працю України, вимоги якого інтегровані в логіку системи.

ВСТУП

В наш час для ефективного управління та виконання стратегічних цілей в будь-якій компанії, не залежно від розміру та форми власності, важливим є організація людських ресурсів. Враховуючи зростання ринку ІТ рішень для даної галузі майже 2 двічі до 2030 року, що буде становити близько 34 мільярдів доларів[22]. Такий ріст підтверджує, що вибрана тема є актуальною для вивчення. Наразі майже 85 % компаній частково втілили цифрові інструменти у своїй діяльності, але це в більшості випадків ефективність від цього не збільшилась[20]. Найбільшою проблемою в такій малій ефективності варто вважати відсутності інфраструктурної гучності, що насамперед полягаю у швидких змінах бізнесу та законодавства, зокрема перехід на цифровий документообіг. Для українських компаній швидкий перехід на цифровий документообіг та використання криптографічних інструментів(таких як Кваліфікований Електронний підпис (КЕП)) вже є обов'язковою умовою для стратегічного розвитку та виживання, в час військових дій[5]. Щоб підвищити ефективність цього переходу варто розглядати рішення з Business Process Management(BPM)[18]. Такий підхід дозволить підвищити прозорість та керованість кадрових процесів компанії.

Враховуючи вище описане, метою роботи є проектування та розробка інформаційної системи для автоматизації процесів відділу кадрів завдяки екосистемі .NET з інтеграцією BPM-платформи Camunda.

Ідеєю проєкту полягає на відокремленні бізнес логіки від програмної залежності. Це передбачає відхід від додавання логіки виконання заявок в самому коді програми, а винесення її в процеси BPM. Це забезпечує гнучкість зміни маршрутів погодження та їх редагування при потребі, налаштування нагадувань часових регламентів(SLA), формування електронних документів в форматі PDF та їх верифікацію за допомогою QR-code.

Об'єктом дослідження роботи є процеси управління персоналом, що допомагають здійснювати адміністрування роботи відділу. В роботі досліджується повний життєвий цикл кадрових даних, від створення до формування юридичного документу. При чому особлива увага приділяється механізмам, що зменшують вплив людського фактору при формуванні документів. Для оптимізації операційного контролю та чіткої маршрутизації завдань використовується розділення зон відповідальності в BPMN моделі.

Предметом дослідження є моделі, методи та засоби інтеграції системи управління бізнес процесами в веб систему з платформою .NET. Особлива увага в дослідженні була сфокусована: на візуалізації логіки в BPMN схемі; взаємодії між веб сервером та BPMN рушієм Camunda за допомогою REST API; механізми створення цифрових документів та ідентифікаторів до них.

Для виконання мети роботи складено та виконано наступні завдання:

- Виконано аналіз наявних рішень, який підтвердив, що закрита архітектура цих рішень впливає на швидкість прийняття необхідних змін. Це дає підтвердження, що використання зовнішнього BPM рішення, наприклад Camunda, дозволить впровадити інструментарій для швидкої зміни логіки процесів.
- Спроектовано систему за патерном MVC та UML діаграм погоджено протоколи обміну між клієнтом, контролерами системи та API.
- Створено BPMN-схему в середовищі Camunda Modeler, що відповідає функціональним вимогам бізнесу та при потребі може модифікуватися.
- Реалізовано модулів генерації документів, що дозволяють за допомогою Html-шаблонів автоматично створювати документи в PDF форматі[30] та додавати до них уніфікований ідентифікатор, що підтверджує їх справжність.
- Впроваджена система верифікації, що включає в себе перевірку введених даних користувачем та логування дій в системі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СПЕЦИФІКИ HR-ПРОЦЕСІВ

1.1. Дослідження особливостей організації бізнес-процесів

Щоб збудувати систему автоматизації HR в першу чергу треба проаналізувати як влаштовані процеси зсередини. В подальшому в цій роботі розглядається один із найрозповсюджених процесів, що використовується у роботі відділу кадрів, а саме заява на відпустку. Процес заявки на відпустку включає в себе погодження згідно з ієрархії структури компанії, що додатково забезпечує валідацію вказаних даних.

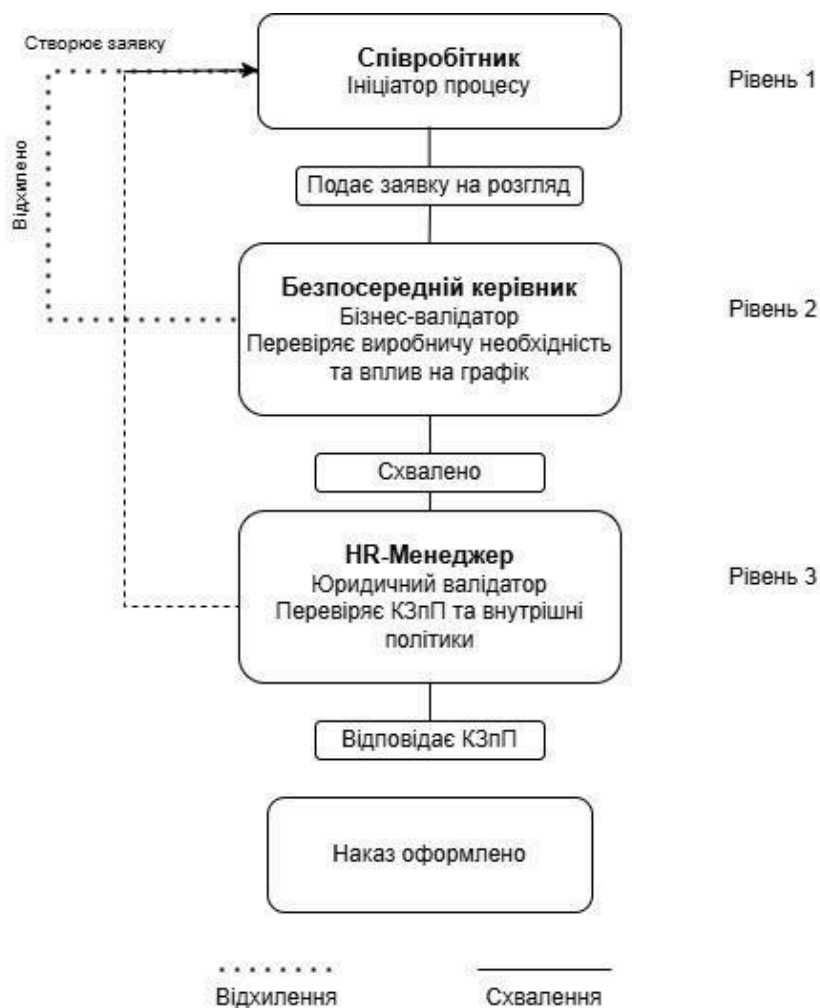


Рис. 1.1. Схема ієрархії погодження кадрових заяв

Рольова взаємодія погодження кадрових заяв

На рисунку 1.1 наведено схему погодження кадрової заявки на відпустку, яка розпочинається з формування запита Співробітник компанії, з вказанням типу відпустки та періоду дії відпустки. При збереженні відбувається перевірка доступного ліміту відпускних днів, якщо вказаний тип заявки цього потребує. Якщо перевірка пройдена, то заява переходить на погодження менеджеру.

Менеджер, він же безпосередній керівник співробітника, перевіряє чи вказаний період відпустки не впливає на роботу підлеглого підрозділу та його навантаження. Ця перевірка виконується як бізнес-валідація, що забезпечує додаткову перевірку коректності введених даних Співробітником. Безпосередній керівник може погодити заяву або відхилити її.

Після погодження Менеджером, заявка надходить HR-менеджеру, який перевіряє вказані дані по заяві на відповідність внутрішнім нормативним актам та вимогам трудового законодавства. При потребі HR-менеджер може відхилити цю заяву, вказавши причину відхилення. Результатом успішного погодження заявки є формування наказу на відпустку.

Нормативно-правове регулювання (згідно з КЗпП України[1])

Автоматизована система має відповідати Кодексу законів про працю(КЗпП) України[1], це обумовлює жорсткі вимоги:

- Перевірка з графіком відпусток та наявною кількістю днів, відповідно до статті 79 КЗпП[1]. Система має перевіряти, щоб співробітник міг подати заявку на основну щорічну відпустку на ту кількість днів, яка йому доступна за вказаний період.

- Інформування про початок відпустки не пізніше ніж за 14 днів до вказаного терміну початку відпустки. Це реалізується через механізм Timer Events, що мінімізують цей ризик.
- Фінансові обов'язки, що зафіксовані в статті 115 КЗпП[1], згідно цього виплата заробітної плати працівнику повинна бути виконана за 3 дні до початку відпустки.
- Звітність, що уособлює в собі наказ на відпустку, що відповідає формі П-3[2]. Система генерує ці документи за наявними шаблонами, що допомагає запобігти помилок.

Аналізуючи проблеми, що характерні для ручного управління цих процесів, потрібно відзначити наступні:

- Порушення вказаних термінів по погодженню документів, наприклад документ може затриматися у керівника.
- Інформування про поточний стан опрацювання документу, виникає ситуація коли співробітник не знає чи погоджена його заява та на якому етапі погодження вона знаходиться[5].
- Помилки при перенесені даних з друкованих заяв, що можуть призвести до юридичної недійсності кадрових документів.

Враховуючи вище описано, можна зазначити, що автоматизація кадрових процесів є необхідним напрямком для підвищення ефективності управління бізнесу.

1.2. Аналіз предметної області та життєвого циклу заяви

В автоматизованій системі запит що створив співробітник має трансформуватися в юридичний документ - наказ. Шлях що проходить заявка на відпустку супроводжується зміною станів у межах визначеної бізнес-логіки, що й

визначає життєвий цикл документу. При проходженні цього шляху виконуються додаткові валідація та контроль.

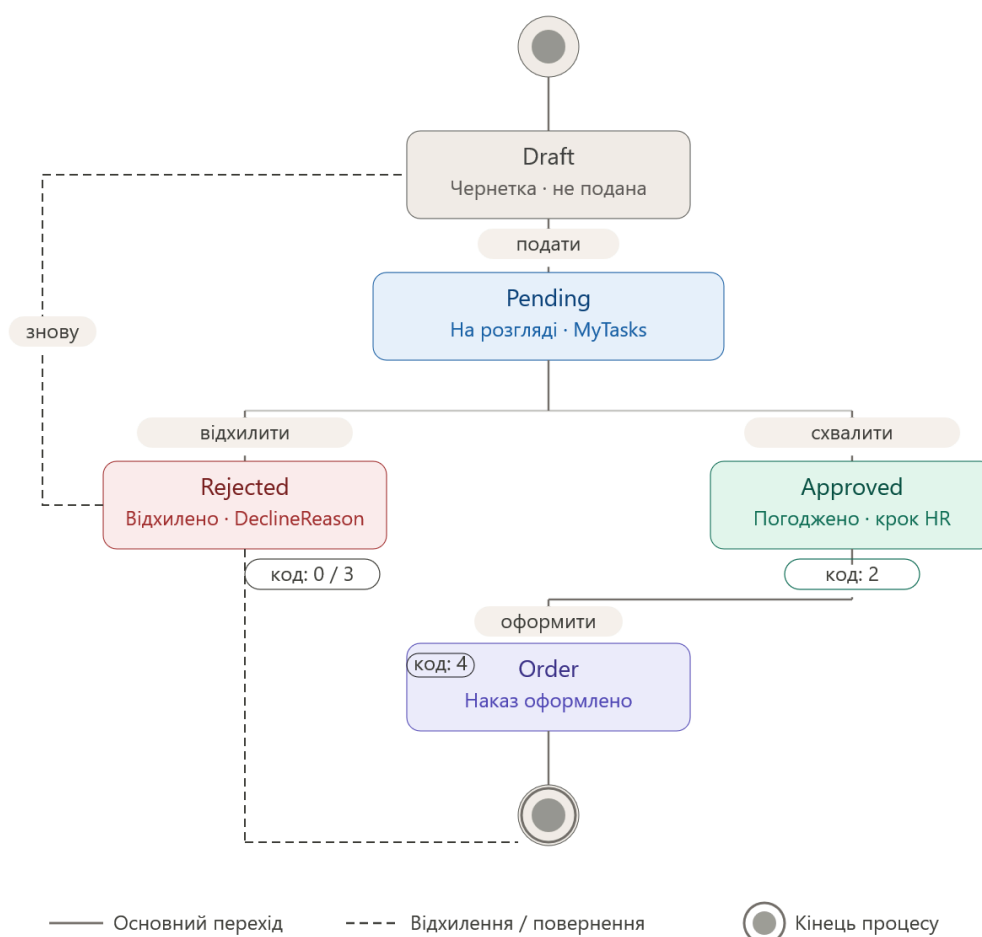


Рис. 1.2. Діаграма життєвого циклу заявки

На рис. 1.2. відображено життєві цикли запиту, які можна розділити на наступні фази:

1. Створення та заповнення(Статус - Чернетка(Draft))

Спочатку співробітник в системі створює заявку, в котру заносяться тип відпустки(щорічна основна, за власний рахунок або лікарняний)та період відпустки. Після створення заявки, дані зберігаються в базі даних та розпочинається процес в середовищі Camunda[17].

2. Перевірка внесених параметрів заявки.

Після внесення даних перед їх збереженням система перевірить чи внесені дати не суперечать наступним правилам (дата завершення повинна бути не менше дати початку, на вказані дати в системі не повинно бути активних заяв).

3. Управлінська перевірка(Статус - Очікує погодження(Pending Approval))

На цьому етапі безпосередній керівник співробітника визначає, чи можлива відсутність підлеглому у вказаний період та як це може вплинути на продуктивність підлеглому підрозділу. Керівник схвалює або відхиляє заявку.

4. Юридична трансформація заявки(Статус - Наказ(Order))

Завершальна стадія в якій заявка трансформується у юридичний документ - наказ. До наказу додається унікальний ідентифікатор QR код[23]. Цикл заявки можна вважати завершеним, коли співробітник підтверджує що він ознайомився з наказом.

Ключовими об'єктами нашої моделі є :

-Співробітник(Employee). Включає в себе персональні дані, назву посади, департаменту та актуальні відпускні ліміти.

-Заява(VacationRequest). Зберігає в собі параметри запиту, коментарі та історію змін по заявці.

-Наказ(Order). Юридичний документ, який містить у собі дані про реєстрацію та цифровий маркер.

Можна зробити висновок, що чим детальніше вказати життєвий цикл, тим інформативнішим буде система та подальшому допоможе в роботі відділу кадрів, це може дозволити автоматизувати до 80% щоденних перевірок [8]. Варто зазначити, що до фінансового відділка вже будуть надходити перевірені документи згідно внутрішніх стандартів та норм.

1.3. Формування вимог до програмного продукту

Майбутнє рішення повинне відповідати вимогам що були сформовані з результатів аналізу процесів відділу кадрів та їх специфіки роботи. Ці вимоги поділені за функціональними та якісним змістом, що відображає архітектурні особливості рішення.

1.3.1. Функціональні вимоги до системи

Цей тип вимоги відображає список сервісів, які за допомогою різних класів забезпечують користувачам повний цикл управління кадрових запитів:

- Керування повним циклом заяв. Співробітник повинен мати можливість функціонал для створення запитів з перевіркою типів заяв та часових меж. Сама система повинна перевіряти введені дані на невідповідність та виявлення суперечностей затвердженням раніше графіком.
- Координація та розподіл завдань(Task Management). Впровадження особистих ліній для суб'єктів що приймають рішення (керівник співробітника та працівника відділу кадрів). Результатом процесу повинно бути прийняте рішення та фіксація комунікації по процесу.
- Формування документів. Система повинна автоматично генерувати кадрові документи у формат PDF. В цих документах повинні міститися дані із процесу та вони мати QR код, для швидкої перевірки на унікальність.
- Оновлення статусів. Статуси повинні бути однакові для всіх середовищ -SQL Server чи Camunda BPM.

1.3.2. Нефункціональні вимоги

Надійність та продуктивність системи визначається наступними критеріями:

- Швидкодія, що передбачає швидкий відгук на запит до Camunda API та асинхронність.

- Керування безпекою, що повинна забезпечуватися моделлю RBAC (Role-Based Access Control) для розмежування прав по ролям. Персональні дані співробітників повинні шифруватися на рівні бази даних.
- Адаптивність інтерфейс (UI/UX), що передбачає роботу на різних типах пристроїв та зрозуміле візуальне відображення статусів;
- Стійкість, що передбачає інформування користувача при некоректній роботі системи та цілісність даних у сховищі при втраті з'єднання.

1.3.3. Технічні та експлуатаційні обмеження

Для стабільної роботи системи визначимо межі застосування системи:

- Архітектурні обмеження, що включають у себе доступність каналу між системою та Camunda через протокол REST, в разі відсутності переходити в режим перегляду даних та інформувати про це користувачів.
- Користувацькі обмеження, які передбачають що для коректної роботи користувач повинен використовувати сучасні версії браузерів.
- Юридичні обмеження, обумовлюють відповідність сформованих документів в системі до чинного законодавства[1].

1.4. Висновки до першого розділу

Після проведення аналізу предметної області, можна стверджувати, що успішна реалізація рішення по автоматизації кадрових процесів вимагає комплексного архітектурного підходу на відміну від часткового виконання. Підсумки, що стануть основою для подальшої розробки, та були сформовані при завершенні дослідження:

- Необхідність цифрової трансформації, що було підтверджено при аналізі роботи кадрового відділу - паперові методи погодження кадрових документів вже є не актуальними та застарілими. Враховуючи ризики при паперового обігу - втрата інформації при транспортуванні документації та час обробки,

що регламентується КЗпП України, роблять перехід до цифрового формату є найкращим вибором для забезпечення юридичної безпеки бізнесу.

- Деталізація процесної моделі, яке було досягнуто завдяки дослідженні взаємодії «Співробітник-Керівник-Відділ кадрів», а саме життєвого циклу заяви, що втілений не як набір статичних даних, а як динамічний потік станів. Це надало сформувати точки впровадження автоматичних валідацій, які мінімізують вплив людського фактору при валідації даних процесів.
- Затвердження функціонального ядра, що спирається на сам перед на транзакційній стійкості, яка втілюється в синхронізації між SQL сховищем та BPMN оркестратором, забезпечення динамічного створення PDF документів та їх верифікації за допомоги QR кодів.
- Встановлення якісних орієнтирів, що додатково були додатково додані до нефункціональних характеристик продукту. Серед яких багаторівнева модель доступу для захисту конфіденційної інформації та параметри продуктивності API, що забезпечують якісного користувацького досвіду в корпоративному середовищі.

Підводячи висновки, можна зауважити що була сформована цілісна теоретична та методична база, що дозволяє перейти до наступного етапу- проєктування архітектури рішення та розробки інформаційної моделі, що буде висвітлено далі.

2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Порівняльний аналіз програмних рішень-аналогів

Сьогодні існує велике різноманіття рішень для автоматизації кадрових процесів, які між собою відрізняються архітектурною філософією, вартість та рівнем реалізації. Щоб визначити оптимальний варіант для майбутньої розробки було проведено детальний огляд трьох основних підходів: впровадження монолітної Enterprise Resource Planning(ERP) платформи, використання Software as a Service(SaaS) продукту та створення внутрішньої системи(in-house) без додавання процесного керування.

2.1.1. Комплексні ERP-платформи

В системах типу Enterprise Resource Planning (ERP) функціонал для відділу кадрів виноситься в окремий модуль, що є одним з елементів екосистеми. Одні з найпопулярніших рішень цього класу є SAP, Oracle та Microsoft Dynamics 365, з локальних, варто відзначити, IT-Enterprise та системи лінійки BAS.

До переваг цих систем варто відносити єдність системи та даних в ній. При використанні таких рішень система надає великі можливості по різних напрямкам, але здебільш функціонал системи є шаблонним, що може супроводжуватися «операційною інерцією»[27] при впровадженні. Це виражається у високій вартості майбутніх змін при потребі та займає тривалий час для втілення потрібних змін. Варто відзначити, що в глобальних гравців цього рішення зазвичай діють загальні налаштування, що не завжди дозволяють кастимізувати потрібні кадрові вимоги з вимогами поточного законодавства.

2.1.2. Хмарні SaaS-екосистеми (Software as a Service)

Рішення типу Software as a Service (SaaS) дозволяє скоротити час впровадження, так як на відміну від ERP рішення не потребує налаштувань інфраструктури, бо сервіси знаходяться в хмарному просторі і доступ в систему реалізується через веб-інтерфейс[21].

Хмарні рішення мають переваги в швидкості розгортання та надання доступу, але з іншого боку є велика залежність від постачальника рішення, зберігання конфіденційних даних на сторонніх сервісах має великий ризик, в більшості вітчизняних компаніях це є перепорою для провадження цього рішення.

2.1.3. Внутрішня розробка (In-house) без використання BPM

Компанія може прийняти рішення на розробку програмного забезпечення силами внутрішньої команди для отримання продукту , що максимально задовільнить її потреби[9].

Гнучкість прийняття рішень, швидкість реалізації та безпека даних можна вважати як переваги цього вибору. В більшості випадків маршрути погодження та бізнес-правила додані в програмний код, що дуже обмежує зміну логіки при потребі. Також Одним з недоліків таких систем є відсутність візуального відслідковування зміни статусів кадрових даних, що унеможливорює їх відслідковування в реальному часі.

2.1.4. Резюме та обґрунтування вибору

Проаналізувавши наявні рішення, можна зробити висновок, що кожен із розглянутих типів систем не задовольняє балансу між гнучкістю та системністю.

Бізнес логіка буде побудована на базі Camunda BPM, що забезпечить нам прозорість процесів та швидкість їх змін, а бекенд буде реалізований як кастомний .NET-додатком, що забезпечить надійність та масштабованість.

2.2. Моделювання функціональних можливостей системи

Застосовуючи методологію взаємодії (Use Case modeling)[29] було сформулювати ключові сервіси майбутньої системи та визначити чітку топологію між програмним ядром взаємодії та людськими ресурсами.

2.2.1. Рольова модель акторів

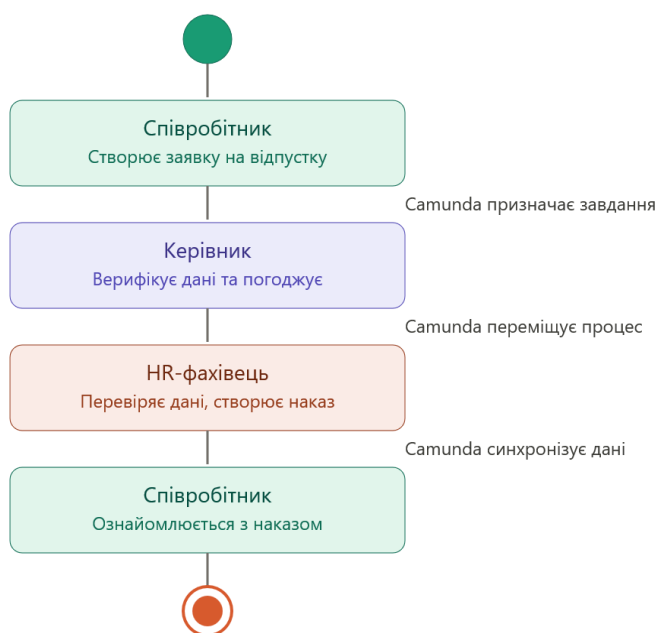


Рис. 2.2.1.1. Рольова модель акторів

На рис. 2.2.1.1. зображена рольова модель акторів, що включає в себе три основні категорії користувачів та одного системного компонента, що теж приймає участь в інформаційному обміні:

- Співробітник (Employee). Є ініціатором процесу та виконує такі дії: розпочинає процес зі створення заявки на відпустку, відстежує статус заявки та виконує ознайомлення з наказом на відпустку.
- Керівник (Manager). Відповідальний за прийняття управлінських рішень. Виконує верифікацію введених даних та погоджує відпустку.
- HR-фахівець (HR Manager). Відповідальний за кадрові вимоги. Виконує перевірку введених даних та створює наказ.
- Camunda API (зовнішня система). Виконує роль процесного оркестратора. Забезпечує керування бізнес-процесом відповідно до BPMN-моделі: створює та призначає завдання, переміщує процес між станами та синхронізує дані із системою.

2.2.2. Сценарії взаємодії та функціональна логіка



Рис. 2.2.2.1. Поведінкова модель системи

На рис. 2.2.2.1 зображено схему поведінкової моделі системи, що формується через набір базовий сценаріїв, кожен з яких відображає бізнес операцію:

1. Ініціація відпусного запиту: Процес розпочинається після заповнення метаданих заявки співробітником, далі проходить валідації полів, післячого .NET-додаток через REST-протокол звертається до Camunda API для створення

нового екземпляра процесу (Process Instance), це фактично запускає механізм маршрутизації.

2. Опрацювання черги завдань (User Task Processing): При визначенні виконавця (Менеджер чи HR-фахівць) система формує модель «Push-сповіщень», де завдання з'являються в персоналізованих списках акторів. Обробка включає перегляд агрегованих даних заяви та фіксацію вердикту в системних змінних.
3. Автоматизована збірка документації: Подія, що активується автоматично після проходження всіх контрольних точок погодження. Система витягує дані з SQL-бази, наповнює ними HTML-шаблон та конвертує результат у PDF-файл, що відповідає державним стандартам діловодства.
4. Спостереження за станом процесу (Observability): Надає акторам можливість ретроспективного та поточного аналізу. Це включає візуалізацію пройдених етапів та ідентифікацію поточного статусу заявки в маршруті погодження.

2.2.3. Архітектурна інтерпретація Use Case Diagram

Розроблена діаграма варіантів використання (Use Case Diagram) слугує візуальним підтвердженням того, що система не є простою базою даних, а функціонує як динамічний механізм, керований подіями. Центральне місце на діаграмі посідає взаємодія між акторами та процедурним ядром через абстракцію User Tasks.

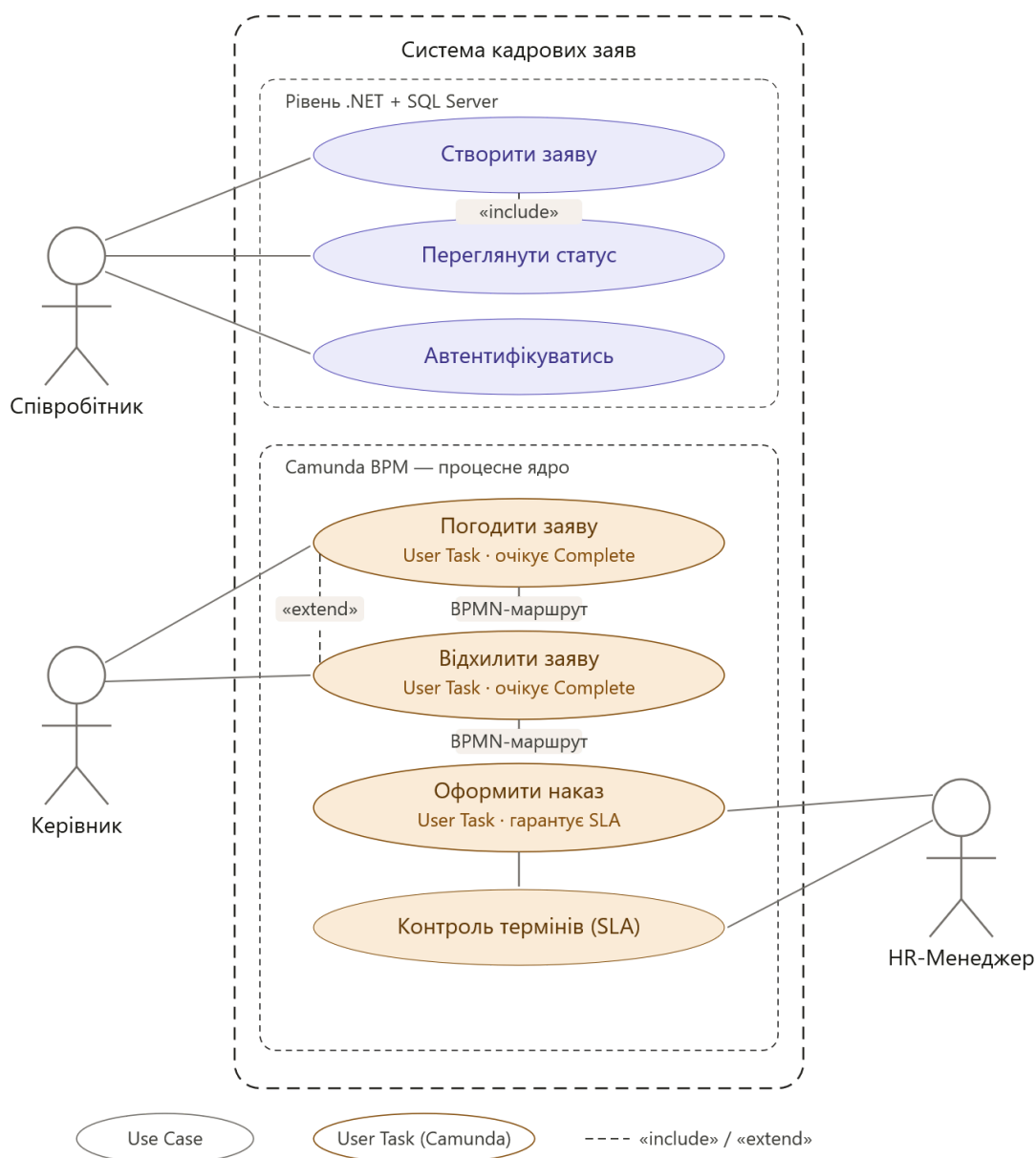


Рис. 2.2.3.1. Діаграма варіантів використання (Use Case Diagram)

На рис. 2.2.3.1. відображена модель взаємодії, що ілюструє не пасивне збереження даних, а активне управління діями користувача. Саме Camunda визначає наступний крок системи, це забезпечує послідовність виконання дій[29]. Додатковою особливістю даної моделі є ще те, що сценарії «Погодити запит» та «Оформити наказ» є не просто функціями інтерфейсу, а системними завданнями, які очікують на завершення (*Complete*) для подальшого просування процесу за схемою.

Це гарантує суворий контроль за термінами виконання (*SLA*) та прозорість кожної кадрової операції [26].

2.3. Проєктування архітектури та інформаційного забезпечення

При проєктуванні внутрішньої структури системи було зроблено акцент на принципі модульності та слабкої пов'язаності компонентів, це забезпечує створення стабільної архітектури, що забезпечить безперебійну взаємодію між інтерфейсом користувача та ядром процесу, та забезпечить довгострокову перспективу зберігання кадрової інформації[16].

2.3.1. Архітектурна парадигма: Гібрид MVC та BPM-оркестрації

Для реалізації системи було обрано багаторівневу сервісно-орієнтовану архітектуру, в основі якої лежить класичний патерн MVC (*Model-View-Controller*). Однак, на відміну від стандартних вебзастосунків, де бізнес-логіка інкапсульована в контролерах або сервісах додатка, у даному проєкті реалізовано зовнішню оркестрацію процесів[6].

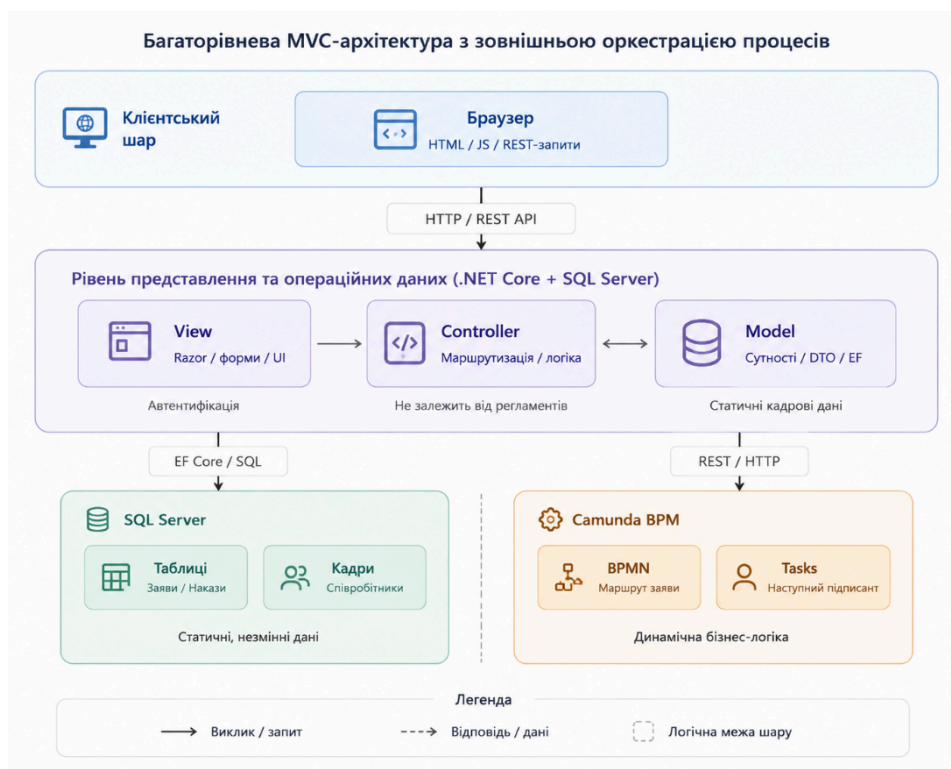


Рис. 2.3.1.1. Багаторівнева MVC-архітектура з зовнішньою оркестрацією процесів

На рис. 2.3.1.1. проілюстровано MVC-архітектура з зовнішньою керуванням процесів, яка розмежовує відповідальності на два технологічні рівні:

- Рівень представлення та операційних даних (.NET + SQL Server): цей стек відповідає за рендеринг інтерфейсних форм, механізми автентифікації та збереження статичної інформації про кадровий склад. Тут реалізовано логіку, яка не змінюється залежно від бізнес-регламентів.
- Рівень процесного керування (Camunda BPM): виступає як інтелектуальний шар, що динамічно визначає маршрут руху кожної заявки. Застосунок на базі .NET не володіє інформацією про те, хто є наступним підписантом у ланцюжку — він лише делегує запит до Camunda через REST-інтерфейс та отримує інструкції щодо подальших кроків.

2.3.2. Проектування реляційної моделі даних (ER-модель)

Інформаційне ядро системи спроектоване за принципами нормалізації реляційних баз даних. На рис. 2.3.2.1. зображена ER-діаграма основних таблиць бази даних проекту.

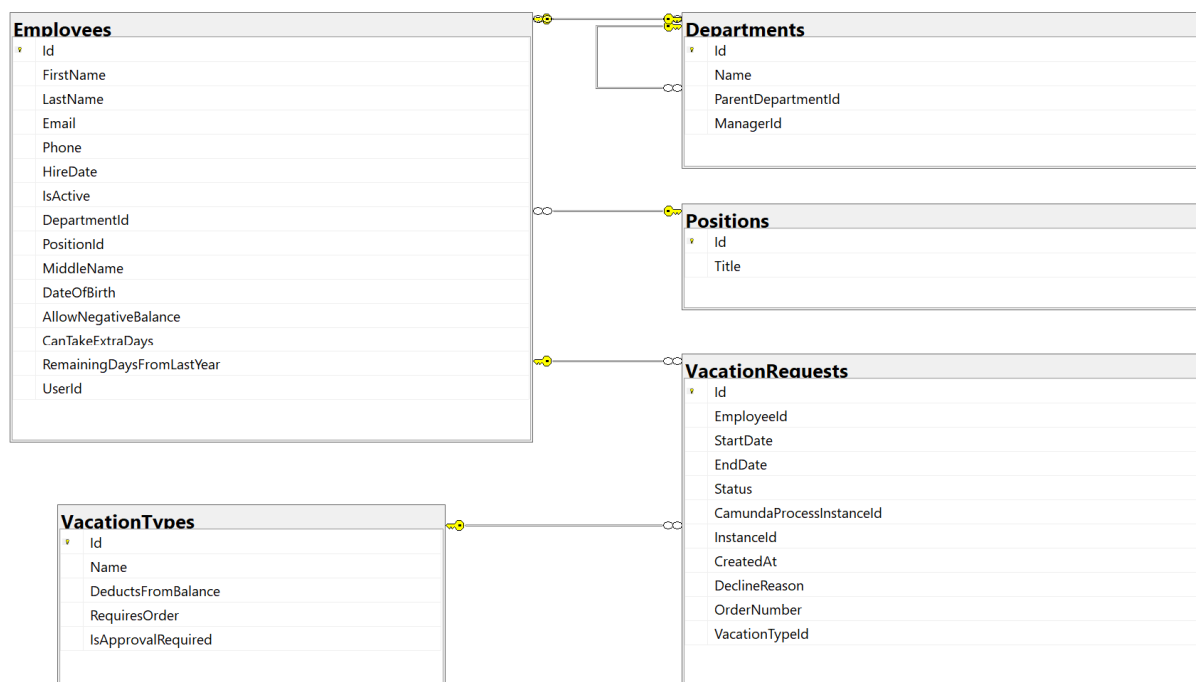


Рис. 2.3.2.1. ER-діаграма бази даних

До структури бази даних включено такі базові сутності:

- **Employee** (Співробітник): центральний об'єкт, що містить персональні ідентифікатори (Id, FullName), професійні атрибути (Position, DepartmentId) та критично важливий для валідації параметр VacationBalance. Останній динамічно оновлюється при кожному успішному завершенні циклу відпустки.
- **VacationRequest** (Заява): об'єкт, що фіксує параметри конкретного запиту (StartDate, EndDate, VacationType). Ключовим атрибутом тут є CamundaProcessId — унікальний зовнішній ключ, який дозволяє системі миттєво ідентифікувати стан виконання процесу в оркестраторі при зверненні до конкретного запису в базі.

- Order (Наказ): сутність, що виникає як результат успішного завершення процесу. Вона містить посилання на фізичний PDF-файл (FilePath), реєстраційний номер та булеву ознаку IsSigned, що підтверджує юридичне ознайомлення працівника з документом.

2.3.3. Механізми синхронізації та парадигма Dual Storage

Одним із важливих викликів при інтеграції зовнішнього BPM-двигуна є забезпечення цілісності інформації. У проєкті реалізовано стратегію Dual Storage (подвійного зберігання), де кожна база даних має виконувати свою специфічну роль.

Згідно з цією моделлю, SQL Server виступає «джерелом істини» (Source of Truth) для бізнес-аналітики та анкетних даних[24]. Водночас Camunda DB (внутрішнє сховище двигуна) оперує метаданими процесу: часом виконання завдань, поточними виконавцями та історією переходів між шлюзами.

Проблема десинхронізації вирішується через сервісний прошарок на базі .NET Core. Система використовує подієву модель: при настанні певної події в Camunda (наприклад, завершення завдання менеджером) ініціюється зворотний виклик (callback) до API застосунку. Це спричиняє автоматичне оновлення поля Status у таблиці VacationRequests. Такий підхід гарантує, що користувач завжди бачить актуальний стан своєї заяви в особистому кабінеті, незалежно від складності маршруту погодження в оркестраторі.

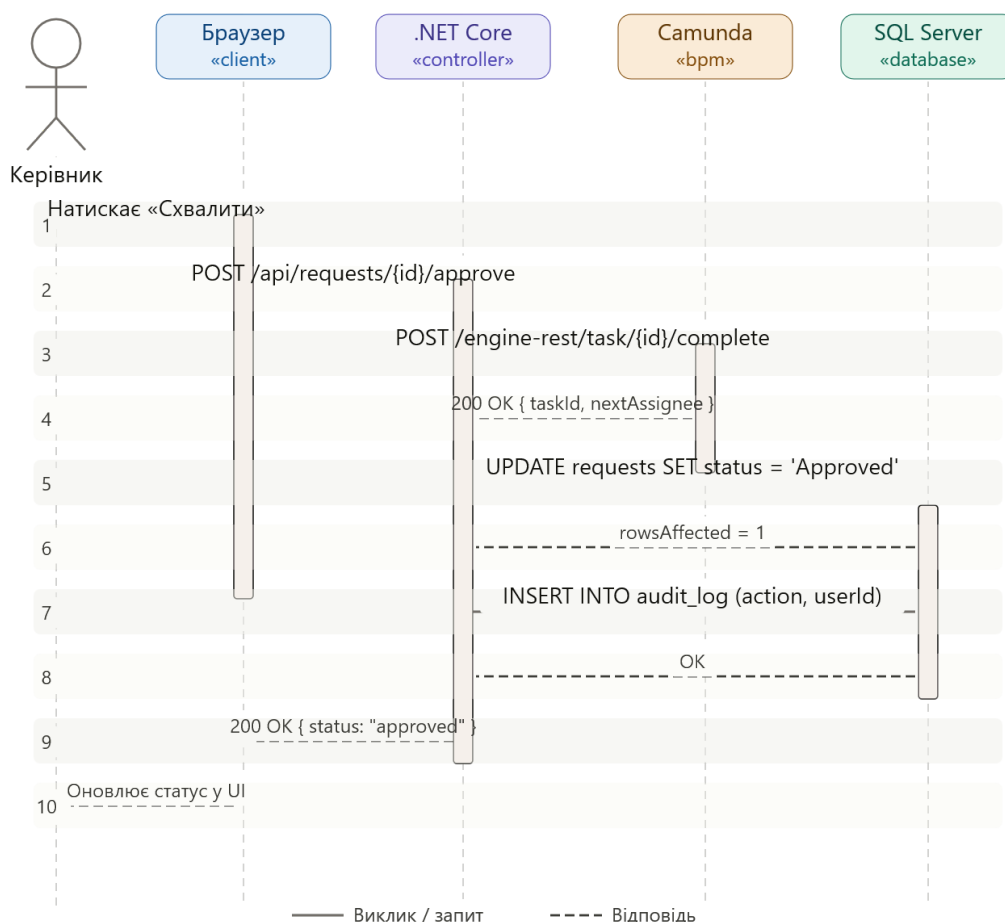


Рис. 2.3.3.1. Діаграма послідовності

2.4. Обґрунтування вибору технологічного стеку та засобів розробки

Згідно вимог відмовостійкості, потенціалу масштабування та швидкодії змін є сформований базис системи автоматизації HR-процесів. Вибраний стек дозволяє реалізувати складну бізнес-логіку, забезпечую при цьому безпеку даних та чистоту архітектури.

2.4.1. Платформа .NET та мова програмування C#

Для розробки серверної частини (backend) обрано екосистему .NET 8/9 у поєднанні з мовою C#.

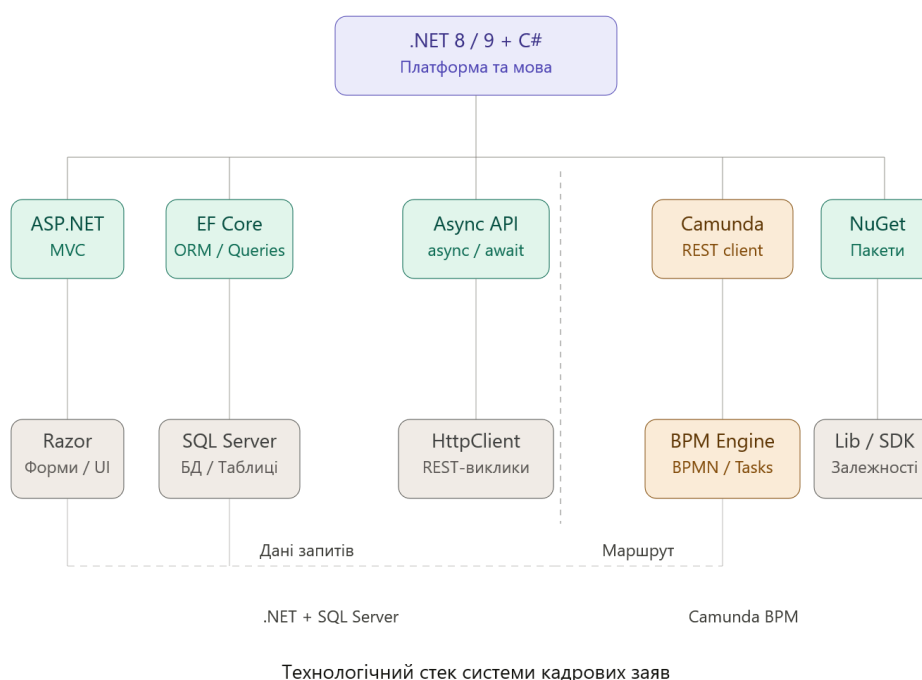


Рис. 2.4.1.1. Ілюстрація технологічного стеку системи

На рис. 2.4.1.1. показано схемо технологічного стеку проєкту, що дозволяє зробити такий висновок, даний вибір не є випадковим і базується на наступних аргументах:

- Суворі статична типізація: У контексті обробки юридично значущих наказів та персональних даних працівників використання C# дозволяє мінімізувати помилки на етапі компіляції. Це гарантує детермінованість типів даних, що критично для фінансових розрахунків та дотримання часових лімітів КЗпП.
- Парадигма асинхронності: Глибока інтеграція паттерну `async/await` забезпечує ефективне управління зовнішніх викликів до Camunda REST API. Це дозволяє системі обробляти велику кількість паралельних запитів без зайвого споживання ресурсів сервера [15].

- Інженерна зрілість: Актуальні версії .NET (8 та вище) демонструють високу швидкодію в бенчмарках вебзастосунків. Наявність розгалуженої системи пакетів (NuGet) надає перевірені рішення для криптографії, генерації документів та QR-кодування, що дозволяє зосередитися на бізнес-логіці, а не на низькорівневій реалізації.

2.4.2. Система управління бізнес-процесами Camunda BPM

Впровадження спеціалізованого оркестратора Camunda дозволяє відмовитися від «жорсткого кодування» маршрутів погодження.

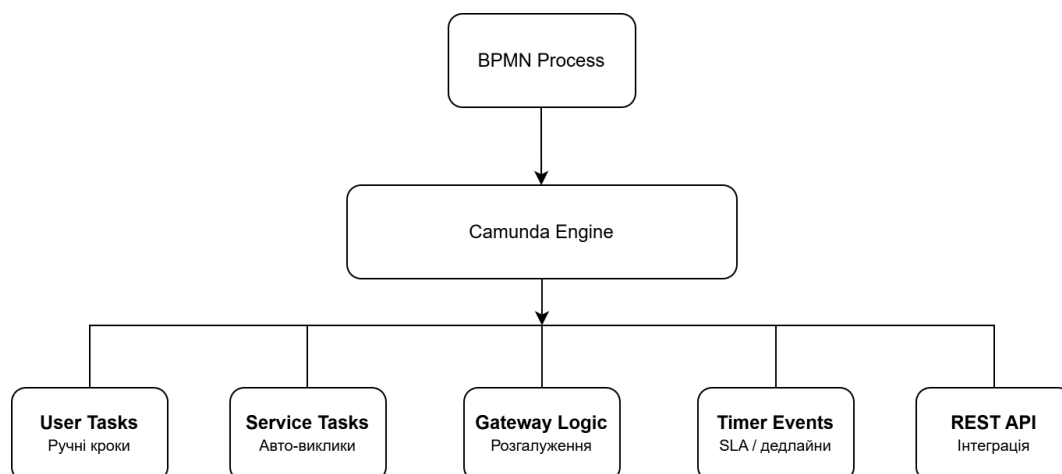


Рис. 2.4.2.1. Схема BPM-оркестрації Camunda

На рис. 2.4.2.1. зображено схему BPM керування Camunda, що засвічує що це рішення обумовлене стратегічним переходом від статичних алгоритмів до гнучкого процесного управління:

- Абстракція за стандартом BPMN 2.0: Використання міжнародної нотації дозволяє трансформувати візуальні моделі (аналогічні розробленим у пункті 2.2) безпосередньо у виконувані сценарії. Це створює спільне інформаційне поле для розробників та HR-аналітиків, спрощуючи верифікацію логіки.
- Декапсуляція бізнес-правил (Separation of Concerns): Відокремлення послідовності кроків від коду додатка дозволяє модифікувати регламенти

- (наприклад, змінювати кількість рівнів підпису) без перепрограмування. Корекція схеми в Camunda Modeler не потребує перезапуску всього вебсервісу.
- Прозорість та моніторинг: Наявність вбудованого REST-інтерфейсу надає системі можливість відстежувати стани кожного екземпляра процесу в реальному часі, ідентифікуючи затримки (SLA) та оптимізуючи HR-цикли [19].

2.4.3. Технології доступу до даних: Entity Framework Core

Для мапінгу об'єктів у реляційне середовище обрано Entity Framework Core (EF Core). Його роль у системі полягає у спрощенні взаємодії з SQL Server та підвищенні безпеки операцій:

- Code-First: Суть цієї стратегії, в тому що вона дозволяє описувати структуру даних мовою програмування, через класи, які автоматично синхронізують зміни через механізм міграцій. Це допомагає попередити ризик розбіжностей між кодом та схемою БД.
- SQL-ін'єкцій: Entity Framework Core використовує параметризовані запити, що створює надійний бар'єр проти SQL-ін'єкцій — одного з найпоширеніших векторів атак на корпоративні системи[25].
- LINQ: Використання мови запитів LINQ дозволяє писати вибірки даних як частину C# коду. Це не лише покращує читабельність, а й забезпечує перевірку запитів під час компіляції, що неможливо при використанні сирого SQL[11].

2.4.4. Допоміжні бібліотеки для документообігу

Для автоматизації фінальної стадії життєвого циклу заяви обрано спеціалізовані інструменти:

- Бібліотека DinkToPdf, що допомагає перетворювати HTML сторінку, в якій зберігається шаблон документу, на PDF файл[30]. Такий підхід дозволяє втілювати складні шаблони та швидко їх змінювати.
- Бібліотека QRCode дозволяє створювати матричні QR коди, завдяки інтеграції в документ[31]. Це дозволяє виконати швидку перевірку справжності документа за допомогою мобільного пристрою.

2.5. Висновки до другого розділу

Завдяки отриманим результатам проєктування майбутнього рішення, можна стверджувати, що обрана стратегія розробки дозволяє створити міцну технологічну основу для автоматизації роботи відділу кадрів. На основі виконаного архітектурного аналізу окреслено наступні тез:

Гнучкість, що забезпечується відокремлення бізнес-логіки в окремий шар Camunda BPM. Таке рішення забезпечує високу гнучкість при потребі зміни логіки процесів, що мінімізує зміну в програмному коді, технічні ризики та знижує сумарну вартість володіння системою у перспективі [18].

Модульність, яка забезпечується використанням патерну MVC поєднанні з сервісно-орієнтованим підходом дозволило розробити систему з високим потенціалом до горизонтального масштабування[11]. Створена модель забезпечує цілісність даних та їх подальшому доступність.

Стійкість системи забезпечується завдяки обраному стеку (.NET 8, C#, MS SQL Server), що задовольняє найвищі стандарти корпоративного програмного забезпечення[24]. Надійним бар'єром проти критичних помилок є сувора типізація C# з ACID-транзакцій бази.

Завершеність системи підтверджується побудовою цілісного набору UML діаграм, що слугують навігаційною картою для подальшої розробки. Додавання

модулів автоматичного створення PDF документів та їх підтвердження QR-ідентифікацією завершує повний життєвий цикл цифрового документа в системі.

Розглянута архітектурна модель є цілісною, збалансованою та завершеною, що дозволить їй виконати не лише завдання по автоматизації роботи відділу персоналу, а дає можливість в подальшому нарощувати додатковий функціонал.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура програмної реалізації

Процес впровадження розробленої системи ґрунтується на поєднанні наступних принципів - чистої архітектури (Clean Architecture) та концепції розділення відповідальності (Separation of Concerns)[12]. Цей підхід дозволив розробити гнучку систему, елементи якої мають велику автономність, а саме зміна механізму формування PDF або оновлення версії процесу в Camunda не призведе до глобальних помилок в базі даних чи інтерфейсі користувача.

3.1.1. Архітектурна побудова та склад компонентів

Проект побудовано як багаторівневий застосунок на базі фреймворк ASP.NET Core[7], де кожний елемент має визначену локацію та зону відповідальності.

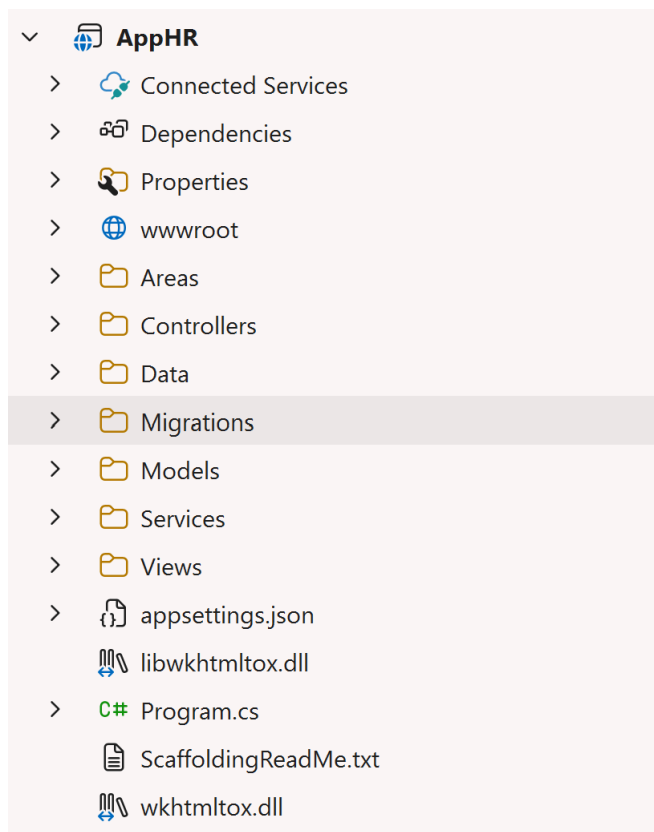


Рис. 3.1.1.1. Структура проєкту в Visual Studio

Проект має такий склад:

- Контролери (Controllers) виконують функцію шлюзів системи, що приймають HTTP-запити, їх обробляють та перенаправляють до інших компонентів системи. Для прикладу, контролер `VacationController` опрацьовує вхідні запити на відпустку, а `TaskController` - відповідає за черги завдань в системі.

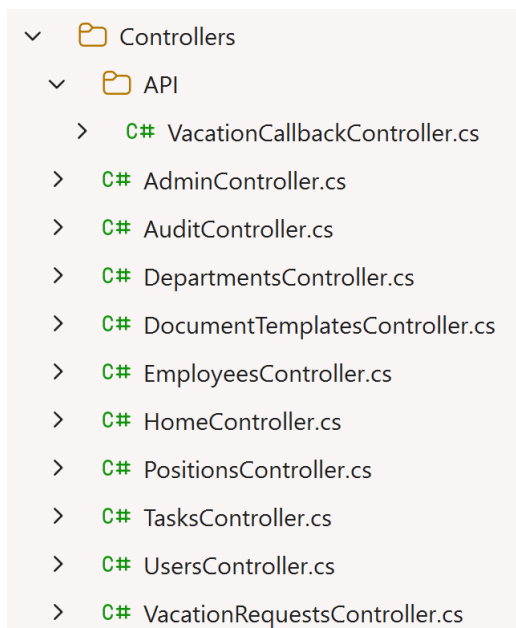


Рис. 3.1.1.2. Список контролерів в проєкті

- Сервіси (Services) відповідають за бізнес-логіку в системі, викликаючи відповідні методи сервісів. Для прикладу сервіс CamundaService відповідає за взаємодію між .Net елементами та процесами в Camunda.

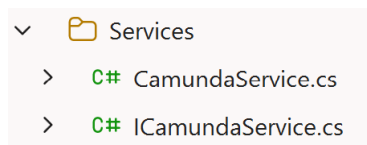


Рис. 3.1.1.3. Список сервісів в проєкті

- Моделі (Models) - класи, що відображають елементи об'єктно-реляційної структури даних. Завдяки фреймворку Entity Framework Core (EFC) класи мігрують у схеми SQL Server, що забезпечує цілісність даних на рівні сховища.

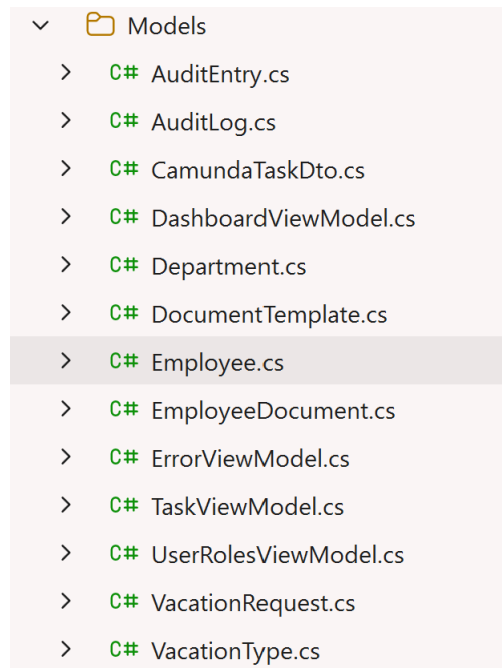


Рис. 3.1.1.4. Список моделі в проєкті

- Представлення (View) відповідають за підготовку та відображення даних на сторінках системи. Завдяки представленням відбувається оптимізація трафіку між сервером та клієнтом, також інтерфейси користувача, як їх ще можна назвати, є засобом захисту від вразливостей «overposting», що дозволяє приховати технічну інформацію та параметри моделей[25].

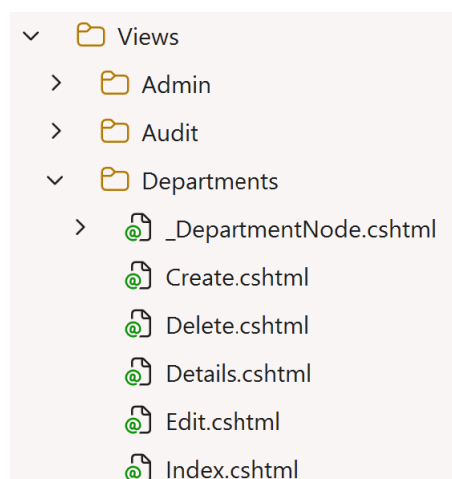


Рис. 3.1.1.5. Список приставлень сторінки Departments в проєкті

3.1.2. Імплементация механізму впровадження залежностей (Dependency Injection)

Враховуючи, що в ASP.NET Core[7] початково інтегрований патерн Dependency Injection (DI), що забезпечує гнучкість розробленої системи. DI дозволяє відмовитися від прямого виклику об'єктів у коді контролерів, а використовуючи контейнер залежності.

Цей підхід забезпечив низку критичних переваг для архітектури:

- Слабкий зв'язок (Loose Coupling), завдяки інверсії управління компоненти залежать від абстрактних інтерфейсів (наприклад, ICamundaService), а не від їхніх конкретних реалізацій. Це дозволяє під час модульного тестування легко замінювати реальні сервіси на імітаційні об'єкти (Mock-об'єкти), не вносячи змін у код виклику.
- Керування життєвим циклом, що здійснює система розподіляючи тривалість життя об'єктів(HTTP-запитів, миттєвих операцій тощо), таким чином оптимізує використання оперативної пам'яті.
- Прозорість опису конфігурації, що забезпечується через структуру конструкторів класів, з яких стає зрозуміло які зовнішні інструменти потрібно підключити до компонента для виконання його функціоналу.

В файлі Program.cs здійснюється реєстрація та конфігурація вище вказаних зв'язків. На рис. 3.1.2.1. наведено приклад додавання конфігурації DI-контейнерів:

```
builder.Services.AddScoped<ICamundaService, CamundaService>();  
builder.Services.AddTransient<IQRCodeGenerator, QRCodeGenerator>();
```

Рис. 3.1.2.1. Приклад типової конфігурації DI-контейнера

Така реалізація забезпечує системі власноруч обмежувати необхідні залежності в період роботи, що відповідає кращим практикам розробки програмного забезпечення.

3.2. Реалізація бізнес-процесу засобами Camunda Modeler

Практична імплементація алгоритму подання та обробки заявок на відпустку базується на виконанні BPMN-моделі, розробленої в середовищі Camunda Modeler[8].

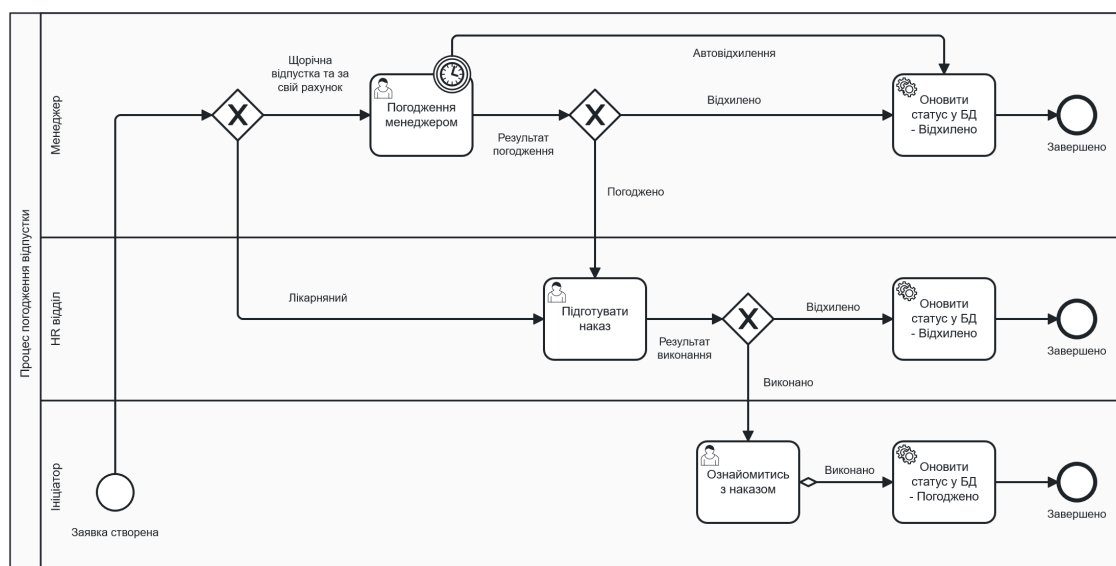


Рис. 3.2.1. BPMN-схема процесу в Camunda Modeler

В процесі, що відображено на рис. 3.2.1, продемонстровано розділення зон відповідальності (Swimlanes) по ролям відповідній послідовності погодження для забезпечення структурної чіткості та розмежування відповідальності:

- Працівник (Employee). Розпочинає процес та виконує завдання користувача (User Task) для введення даних по заявці на відпустку;
- Керівник (Manager). Приймає рішення щодо бізнес доцільності вказаних даних співробітником;

- Відділ кадрів (HR). Виконує юридичну перевірку внесених даних та фіналізацію процесу (формування наказів).

3.2.1. Логічне розгалуження та обробка бізнес-рішень

Основним елементом управління потоком операції в процесі є шлюз Exclusive Gateway («Погоджено»). Він аналізує булеву змінну *approved*, що надходить з інтерфейсу керівника після завершення його завдання.

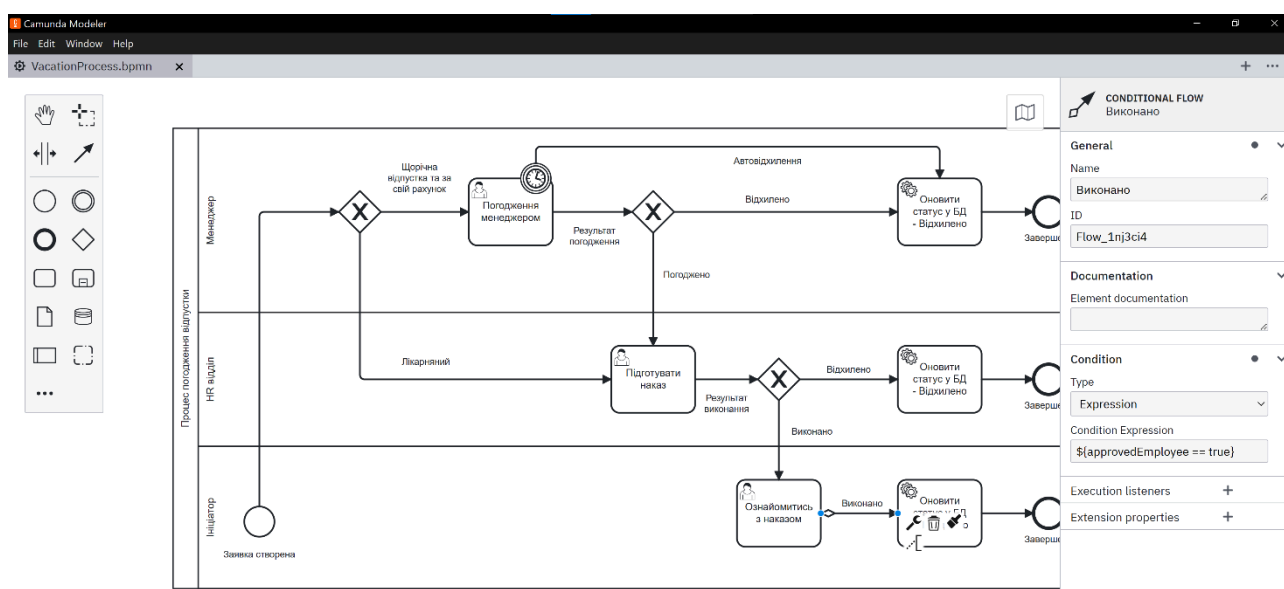


Рис. 3.2.1.1. Булева змінна в елементах BPMN-схеми

На рис.3.2.1.1. відображено приклад перевірки значення булевої функції *approved* при завершенні завдання, що вказує залежність від значення булевої функції та результаті виконуються наступні сценарії руху процесного токена:

- Позитивний вектор(*approved == true*). Токен спрямовується до доріжки HR департаменту для початку процедур формування наказу.
- Негативний вектор (*approved == false*).Співробітнику надходить сповіщення про відмову, після чого процес завершується.

Кожне рішення шлюзу додається у список логування Camunda History Service, це дозволяє при потребі проводити аудит дій з додатковими даними часових показників[19].

3.2.2. Автоматизація через сервісні завдання (Service Tasks)

Операціях, що не потребують взаємодії з людьми, використовується паттерн External Task[24], який дозволяє відділити виконання процесів від програмної реалізації, в Camunda створює запис у черзі завдань з вказанням назви, а зовнішній .NET-воркер (External Worker) працює з ним. У системі реалізовано наступні сервісні вузли:

- Створення документів(GeneratePDF). Потік відслідковує .NET-сервіс, який опрацьовує переміни процесу, та виконує будувє PDF файл за шаблоном, та посилання на файл записується у контекст процесу.

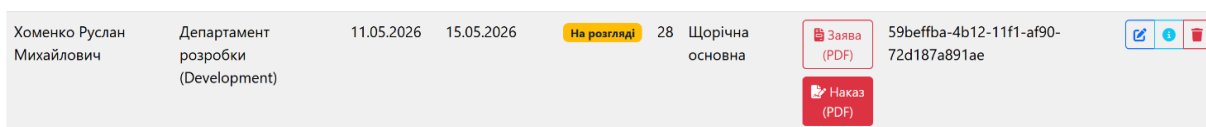


Рис. 3.2.2.1. Приклад відображення завдання з PDF-файлами

- Синхронізація статусів (UpdateStatus). За допомогою викликів API оновлюються статуси у базі даних, це дозволяє відображати актуальний стан запитів в кабінеті без прямого звернення до Camunda.

Для чіткою маршрутизації викликів між .NET середовищем та BPMN використовується унікальний ID, який присвоюється для кожного елемента моделі, на рис. 3.2.2.1 проілюстровано код – *59beffba-4b12-11f1-af90-72d187a891ae*.

3.2.3. Управління часовими регламентами (Timer Events)

Інтеграція часових подій у модель, що допомагають автоматизувати контроль дедлайнів, для забезпечення внутрішніх SLA організації та нормам КЗпП[1]:

- Boundary Timer Event. Межовий таймер, що відслідковує виконання завдання погодження керівника, якщо завдання не завершено після 48 годин очікування. На рис. 3.2.3.1. відображається реалізація автовідхилення заявки, якщо менеджер не виконав завдання у вказаний період.
- Intermediate Catch Timer Event. Таймер проміжного очікування, застосовуються для автовідхилення запитів, так як завдання буде формувати відповідальному кадровому відділу тільки за 3 дні до фактичної початки відпустки.

Для коректної обробки запланованих подій потрібно синхронізувати часових поясів (UTC) та налаштувати Job Executor у Camunda. Кожна подія таймера повинна логуватися, що посилює систему моніторингу якості бізнес-процесів.

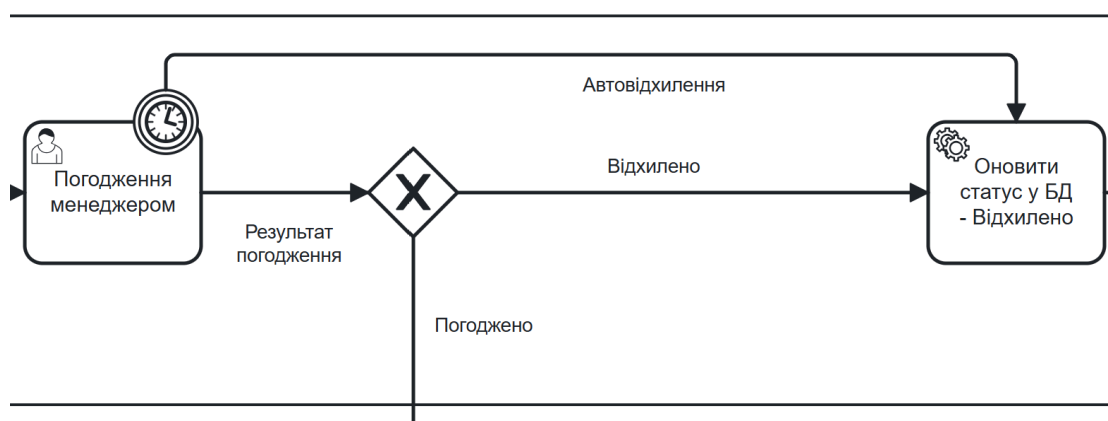


Рис. 3.2.3.1. Реалізація часових подій в BPMN-схемі процесу

3.3. Реалізація функціональних модулів системи

Три автономні модулі дозволили забезпечити технічний базис системи, результатом їхньої спільної роботи є трансформація віртуального бізнес-процесу на юридично змінений результат.

3.3.1. Модуль динамічної генерації PDF-наказів (DinkToPdf)

Для трансформації структурованих кадрових даних у легітимні накази імплементовано модуль на базі бібліотеки DinkToPdf[30]. Це рішення є високорівневою .NET-обгорткою над рушієм wkhtmltopdf[32], що використовує механізми рендерингу WebKit для точного відтворення HTML-розмітки у форматі PDF[18]. На рис. 3.3.1.1 показано форму редагування шаблону заяви на відпустку з вказанням тегів, що доступні для додавання.

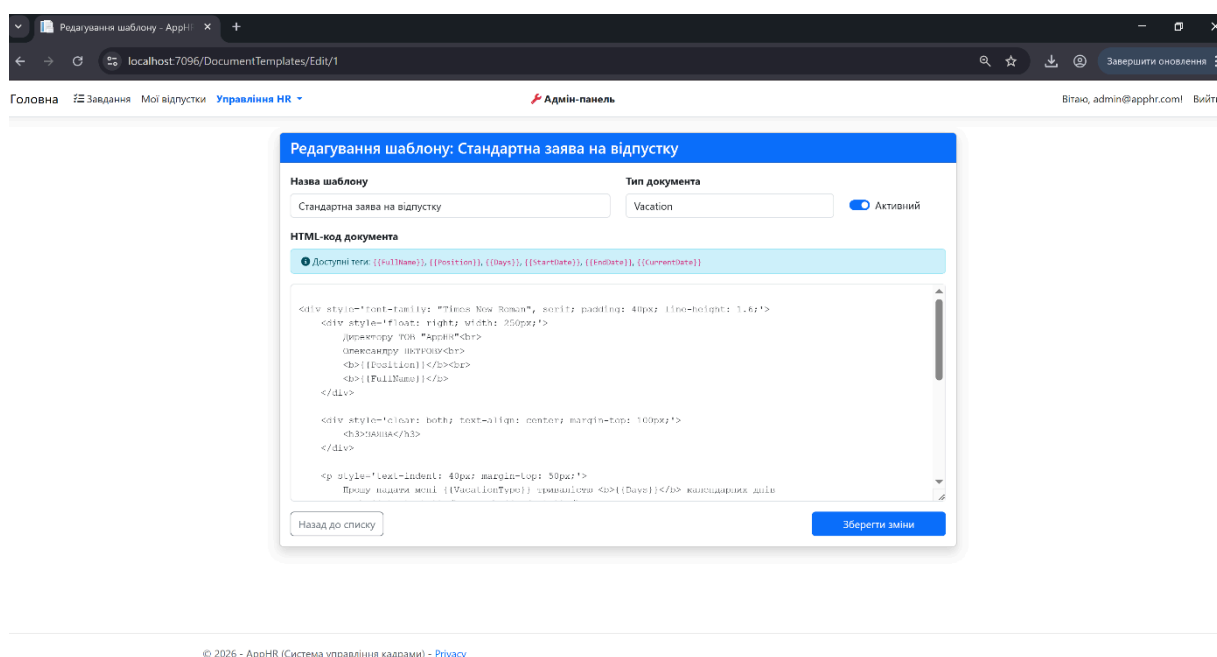


Рис. 3.3.1.1. Редагування шаблону заяви на відпустку

Формування документу варто розглядати як виконання наступних дій:

- Групування контексту, що відбувається поєднанням даних з бази даних та змінних даних процесу, отримані дані передаються в об'єкт передачі даних (OrderDto);
- Шаблони Razor, що дозволяє розділити візуальну верстку наказу від програмної логіки. Завдяки двигуну шаблонізації RazorLight, зміні з процесу (наприклад - FullName(ФІО співробітника)) можна додавати в HTML шаблон та генерувати документ. На рис. 3.3.1.2

продемонстровано список шаблонів кадрових документів, які адміністратор системи може редагувати.

- Безпечність потоків, що використовує у DI-контейнері (Program.cs) зареєстровано SynchronizedConverter як Singleton-сервіс, який забезпечує стабільну роботу конвертера у багатопотоковому середовищі вебсервера.

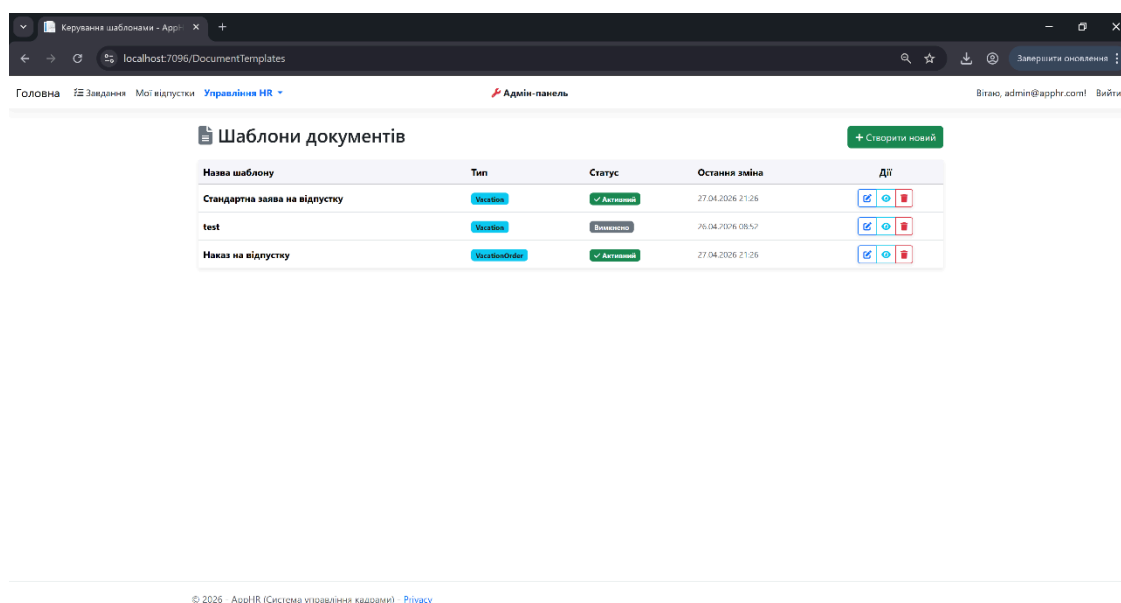


Рис. 3.3.1.2. Реєстр шаблонів кадрових документів у системі

3.3.2. Модуль цифрової верифікації та QR-кодування (QRCoder)

Щоб забезпечити захист кадрової документації від маніпуляцій додано модуль QR-ідентифікації на основі бібліотеки QRCoder[31]. Автономність цього інструменти, що створює матричні коди без застосування зовнішніх API, що забезпечують безпеку та автономність системи.

ТОВ "APPHR"

НАКАЗ № 24-B

08.05.2026 р.

Про надання відпустки

НАКАЗУЮ:

Надати **Хоменко Руслан Михайлович** що займає посаду **Інженер технічної підтримки(Support Engineer)**, за власний рахунок відпустку тривалістю **1** календарних днів з **26.04.2026** по **26.04.2026**.

Директор ТОВ "AppHR"
(підпис)

Олександр ПЕТРОВ

З наказом ознайомлений: _____ Хоменко Руслан Михайлович

ТОВ "AppHR"

Код ЄДРПОУ: 12345678



Скануйте для перевірки

Рис. 3.3.2.1. Наказ про відпустку з QRCode

Захист документу включає наступні кроки:

- Створення посилання перевірки, яке формує система з GUID наказу. За допомогою публічної точки доступу(endpoint) на яке веде посилання, здійснюється перевірка справжності документа.
- Математичне кодування, що виконується за допомогою класу QRCodeGenerator, який перетворює URL-адреса у графічний елемент з високим рівнем корекції помилок(ECCLevel.Q). Перед конвертацією наказу зображення додається в форму наказу. На рис.3.3.2.1. проілюстровано відображення QR коду в наказі.
- Механізм перевірки, що підтверджує справжність та актуальність документа шляхом сканування QR коду.

3.3.3. Модуль архітектурної синхронізації (Orchestration & State Sync)

Важливим завданням є відповідність між станами бізнес-процесу в Camunda BPM та записом в базі даних, завдяки принципу Eventual Consistency (узгодженість у кінцевому підсумку), де оновлення бази даних відбувається тільки після завершення операції в Camunda.

Взаємодія між компонентами відбувається наступним чином:

- Створення завершення, фінальні змінні процесу погодження передаються в метод `CompleteTaskAsync`, за допомогою звернення контролер .NET до Camunda REST API.
- Підтвердження від Camunda, виконується тільки коли вхідний статус 204 No Content, що вказує про успішне переміщення токена на наступний етап, тоді контролер змінює статус в SQL-базі через Entity Framework Core.
- Забезпечення відповідності, що забезпечується поточний порядок дій, вразі мережевого збою або помилки в Camunda, статус в базі даних не зміниться. у базі, хоча процес у Camunda ще не завершений або був перерваний.

Такий підхід забезпечує синхронність станів та високу надійність кадрового обліку при розподіленій архітектурі.

3.4. Реалізація бази даних та Entity Framework

Завдяки підходу Code-First, що був реалізований на базі Entity Framework Core[7], що дозволило розглядати базу даних як відображення доменної моделі проекту. Повна синхронізація між бізнес-логікою та фізичним сховищем була реалізована завдяки автоматичній генерації схеми на основі C# класів, що є важливим фактором для організації цілісності даних у динамічних кадрових процесах.

3.4.1. Управління міграціями та версіонування схеми

Для забезпечення еволюційного розвитку бази даних без ризику втрати інформації було впроваджено механізм міграцій. У контексті EF Core міграції виступають системою контролю версій для схеми БД, де кожна зміна зафіксована у вигляді програмного коду. На рис. 3.4.1.1 показано приклад міграції з додаванням таблиці логуювання – *AuditLogs*.

```
namespace AppHR.Migrations
{
    /// <inheritdoc />
    1 reference
    public partial class AddAuditLogUpdate : Migration
    {
        /// <inheritdoc />
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AuditLogs",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    UserId = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Type = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    TableName = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    DateTime = table.Column<DateTime>(type: "datetime2", nullable: false),
                    OldValues = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    NewValues = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    AffectedColumns = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    PrimaryKey = table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AuditLogs", x => x.Id);
                });
        }
    }
}
```

Рис. 3.4.1.1. Приклад коду міграції

Процес оновлення структури розділяється на такі типи:

- Декларативне моделювання. На рівні коду визначаються класи-сутності (Employee, VacationRequest), які інкапсулюють атрибути майбутніх таблиць.
- Створення трансформації. За допомогою інструкції *dotnet ef migrations add* створюється файл, що містить опис переходу між станами системи. Метод *Down()* забезпечую безпечне повернення до попередньої версії, а метод *Up()* вказує про зміни в нових елементах. На рис. 3.4.1.2. показаний приклад використання методів *Up()* та *Down()*.

```

1  using Microsoft.EntityFrameworkCore.Migrations;
2
3  #nullable disable
4
5  namespace AppHR.Migrations
6  {
7      /// <inheritdoc />
8      1 reference
9      public partial class AddCamundaTaskDto : Migration
10     {
11         /// <inheritdoc />
12         0 references
13         protected override void Up(MigrationBuilder migrationBuilder)
14         {
15         }
16
17         /// <inheritdoc />
18         0 references
19         protected override void Down(MigrationBuilder migrationBuilder)
20         {
21         }
22     }
23

```

Рис. 3.4.1.2. Приклад міграції з методами Up() та Down()

- Оновлення схеми бази даних, завдяки команді *database update* виконується трансформація інструкції на SQL запити, виконуючи їх у межах поточної транзакції.
- Механізм Model Snapshot. Entity Framework Core дозволяє підтримувати актуальний знімок моделі, завдяки цьому система може визначити різницю між кодом та БД без необхідності з'єднання з сервером баз даних.

3.4.2. Оперування даними за допомогою LINQ

Використовуючи технологію Language Integrated Query (LINQ)[11] для роботи з інформаційним шаром, було спрощено формування запитів до бази даних, що полегшило підтримку та читання коду. Приклади архітектурно важливих запитів:

- *Агрегація історії відпусток*: На рис. 3.4.2.1 продемонстровано запит, який дозволяє отримати впорядкований перелік усіх заявок поточного користувача для відображення в особистому кабінеті.

```

var userVacations = await _context.VacationRequests
    .Where(v => v.EmployeeId == currentUserId)

```

```
.OrderByDescending(v => v.StartDate)
    .ToListAsync();
```

Рис. 3.4.2.1. Приклад коду агрегації

- *Вибірка з випереджальним завантаженням (Eager Loading)*: На рис. 3.4.2.2. відображено використання методу `.Include()`, що дозволяє отримати дані про співробітника разом із заявкою за один цикл звернення до БД.

```
var approvedRequests = await _context.VacationRequests
    .Include(v => v.Employee)
    .Where(v => v.Status == "Approved"
        && v.StartDate >= DateTime.Now)
    .ToListAsync();
```

Рис. 3.4.2.2. Приклад коду вибірки з випереджальним завантаженням

Варто зазначити, що перевагами обраного підходу для HR-системи є:

- Відмова від діалекту SQL: Розробник оперує об'єктами, що нівелює ризик виникнення синтаксичних помилок у запитах.
- Гарантована безпека: LINQ автоматично використовує параметризацію запитів, що забезпечує абсолютний захист від SQL-ін'єкцій на рівні фреймворку[25].
- Вирішення проблеми N+1: Завдяки стратегії Eager Loading, система уникає надлишкових звернень до бази даних при роботі з пов'язаними сутностями. Це суттєво підвищує загальну продуктивність при формуванні великих списків або звітів.

Використання Entity Framework Core у поєднанні з LINQ дозволило створити гнучкий, безпечний та високопродуктивний рівень доступу до даних, який легко адаптується до змін у бізнес-вимогах проєкту[11].

3.4.3. Інтерфейсна реалізація маніпуляції даними

Для забезпечення управління статистичними даними та конфігураціями в системі реалізований адміністративний модуль, він є критичним для стабільного функціонування HR-процесів.

3.4.3.1. Адміністративний модуль керування

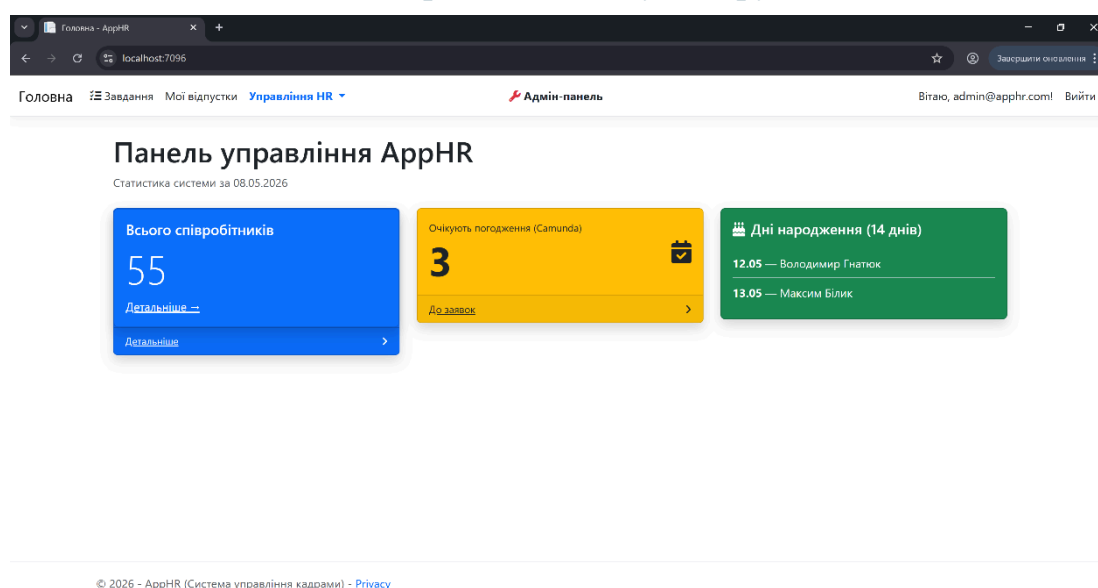


Рис. 3.4.3.1. Вид головної сторінки системи при вході адміністратора

Точкою входу для управління інфраструктурою системи є «**Адмін-панель**» (рис. 3.4.3.1), яка доступна адміністратору системи на головній сторінці. Вона реалізована за модульним принципом, де кожен блок відповідає за окрему площину нефункціональних та технічних вимог.

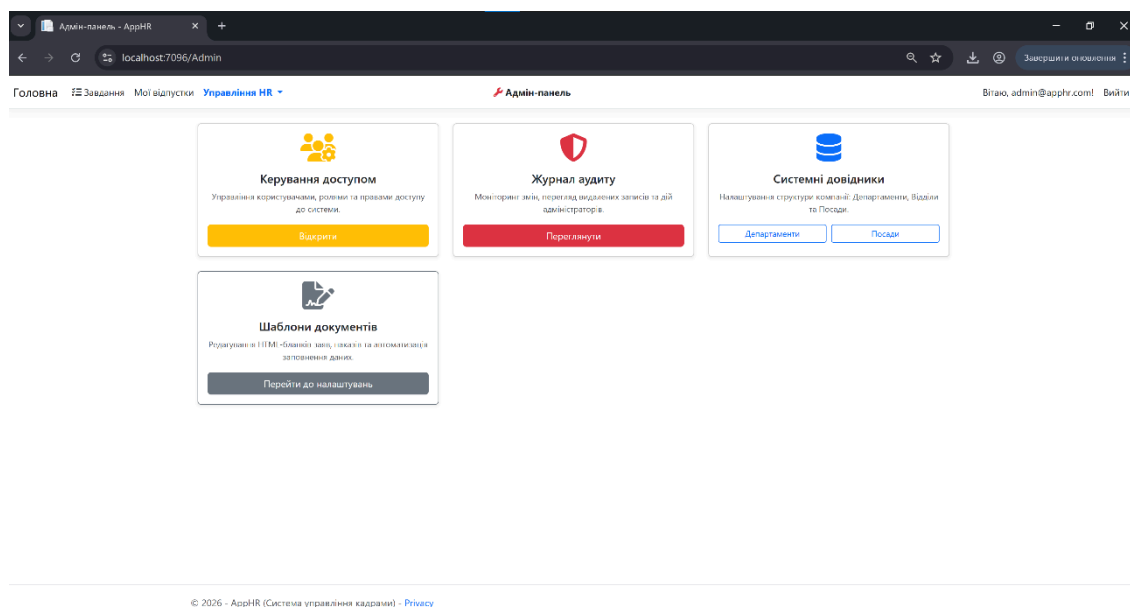


Рис. 3.4.3.2. Головний інтерфейс адміністративної панелі

Згідно з представленою ілюстрацією(рис. 3.4.3.2.), панель включає такі розділи:

- Керування доступом: Розділ в котрому реалізовано моделі *Role-Based Access Control* (RBAC) для адміністрування користувачів та їхніх ролей у системі.
- Журнал аудиту: модуль моніторингу, що відображає історію усіх операцій, включаючи створення, оновлення та видалення записів.
- Системні довідники: Інтерфейси для управління сутностями *Biddili*(*Departments*) та *Посади*(*Positions*), що формують організаційну структуру.
- Шаблони документів: Розділ для динамічного редагування HTML-бланків, що використовуються двигуном *RazorLight* для генерації PDF-наказів.

3.4.3.2. Управління організаційною структурою та довідниками

Зміна даних в системі доступна через інтерфейс, в формі реєстру(List Form) або в формі редагування (Edit Form). Це покращує орієнтацію у великих масивах та точність оновлення вибраних записів.

1. *Реєстр об'єктів (List Form)*. Представляє собою табличне відображення даних, де за допомогою LINQ-запитів із випереджальним завантаженням (Eager Loading) формується повний контекст сутності .

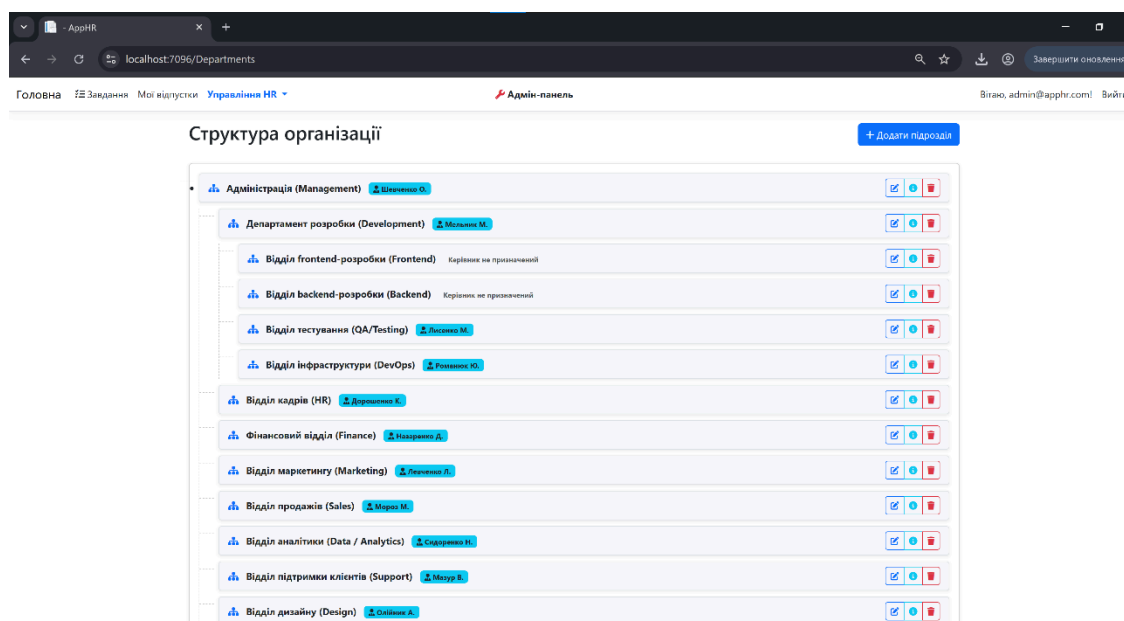


Рис. 3.4.3.2.1. Відображення списку відділів

На приклад, при перегляді списку відділів(рис. 3.4.3.2.1.) система одразу відображає дані про їхніх керівників та батьківські підрозділи, мінімізуючи кількість запитів до БД.

2. *Форма редагування (Edit Form)*. Процес оновлення сутності «Відділ» (Department) наочно демонструє роботу механізмів відстеження станів EF Core та інтеграцію з BPM-оркестратором .

Головна Завдання Мої відпустки **Управління HR** Адмін-панель Вітаю, admin@apphr.com! Вийти

Редагування

Оновлення даних відділу

Назва підрозділу
Відділ тестування (QA/Testing)

Підпорядковується підрозділу
Департамент розробки (Development)

Керівник підрозділу
Лисенко Марина (Керівник відділу тестування) (Head of Department)

Якщо керівника не обрано, заповніть поле на погодження в батьківський відділ

[Зберегти зміни](#) [Повернутися до списку](#)

© 2026 - AppHR (Система управління кадрами) [Приват](#)

Рис. 3.4.3.2. Форма редагування параметрів підрозділу

На рис. 3.4.3.2 продемонстровано екранну форму, де реалізовано наступну логіку мапінгу та валідації:

- Назва підрозділу: текстовий атрибут Name моделі Department, що підлягає обов'язковій перевірці на унікальність.
- Підпорядкованість: візуалізація рекурсивного зв'язку \$1:N\$ (один підрозділ може бути частиною іншого). Через випадаючий список обирається ParentDepartmentId, що дозволяє системі динамічно будувати ієрархічне дерево організації.
- Керівник підрозділу: вибір ідентифікатора співробітника (ManagerId). Це поле є ключовим для інтеграції з Camunda BPM: саме цей ідентифікатор автоматично стає цільовим Assignee для завдань типу User Task на етапі управлінського погодження.

Такий підхід дозволяє адміністратору змінювати структуру компанії у реальному часі без необхідності перепрограмування маршрутів погодження,

оскільки оркестратор завжди звертається до актуального ManagerId, зафіксованого у базі даних SQL Server .

3.4.3.3. Модуль керування доступом та безпекою

Важливим компонентом адміністративного модуля є модуль керування доступом, який забезпечує реалізацію моделі RBAC (Role-Based Access Control). Технічна реалізація цього модуля базується на інтеграції ASP.NET Core Identity із реляційним сховищем через Entity Framework Core, що дозволяє гнучко налаштовувати привілеї для кожного суб'єкта системи.

3.4.3.3.1. Рольова детермінація та ієрархія прав

Згідно з архітектурним проєктом, доступ до функцій системи розмежовано між чотирма основними класами користувачів:

- Співробітник (Employee): має обмежений доступ виключно до власних кадрових запитів та персональних звітів.
- Керівник (Manager): володіє правами на перегляд та валідацію завдань погодження (User Tasks), ініційованих підлеглими його підрозділу.
- HR-менеджер: має розширені права на роботу з довідниками, реєстрацію наказів та аудит лімітів відпусток у масштабах всієї організації.
- Адміністратор (Admin): володіє повним доступом до системних конфігурацій, журналу аудиту та управління обліковими записами.

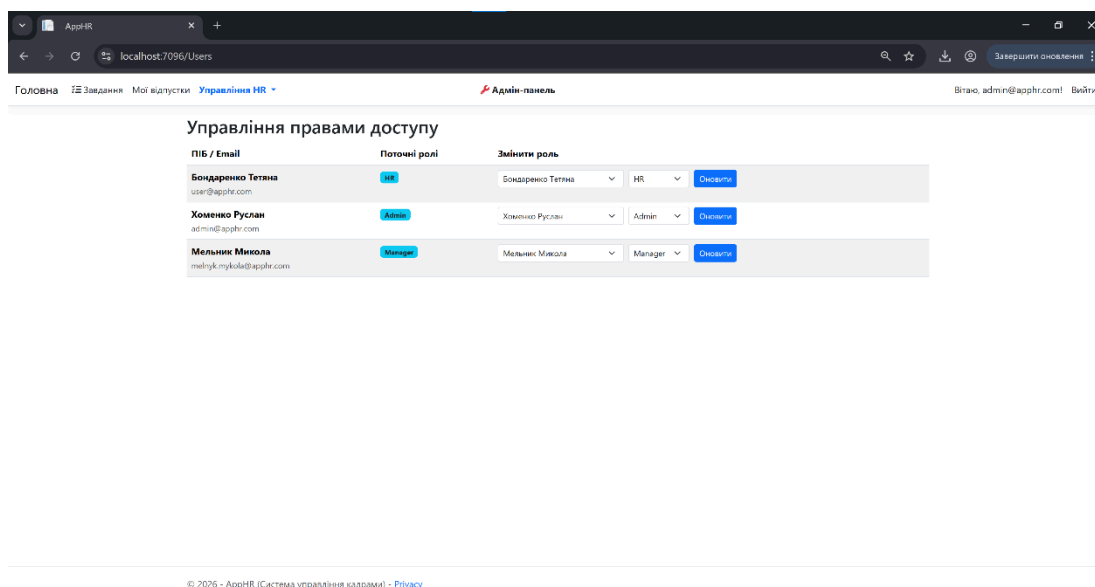


Рис. 3.4.3.3.1. Керування обліковими записами та автентифікація

Інтерфейс «*Керування доступом*» (доступний через головну Адмін-панель, рис. 3.4.3.1) дозволяє адміністратору виконувати реєстрацію нових співробітників, скидання паролів та миттєве блокування облікових записів. Процес автентифікації захищено механізмами JWT (JSON Web Token) або сесійними куками, що гарантує безпеку сеансу та захист від несанкціонованого перехоплення даних.

3.4.3.3.3. Захист персональних даних (РП)

Система оперує конфіденційною інформацією, тому модуль безпеки реалізує принцип "необхідного знання", що на рівні програмної реалізації забезпечується такими методами:

- Шифрування. На рівні бази даних персональні дані працівників потрібно шифрувати для запобігання витоку інформації у разі доступу до сховища.
- Обмеження доступу до завдань, що передбачає видимість завдання тільки визначеному виконавцю (Assignee), та унеможлиблює погодження заяви іншою особою.

- Аудит дій: будь-яка зміна ролі чи спроба доступу до адміністративних функцій фіксується у «Журналі аудиту» (див. рис. 3.5.3.1), що дозволяє відстежувати активність користувачів у ретроспективі .

Дата	Користувач	Дія	Таблиця	Об'єкт (ID)	Деталі змін
08.05.2026 21:09:27	admin@apphr.com	Створення	AspNetUsers	Запис ID: 647a813-3d7f-4d22-8fde-bd85ce0934c	ConcurrencyStamp: 0914e202-740e-400a-8004-e09607c2c4bd → e35280e8-6140-4795-8740-7d86738f93eb
08.05.2026 21:09:27	admin@apphr.com	Створення	AspNetUserRoles	Запис ID:	Запис додано до бази даних
08.05.2026 21:09:27	admin@apphr.com	Вилучення	AspNetUserRoles	Запис ID:	Посилання видалено
08.05.2026 21:09:27	admin@apphr.com	Створення	AspNetUsers	Запис ID: 647a813-3d7f-4d22-8fde-bd85ce0934c	ConcurrencyStamp: b16e603a-c5ed-4004-8004-d4f621774bc9 → 0981a220-a346-406a-8f04-a690572c14bd
29.04.2026 20:57:22	System	Створення	VacationRequests	Запис ID: 27	Статус: 1 → 0
29.04.2026 20:50:05	System	Створення	VacationRequests	Запис ID: 26	Статус: 1 → 0
29.04.2026 20:40:47	admin@apphr.com	Створення	VacationRequests	Запис ID: 27	ConcurrencyStamp: 0914e202-740e-400a-8004-e09607c2c4bd → 0981a220-a346-406a-8f04-a690572c14bd
29.04.2026 20:40:47	admin@apphr.com	Створення	VacationRequests	Запис ID: 247482647	Запис додано до бази даних
27.04.2026 21:26:24	admin@apphr.com	Створення	DocumentTemplates	Стандартна заявка на відпустку ID: 1	Значення поля (ідентичні (опціональні) записи)
27.04.2026 21:26:08	admin@apphr.com	Створення	DocumentTemplates	Наказ на відпустку ID: 5	Значення поля (ідентичні (опціональні) записи)
27.04.2026 21:25:54	admin@apphr.com	Створення	DocumentTemplates	Стандартна заявка на відпустку ID: 1	Значення поля (ідентичні (опціональні) записи)

Рис. 3.4.3.3.3.1. Журнал аудиту системи

Треба зазначити, що модуль контролю доступу спрощує роботу адміністраторам - і водночас стає підтвердженням того, що система дотримується українських правил захисту персональних даних[3], як і внутрішніх норм безпеки компанії.

3.5. Перевірка та оцінка якості системи

Одним з ключових етапів циклу розробки є перевірка програмного забезпечення (ПЗ) на відповідність архітектурним та функціональним специфікаціям, що були визначені раніше в роботі. Процес тестування повинний підтвердити надійність системи при інтенсивній експлуатації та перевірки її на помилки.

3.5.1. Функціональна верифікація та аналіз тестових сценаріїв

Для тестування критичних вузлів бізнес-логіки було застосовано сценарне тестування, в якому особлива увага була приділена граничним значенням (Boundary Value Analysis) та обробці негативних кейсів[28].

Таблиця 3.5.1.1

Реєстр виконання ключових тестових сценаріїв

№	Опис сценарію	Очікувана реакція системи	Стан
1	Спроба ініціації відпустки з ретроспективною датою.	Блокування транзакції на рівні валідатора з виведенням повідомлення: «Дата початку не може передувати поточній».	Успішно
2	Формування запиту, тривалість якого перевищує VacationBalance.	Превентивна деактивація керуючих елементів інтерфейсу; виведення попередження про дефіцит квоти днів.	Успішно
3	Позитивне рішення керівника через інтерфейс «Мої завдання».	Трансформація статусу запису на «Pending HR»; коректна передача процесного токена в наступний Swimlane.	Успішно
4	Активація сценарію відхилення (Rejection flow).	Термінація процесу; зміна статусу в SQL-базі на «Rejected»; відправка сповіщення ініціатору.	Успішно
5	Генерація фінального артефакту (наказу) у форматі PDF.	Рендеринг документа з верифікацією даних та інтеграцією динамічного QR-ідентифікатора.	Успішно

3.5.2. Експертна оцінка користувацького інтерфейсу (UI/UX)

Якість інтерфейсу перевірялася через ергономіку та забезпечення безбар'єрної взаємодії користувача із системою.

У результаті верифікації підтверджено наступні аспекти:

- Toast-сповіщень: Відображають миттєве повідомлення на дії користувача, що знижує рівень когнітивного навантаження при роботі зі складними формами.

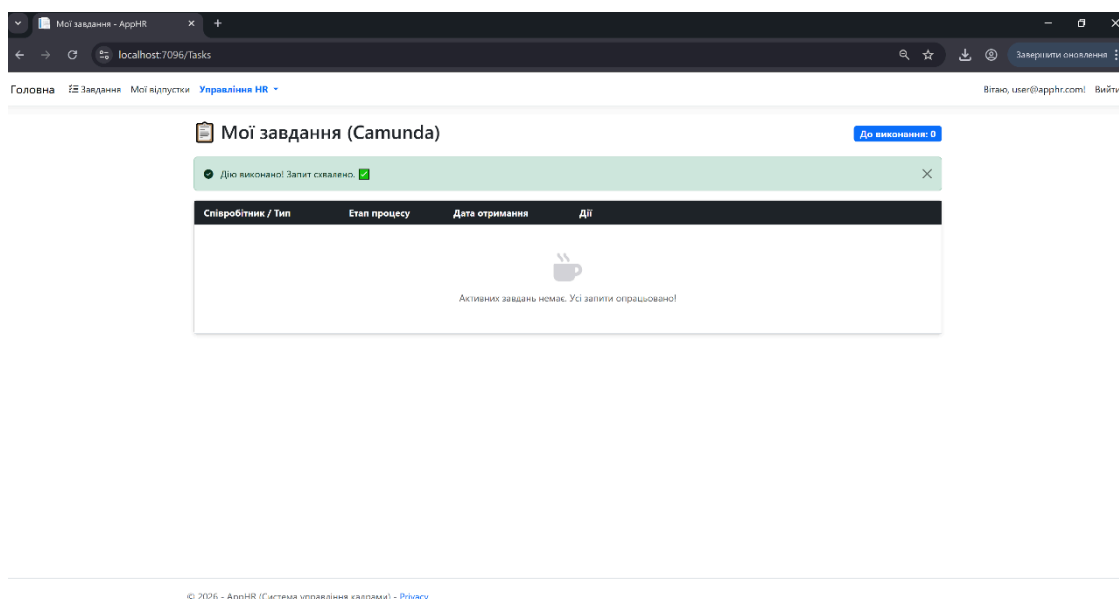


Рис. 3.5.2.1. Приклад Toast-сповіщень при виконанні завдання

- Кольорова візуалізація: Використання усталених візуальних патернів (зелений — акцептовано, червоний — відмовлено, жовтий — очікування) дозволяє користувачу ідентифікувати статус документа на інтуїтивному рівні, мінімізуючи час на аналіз табличних даних.

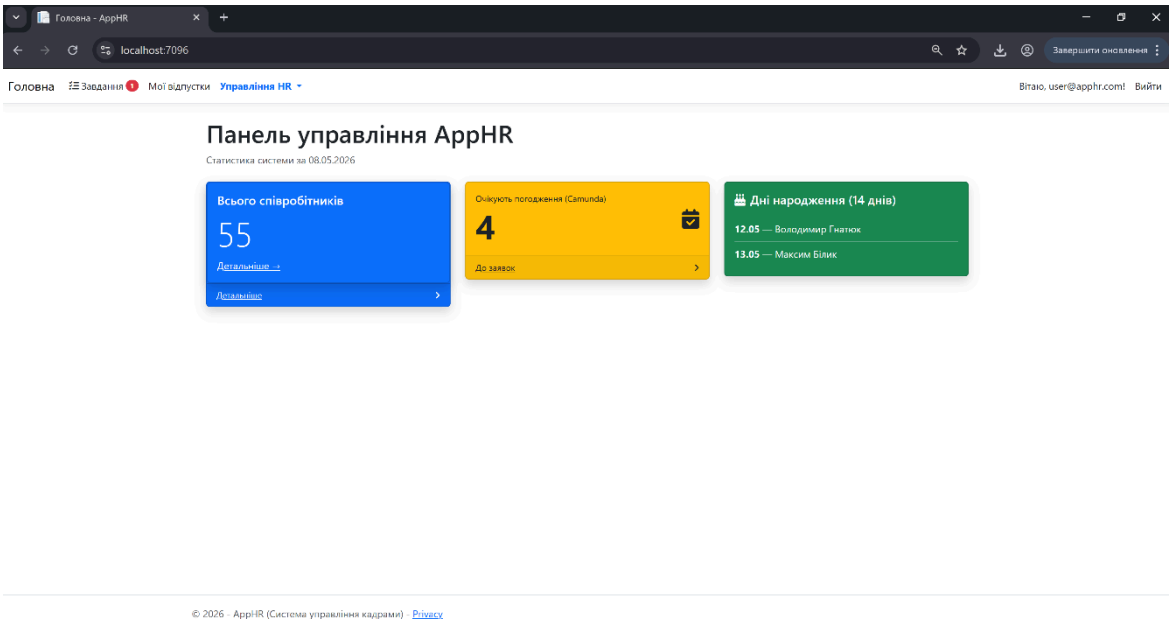


Рис 3.5.2.2. Головна панель управління (Dashboard) для моніторингу стану HR-процесів

На рис. 3.5.2.2. продемонстровано приклад семантичного кольорового кодування, що ілюструє менеджеру наступні дані: синій блок (загальна статистика), жовтий (очікування) та зелений (позитивні події/дні народження).

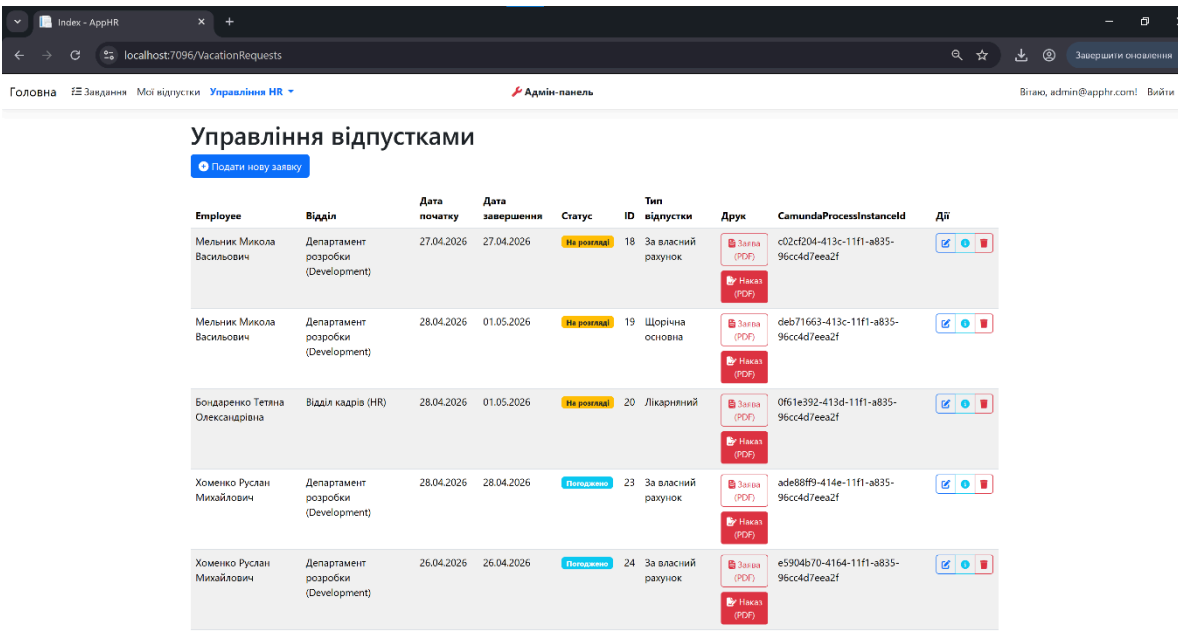


Рис. 3.5.2.3. Відображення статусів заявок на відпустку

- Кроссплатформна адаптивність: Тестування в умовах варіативності в'юпортів (viewports) підтвердило життєздатність гнучкої верстки. Коректне

відображення на мобільних пристроях є критичним для керівної ланки, що забезпечує оперативність погоджень поза межами стаціонарного робочого місця.

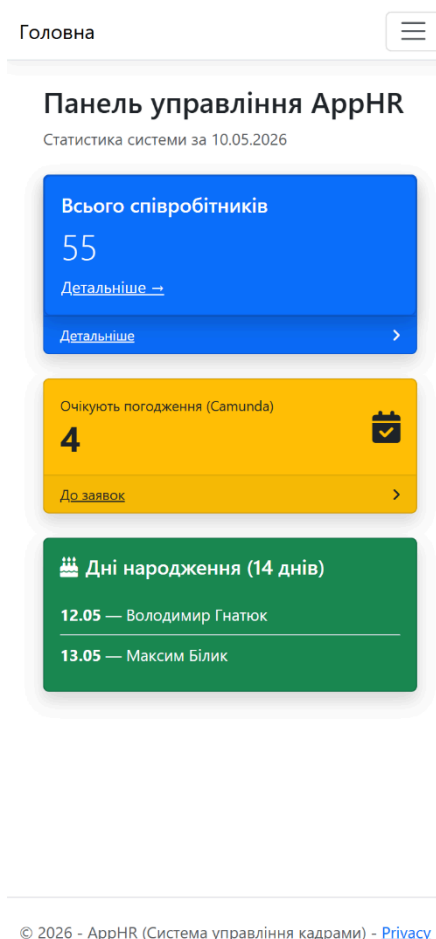


Рис. 3.5.2.4. Відображення системи на мобільних пристроях

3.5.3. Аудит станів та детермінізм бізнес-процесу (State Machine)

Високий рівень детермінізму при переході між станами запиту було досягнуто завдяки інтеграції з платформою Camunda. У ході системного аналізу було зафіксовано наступні результати:

- Чітка послідовність, що забезпечується архітектурою процесу та не дозволяє логічні стрибки статусів. Статус «Order» настає тільки при успішному

- завершені процесу, що гарантує суворе дотримання корпоративних регламентів.
- Синхронна постійність: Реалізовано механізм синхронного оновлення даних, завершення User Task у Camunda жорстко пов'язане з успішною фіксацією змін у SQL-сховищі. Завдяки цьому виключається виникнення станів не синхронізації, де процес завершено, але не змінено у фінансовій чи кадровій звітності[4].
 - *Глибина аудиту*: Використання Camunda Cockpit підтвердило наявність детального «цифрового сліду» (audit trail) для кожного екземпляра процесу. Це дозволяє в реальному часі ідентифікувати «вузькі місця» в ланцюжку погоджень та підтверджує повну відповідність фактичної роботи системи спроектованій BPMN-моделі.

Після всебічного аналізу тестових даних видно - система працює без збоїв. Хоча перевірки тривали довго, ніяких прогалин у захисті не виявили. Тепер її можна запускати на реальних потоках роботи. Навіть при підвищеному навантаженні стан залишається передбачуваним.

3.6. Висновки до третього розділу

В цьому розділі було проведено розгляд практичного виконання рішення, який дозволив підтвердити, що теоретичний та архітектурний задум був виконаний в повному обсягу. Проведене тестування системи дозволило сформулювати наступні висновки:

- Засвідчення функціональних вимог, що було підтверджене наступним функціоналом: керування повним циклом заяв від створення співробітником до формування наказу; координація та розподіл завдань між виконавцями в системі; формування документів з шаблонів з необхідною ідентифікацією їх справжності.

- Модульність системи, яке включає в себе розділення відповідальності між компонентами (бізнес-логіки та сервісної логіки) забезпечило гнучкість архітектури, що суттєво спрощує подальше масштабування та модифікацію продукту без ризику дестабілізації ядра.
- Безпека даних, що забезпечується завдяки Entity Framework Core[7] та LINQ-запитів вдалося створити стійкий рівень збереження даних, впровадження міграції даних надало цілісність структури бази даних, а вбудована параметризація запитів виступила надійним бар'єром проти SQL-ін'єкцій. Ці впровадження забезпечують стабільність роботи з конфіденційною кадровою інформацією та нівелюють ризики втрати даних.
- Якість даних, що забезпечує стійкість системи до значних навантажень та унеможливорює порушення логічної послідовності кадрових операцій.
- Операційна ефективність системи виражається в реалізованих інструментах автоматизації(таких як Service Tasks або Timer Events), що дозволяє скоротити тривалість циклу погодження; мінімізувати людський фактор на етапах перевірки та створення документів.

В підсумку можна зробити наступний висновок, впроваджена система є завершеним продуктом, що може бути впроваджена у будь-якій компанії, що дозволить цифровізувати кадрові процеси.

ВИСНОВКИ

Під час виконання проекту було сформовано наступний підсумок комплексного аналізу, проектування та впровадження стратегії автоматизації менеджменту. При виконанні роботи було перевизначено роль сучасних HR-технологій, також це дозволило визначити обмеження наявних рішень, а також сформулювати вимоги для побудови інформаційної системи, що здатна ефективно працювати з бізнес-вимогами.

В результаті проведеної роботи є веб застосунок, який переводить кадрові процеси в цифровий формат. Втілений функціонал охоплює повний операційний цикл - від обліку персоналу до погодження запитів о відпустках. Реалізація механізмів перевірки за допомогою QR-ідентифікаторів забезпечило цілісність та правомірність електронного документообігу, що дозволить знизити ризики, що може бути при паперовому діловодстві.

Науковою новизною роботи є рішення архітектурної моделі, яка поєднує платформу .NET та BPM Camunda. Таке рішення розділяє бізнес-логіку та процедурну логіку, що забезпечує гнучкість при зміні законодавства, та дозволить безпечно масштабувати продукт. Використання нотацій BPMN 2.0 переводить бізнес-правила в самодокументований актив, що можливо в подальшому автоматизувати[19].

Практичної цінністю цього проекту є програмний продукт, що адаптований до потреб компаній різного масштабу- від стартапу до великих підприємств. Ефективність рішення є зміна бізнес-логіки в редакторі Camunda, що не потребує зміни в коді системи.

Шлях що був подоланий автором при розробці цієї роботи, від моделювання та проектування моделей даних, інтеграцією сценаріїв взаємодії .NET середовища з процесним рішенням, розробці генерації документів, додав добрий внесок до майбутнього розвитку даної системи[12].

В підсумку, треба зауважити що завдання, які ставилися перед виконанням роботи, виконані в повному обсязі, а мета дослідження - була досягнута. Розроблена система є функціонально завершеною та готовою до використання. Подальшим розвитком системи є: розширення функціоналу документообігу, зокрема інтеграція в систему механізмів електронного підпису, це надасть системі юридичної ваги в подальшій інтеграції кадрових процесів; додавання інтеграції з іншими корпоративними системами, що дозволить проводити обміном даних між системами; додавання каналів інформування, що дозволить надсилати співробітникам інформування про нові завдання в системі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кодекс законів про працю України: Закон України від 10.12.1971 № 322-VIII (зі змінами та доповненнями станом на 2026 р.).
2. Про затвердження типових форм первинної облікової документації зі статистики праці: Наказ Державного комітету статистики України від 05.12.2008 № 489.
3. Про захист персональних даних: Закон України від 01.06.2010 № 2297-VI.
4. Ambler S. W. Refactoring Databases: Evolutionary Database Design. Addison-Wesley Professional, 2023. 384 p.
5. Dumas M., La Rosa M., Mendling J., Reijers H. A. Fundamentals of Business Process Management. 2nd ed. Springer, 2018. 527 p.
6. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2012. 533 p.
7. Freeman A. Pro ASP.NET Core 8. 10th ed. Apress, 2024. 1150 p.
8. Freund J., Rücker B. Real-Life BPMN: Using BPMN 2.0 to Analyze, Improve, and Automate Processes in Your Company. Camunda Services GmbH, 2019. 284 p.
9. Harmon P. Business Process Management: A Manager's Guide to Strategy and Implementation. Morgan Kaufmann, 2019. 512 p.

10. Humphrey W. S. Introduction to the Team Software Process. Addison-Wesley, 2025. 490 p.
11. Lerman J., Miller B. Programming Entity Framework: DbContext. O'Reilly Media, 2020. 250 p.
12. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
13. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Wiley, 2011. 240 p.
14. Porter M. E. Competitive Advantage: Creating and Sustaining Superior Performance. Free Press, 2021. 592 p.
15. Price M. J. C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals. Packt Publishing, 2023. 812 p.
16. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. 432 p.
17. Silver B. BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0. Cody-Cassidy Press, 2011. 236 p.
18. Weske M. Business Process Management: Concepts, Languages, Architectures. Springer, 2019. 415 p.
19. Camunda Documentation. Camunda 7 REST API Reference & External Tasks. Official Technical Guides, 2025. URL: <https://docs.camunda.org/>
20. Deloitte. Human Capital Trends: The shift towards adaptive HR technologies. Deloitte Insights, 2023.
21. Gartner. Magic Quadrant for Cloud HCM Suites for 1,000+ Employee Enterprises. 2023.
22. Grand View Research. Human Resource Management Market Size, Share & Trends Analysis Report, 2024–2030. 2023.

23. ISO/IEC 18004:2015. Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification.
24. Microsoft Learn. Data Consistency Patterns in Distributed Systems. Cloud Design Patterns, 2025.
25. Microsoft Learn. Dependency injection in ASP.NET Core. Official .NET Technical Documentation, 2025.
26. Microsoft Learn. Design for Security and Roles in ASP.NET Core. 2025.
27. Microsoft. Dynamics 365 Human Resources: Modernizing HR operations. Official Documentation, 2024.
28. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group, 2020.
29. Object Management Group. Unified Modeling Language (UML) Specification. Version 2.6, 2024. URL: <https://www.omg.org/spec/UML/>
30. DinkToPdf Documentation. C# .NET wrapper for wkhtmltopdf library. GitHub Repository, 2024. URL: <https://github.com/rdvojmoc/DinkToPdf>
31. QRCoder library. Pure C# Open Source QR Code implementation. GitHub Repository, 2025. URL: <https://github.com/codebude/QRCoder>
32. wkhtmltopdf documentation. Command line tools to render HTML into PDF using the Qt WebKit rendering engine. Official Technical Website, 2024. URL: <https://wkhtmltopdf.org/>

Додатки

Додаток А

Графічний інтерфейс користувача та процесні моделі

А.1 Екранна форма авторизації

The screenshot shows a web browser window with the title 'Авторизація - AppHR'. The address bar displays 'localhost:7096/Identity/Account/Login?ReturnUrl=%2F'. The main content area features a login card with a blue header. The header contains a white user icon, the text 'Вхід у систему', and the subtitle 'AppHR — Управління відпустками'. Below the header, there are two input fields: 'Електронна пошта' (Email) with the value 'admin@apphr.com' and 'Пароль' (Password) with masked characters '.....'. A checkbox labeled 'Remember me?' is positioned below the password field. A large blue button with the text 'Увійти' (Login) and a right-pointing arrow is located below the checkbox. At the bottom of the card, there are two links: 'Забули пароль?' (Forgot password?) and 'Створити акаунт' (Create account).

Авторизація - AppHR

localhost:7096/Identity/Account/Login?ReturnUrl=%2F



Вхід у систему

AppHR — Управління відпустками

Електронна пошта
admin@apphr.com

Пароль
.....

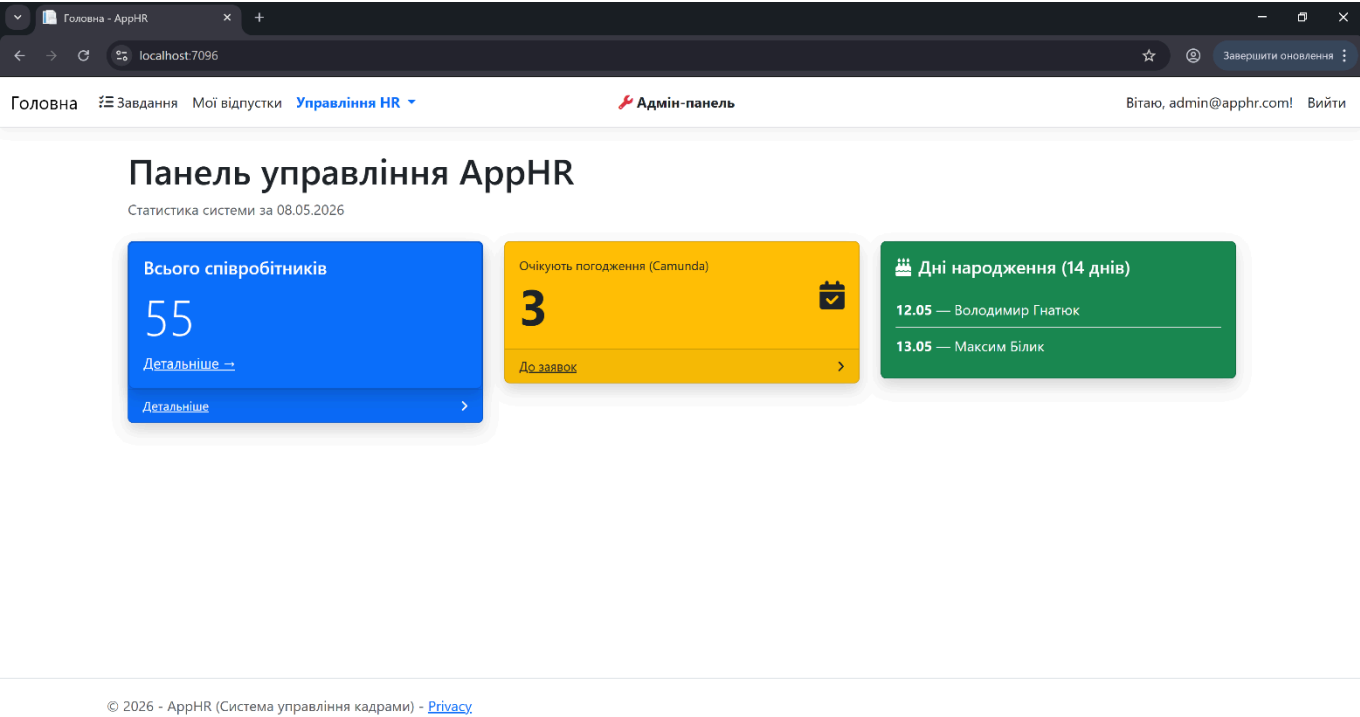
☐ Remember me?

Увійти ➔

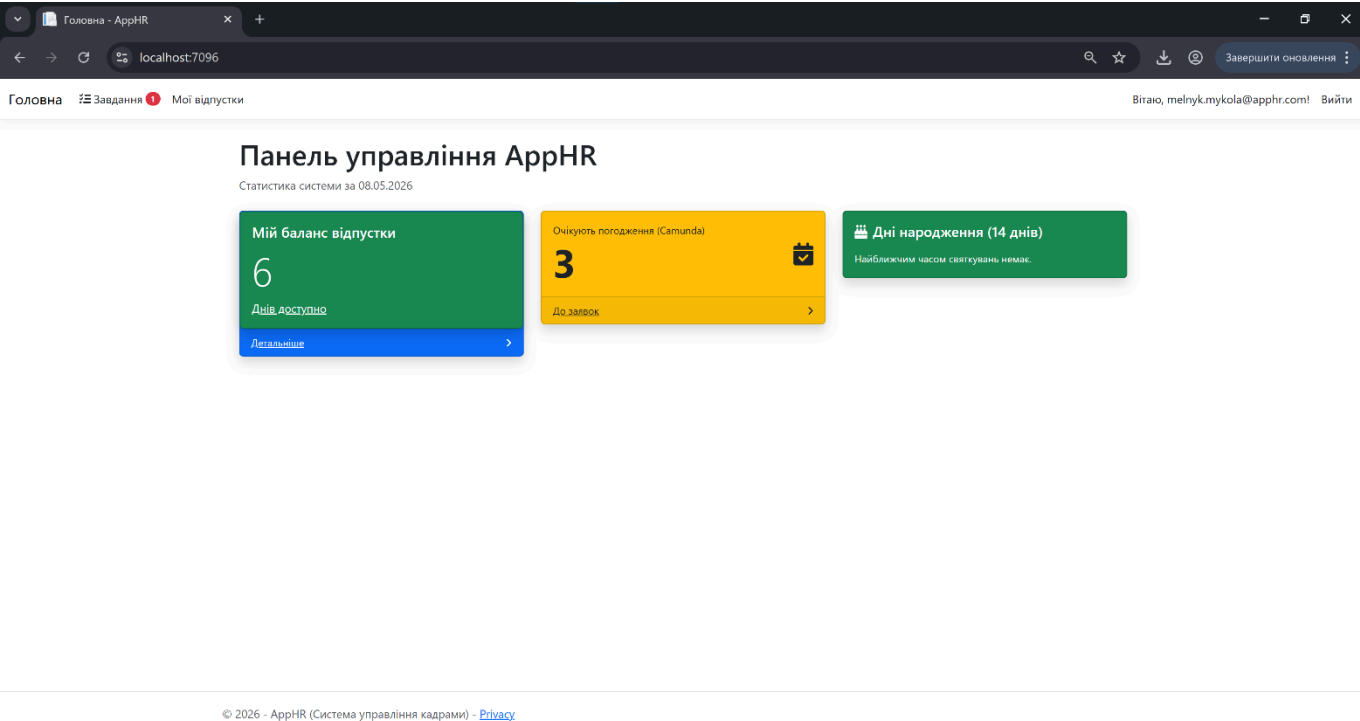
[Забули пароль?](#)

[Створити акаунт](#)

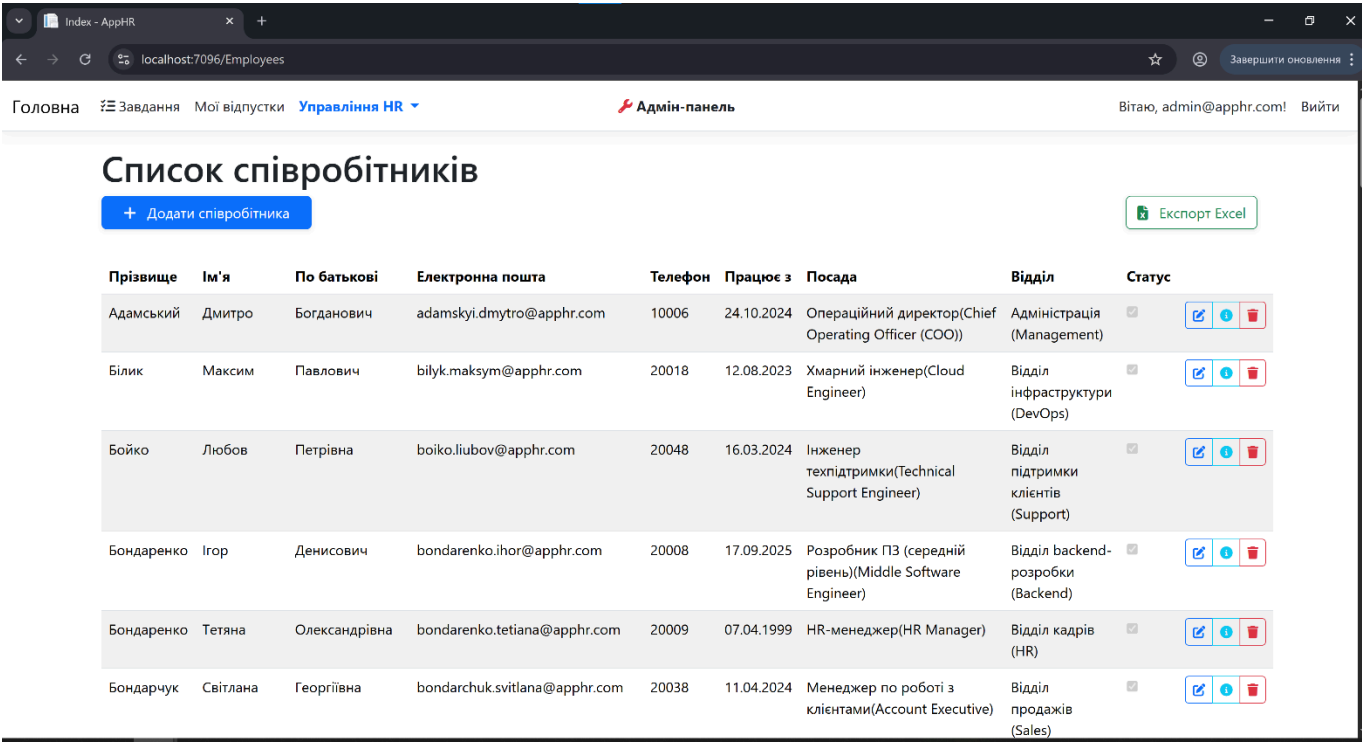
А.2 Головна сторінка



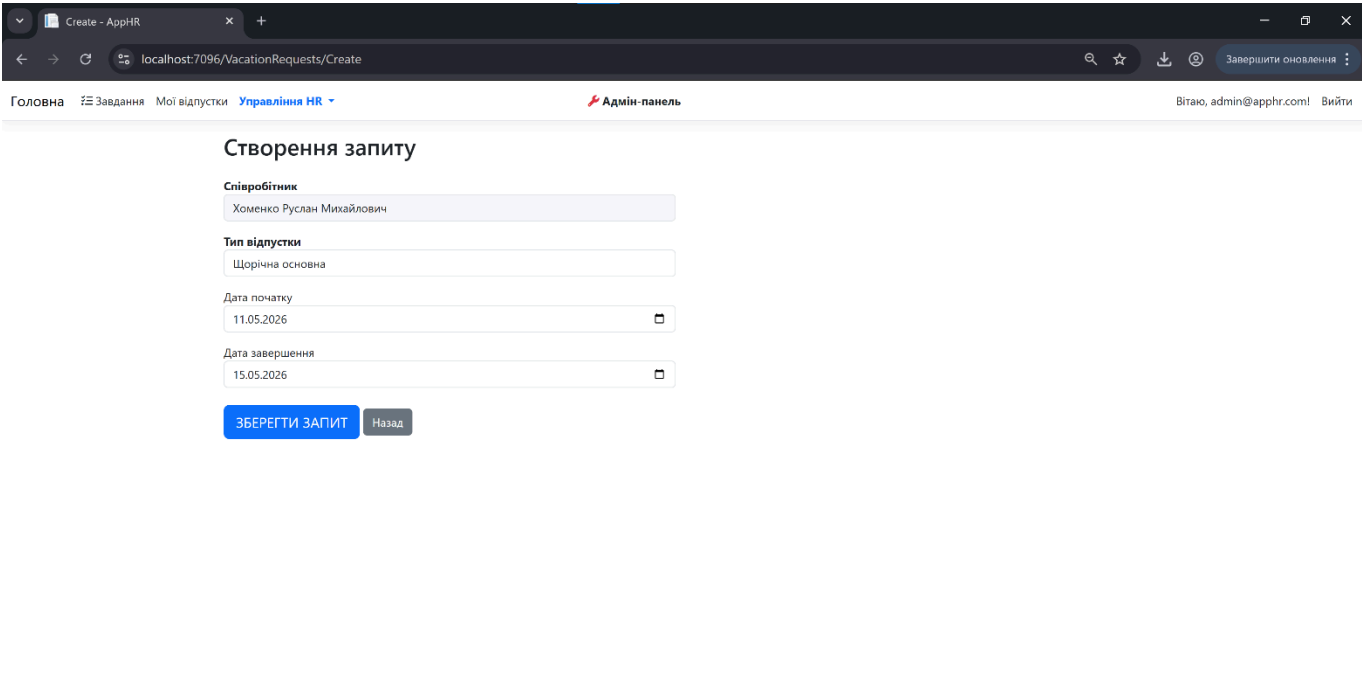
A.3 Інтерфейс особистого кабінету співробітника



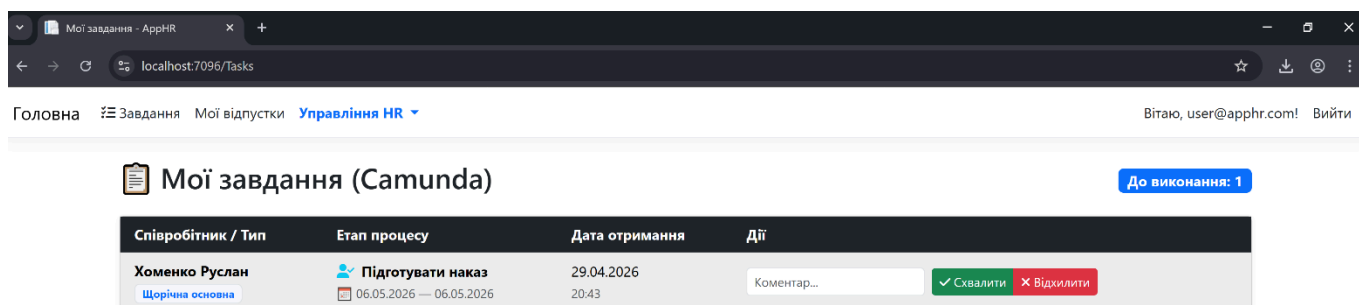
А.4 Реєстр персоналу



А.5 Екранна форма створення запиту на відпустку

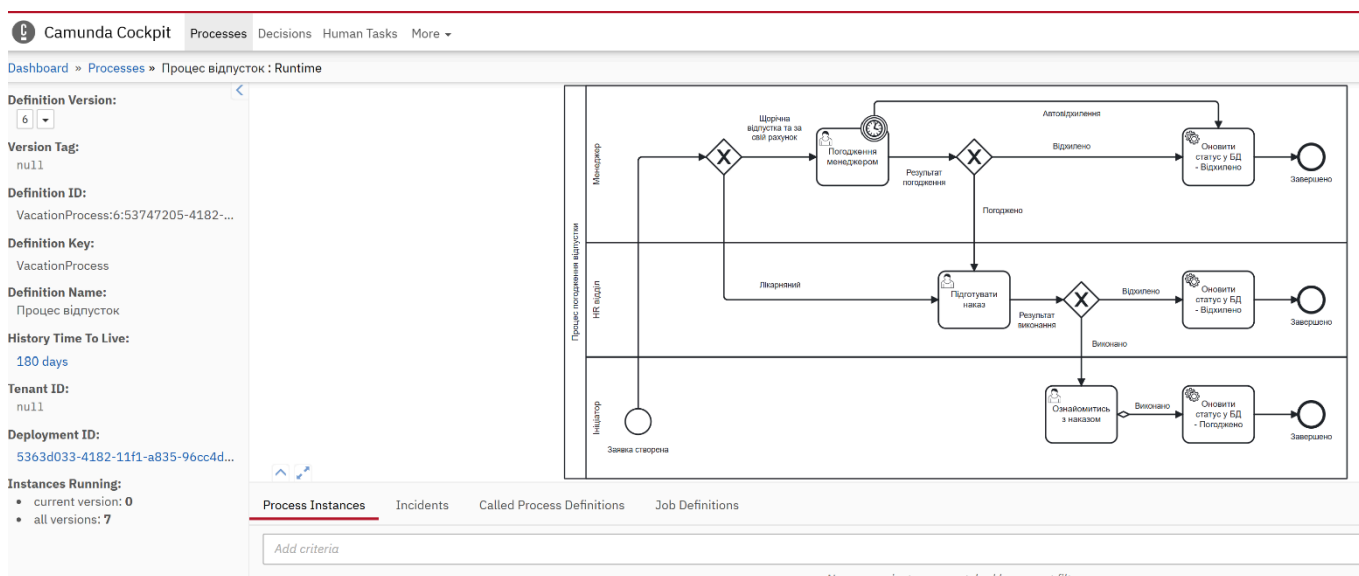


A.6 Інтерфейс менеджера для багаторівневого погодження кадрових документів



Співробітник / Тип	Етап процесу	Дата отримання	Дії
Хоменко Руслан	Підготувати наказ	29.04.2026	Коментар...
Щорічна основна	06.05.2026 — 06.05.2026	20:43	✓ Схвалити ✗ Відхилити

A.7 Модель бізнес-процесу погодження відпусток у нотації BPMN 2.0 (Camunda Modeler)



ДОДАТОК Б

Б.1 Конфігурація конвеєра HTTP-запитів та реєстрація сервісів(Program.cs)

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.InteropServices;
using AppHR.Data;
using AppHR.Services;
using DinkToPdf;
using DinkToPdf.Contracts;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.EntityFrameworkCore;

// --- ОТРИМУЄМО ШЛЯХ ДО DLL ТА ЗАВАНТАЖУЄМО ЇЇ ЗА ДОПОМОГОЮ ДОПОМІЖНОГО КЛАСУ ---
var dllPath = Path.Combine(Directory.GetCurrentDirectory(), "libwkhtmltox.dll");
var pdfLoadContext = new CustomAssemblyLoadContext();
pdfLoadContext.LoadUnmanagedLibrary(dllPath);

// --- ІНІЦІАЛІЗАЦІЯ BUILDER ---
var builder = WebApplication.CreateBuilder(args);

// --- РЕЄСТРАЦІЯ СЕРВІСІВ (DI CONTAINER) ---
builder.Services.Configure<CookiePolicyOptions>(options =>
{
    options.MinimumSameSitePolicy = SameSiteMode.Lax;
    options.HttpOnly = Microsoft.AspNetCore.CookiePolicy.HttpOnlyPolicy.Always;
    options.Secure = CookieSecurePolicy.Always;
});

builder.Services.AddAntiforgery(options =>
{
    options.Cookie.SecurePolicy = CookieSecurePolicy.Always;
    options.Cookie.Name = "AntiforgeryCookie";
});

// --- НАЛАШТУВАННЯ ДЛЯ IDENTITY ---
builder.Services.ConfigureApplicationCookie(options =>
{
    options.Cookie.SecurePolicy = CookieSecurePolicy.Always;

```

```

    options.Cookie.SameSite = SameSiteMode.Lax;
});
builder.Services.Configure<CookieTempDataProviderOptions>(options =>
{
    options.Cookie.SecurePolicy = CookieSecurePolicy.Always;
});
// --- ПЕЄСТРУЄМО PDF КОМБЕПТЕР ---
builder.Services.AddSingleton(typeof(IConverter), new SynchronizedConverter(new PdfTools()));

// --- КОНТЕКСТ БД ---
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// --- НАЛАШТУВАННЯ IDENTITY ---
builder.Services.AddDefaultIdentity<IdentityUser>(options => {
    options.SignIn.RequireConfirmedAccount = false;
    options.Password.RequireDigit = false;
    options.Password.RequiredLength = 4;
}).AddRoles<IdentityRole>()
.AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddHttpContextAccessor();

// --- CAMUNDA СЕРВІС ---
builder.Services.AddScoped<AppHR.Services.ICamundaService, AppHR.Services.CamundaService>();

// --- КОНТРОЛЕРИ ТА RAZOR PAGES З ГЛОБАЛЬНОЮ АВТОРИЗАЦІЄЮ ---
builder.Services.AddControllersWithViews(options =>
{
    var policy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build();
    options.Filters.Add(new AuthorizeFilter(policy));
});
builder.Services.AddRazorPages();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.Use(async (context, next) =>
{
    context.Response.Headers.Add("Content-Security-Policy", "script-src 'self' 'unsafe-inline' 'unsafe-eval'
https://code.jquery.com https://cdn.datatables.net;");
});

```

```

    await next();
});

app.UseHttpsRedirection();
app.UseCookiePolicy();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();

// --- ІНІЦІАЛІЗАЦІЯ ДАНИХ (РОЛІ, АДМІН, ШАБЛОНИ) ---
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var context = services.GetRequiredService<AppDbContext>();
        var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
        var userManager = services.GetRequiredService<UserManager<IdentityUser>>();

        // --- ІНІЦІАЛІЗАЦІЯ ШАБЛОНІВ ДОКУМЕНТІВ З БД --
        DbInitializer.Initialize(context);

        // --- СТВОРЕННЯ РОЛЕЙ ---
        string[] roleNames = { "Admin", "HR", "Manager", "Employee" };
        foreach (var roleName in roleNames)
        {
            if (!await roleManager.RoleExistsAsync(roleName))
                await roleManager.CreateAsync(new IdentityRole(roleName));
        }

        // --- НАЛАШТУВАННЯ ГОЛОВНОГО АДМІНА ---
        var adminUser = await userManager.FindByEmailAsync("admin@apphr.com");
        if (adminUser != null && !await userManager.IsInRoleAsync(adminUser, "Admin"))
        {
            await userManager.AddToRoleAsync(adminUser, "Admin");
        }

        // --- АВТОМАТИЧНЕ ПРИЗНАЧЕННЯ РОЛЕЙ ІНШИМ КОРИСТУВАЧАМ ---
        var allUsers = await userManager.Users.ToListAsync();
        foreach (var user in allUsers)
        {
            var userRoles = await userManager.GetRolesAsync(user);
            if (userRoles.Count == 0)

```

```

        {
            await userManager.AddToRoleAsync(user, "Employee");
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine(">>> ПОМИЛКА ІНІЦІАЛІЗАЦІЇ: " + ex.Message);
}
}

app.Run();

internal class CustomAssemblyLoadContext : System.Runtime.Loader.AssemblyLoadContext
{
    public IntPtr LoadUnmanagedLibrary(string absolutePath)
    {
        return LoadUnmanagedDll(absolutePath);
    }
    protected override IntPtr LoadUnmanagedDll(string unmanagedDllName)
    {
        return LoadUnmanagedDllFromPath(unmanagedDllName);
    }
    protected override System.Reflection.Assembly Load(System.Reflection.AssemblyName
assemblyName)
    {
        return null;
    }
}

```

Б.2 Код контролерів управління кадровими процесами (Controllers)

Б.2.1 Код контролера HomeController.cs

```

using AppHR.Data;
using AppHR.Models;
using AppHR.Services;
using DocumentFormat.OpenXml.InkML;
using DocumentFormat.OpenXml.Spreadsheet;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Diagnostics;
using System.Security.Claims;

namespace AppHR.Controllers
{
    public class HomeController : Controller

```

```

{
    private readonly ILogger<HomeController> _logger;
    private readonly AppDbContext _context;
    private readonly ICamundaService _camundaService;

    public HomeController(ILogger<HomeController> logger, AppDbContext context,
        ICamundaService camundaService)
    {
        _logger = logger;
        _context = context;
        _camundaService = camundaService;
    }

    public async Task<ActionResult> Index()
    {
        var today = DateTime.Today;
        var dayOfYear = today.DayOfYear;
        var model = new DashboardViewModel();

        // --- ОТРИМУЄМО ДАНІ ПРО ПОТОЧНОГО КОРИСТУВАЧА ---
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var currentEmployee = await _context.Employees
            .Include(e => e.Department)
            .FirstOrDefaultAsync(e => e.UserId == currentUserId);

        // --- ОТРИМУЄМО ЗАДАЧІ САМЕ ДЛЯ ЦЬОГО МЕНЕДЖЕРА ---
        var tasks = await _camundaService.GetTasksForUserAsync(currentUserId);

        // --- ПЕРЕДАЄМО КІЛЬКІСТЬ У VIEW ---
        ViewBag.PendingTasksCount = tasks.Count();

        // --- ПЕРЕДАЄМО ПРАПОРЕЦЬ РОЛІ У VIEWMODEL ---
        model.IsAdmin = User.IsInRole("Admin") || User.IsInRole("HR");
        bool isManager = User.IsInRole("Manager");

        // --- СТАТИСТИКА (РІЗНА ЛОГІКА ДЛЯ РОЛЕЙ) ---
        if (model.IsAdmin)
        {
            model.TotalEmployees = await _context.Employees.CountAsync();
            model.PendingVacations = await _context.VacationRequests.CountAsync(v => v.Status ==
                VacationStatus.Pending);
        }
        else if (isManager && currentEmployee != null)
        {
            // --- МЕНЕДЖЕР БАЧИТЬ СВІЙ БАЛАНС ---
            model.TotalEmployees = currentEmployee.RemainingDaysFromLastYear;

            // --- МЕНЕДЖЕР БАЧИТЬ КІЛЬКІСТЬ ЗАЯВОК, ЯКІ ЧЕКАЮТЬ НА ЙОГО ПОГОДЖЕННЯ (СВІЙ ВІДДІЛ) --
            model.PendingVacations = await _context.VacationRequests

```

```

        .CountAsync(v => v.Employee.DepartmentId == currentEmployee.DepartmentId &&
v.Status == VacationStatus.Pending);

        // --- ДНІ НАРОДЖЕННЯ ВСЬОГО ВІДДІЛУ ---
        model.UpcomingBirthdays = await _context.Employees
            .Where(e => e.DepartmentId == currentEmployee.DepartmentId &&
                e.DateOfBirth.DayOfYear >= today.DayOfYear && e.DateOfBirth.DayOfYear <=
today.DayOfYear + 14)
            .ToListAsync();
    }
    else
    {
        // --- ДЛЯ ЗВИЧАЙНОГО КОРИСТУВАЧА ПОКАЗУЄМО ЙОГО ОСОБИСТІ ДАНІ ---
        model.TotalEmployees = currentEmployee?.RemainingDaysFromLastYear ?? 0;
        model.PendingVacations = await _context.VacationRequests
            .CountAsync(v => v.EmployeeId == currentEmployee.Id && v.Status ==
VacationStatus.Pending);
    }

    // --- ДНІ НАРОДЖЕННЯ (ФІЛЬТРАЦІЯ ЗА ВІДДІЛОМ ДЛЯ ЮЗЕРІВ) ---
    var allEmployees = await _context.Employees.ToListAsync();

    var birthdayQuery = allEmployees.AsEnumerable();

    if (!model.IsAdmin && currentEmployee != null)
    {
        // --- ЗВИЧАЙНИЙ ЮЗЕР БАЧИТЬ ДНІ НАРОДЖЕННЯ ТІЛЬКИ СВОГО ВІДДІЛУ ---
        birthdayQuery = birthdayQuery.Where(e => e.DepartmentId == currentEmployee.DepartmentId);
    }

    model.UpcomingBirthdays = birthdayQuery
        .Where(e => e.DateOfBirth.DayOfYear >= dayOfYear && e.DateOfBirth.DayOfYear <=
dayOfYear + 14)
        .OrderBy(e => e.DateOfBirth.DayOfYear)
        .ToList();

    return View(model);
}

public IActionResult Privacy()
{
    return View();
}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
}

```

```

    }
}
}

```

Б.2.2 Код контролера EmpoleeController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ClosedXML.Excel;
using System.IO;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;
using AppHR.Data;
using AppHR.Models;

namespace AppHR.Controllers
{
    [Authorize] // ---ТІЛЬКИ ДЛЯ ЗАЛОГІНЕНИХ КОРИСТУВАЧІВ ---
    public class EmployeesController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly IWebHostEnvironment _webHostEnvironment;
        public EmployeesController(ApplicationDbContext context)
        {
            _context = context;
        }

        // --- GET: EMPLOYEES ---
        public async Task<IActionResult> Index()
        {
            var employees = await _context.Employees
                .Include(e => e.Department)
                .Include(e => e.Position)
                .OrderBy(e => e.LastName)
                .ThenBy(e => e.FirstName)
                .ThenBy(e => e.MiddleName)

```

```

        .ToListAsync();

    return View(employees);
}

[Authorize(Roles = "Admin,HR")]
public async Task<IActionResult> ExportToExcel()
{
    var employees = await _context.Employees
        .Include(e => e.Department)
        .Include(e => e.Position)
        .ToListAsync();

    using (var workbook = new XLWorkbook())
    {
        var worksheet = workbook.Worksheets.Add("Співробітники");
        var currentRow = 1;

        // --- ЗАГОЛОВКИ ---
        worksheet.Cell(currentRow, 1).Value = "Прізвище";
        worksheet.Cell(currentRow, 2).Value = "Ім'я";
        worksheet.Cell(currentRow, 3).Value = "По батькові";
        worksheet.Cell(currentRow, 4).Value = "Відділ";
        worksheet.Cell(currentRow, 5).Value = "Посада";

        // --- ДАНІ ---
        foreach (var emp in employees)
        {
            currentRow++;
            worksheet.Cell(currentRow, 1).Value = emp.LastName;
            worksheet.Cell(currentRow, 2).Value = emp.FirstName;
            worksheet.Cell(currentRow, 3).Value = emp.MiddleName;
            worksheet.Cell(currentRow, 4).Value = emp.Department?.Name;
            worksheet.Cell(currentRow, 5).Value = emp.Position?.Title;
        }

        using (var stream = new MemoryStream())
        {
            workbook.SaveAs(stream);
            var content = stream.ToArray();

            return File(
                content,
                "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
                "Employees_Export.xlsx");
        }
    }
}

// --- GET: EMPLOYEES/DETAILS/5 ---

```

```

public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var employee = await _context.Employees
        .Include(e => e.Documents)
        .Include(e => e.Position)
        .Include(e => e.Department)
        .FirstOrDefaultAsync(m => m.Id == id);

    if (employee == null)
    {
        return NotFound();
    }

    return View(employee);
}

// --- GET: EMPLOYEES/CREATE ---
public IActionResult Create()
{
    // Завантажуємо дані з бази для випадваючих списків
    ViewData["DepartmentId"] = new SelectList(_context.Departments, "Id", "Name");
    ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Title");
    return View();
}

// --- POST: EMPLOYEES/CREATE ---
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin,HR")]
public async Task<IActionResult>
Create([Bind("Id,FirstName,LastName,MiddleName,Email,Phone,HireDate,PositionId,DepartmentId,IsActive")] Employee employee)
{
    if (ModelState.IsValid)
    {
        _context.Add(employee);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(employee);
}

// --- GET: EMPLOYEES/EDIT/5 ---
[Authorize(Roles = "Admin,HR")]

```

```

public async Task<IActionResult> Edit(int? id)
{
    if (id == null) return NotFound();

    var employee = await _context.Employees.FindAsync(id);
    if (employee == null) return NotFound();
    // Отримуємо список всіх зареєстрованих користувачів
    var users = _context.Users.Select(u => new { u.Id, u.Email }).ToList();
    ViewData["UserId"] = new SelectList(users, "Id", "Email", employee.UserId);
    // Передаємо існуючі значення у списки
    ViewData["DepartmentId"] = new SelectList(_context.Departments, "Id", "Name",
employee.DepartmentId);
    ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Title", employee.PositionId);
    return View(employee);
}

// --- POST: EMPLOYEES/EDIT/5 ---
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Edit(int id,
[Bind("Id,FirstName,LastName,MiddleName,Email,Phone,HireDate,PositionId,DepartmentId,IsActive")]
Employee employee)
{
    if (id != employee.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            var employeeInDb = await _context.Employees.FindAsync(id);

            if (employeeInDb == null)
            {
                return NotFound();
            }

            // --- ОНОВЛЮЄМО ПОЛЯ ОБ'ЄКТА З БАЗИ ДАНИМИ, ЩО ПРИЙШЛИ З ФОРМИ --
            employeeInDb.FirstName = employee.FirstName;
            employeeInDb.LastName = employee.LastName;
            employeeInDb.MiddleName = employee.MiddleName;
            employeeInDb.Email = employee.Email;
            employeeInDb.Phone = employee.Phone;
            employeeInDb.HireDate = employee.HireDate;
            employeeInDb.PositionId = employee.PositionId;
            employeeInDb.DepartmentId = employee.DepartmentId;

```

```

        employeeInDb.IsActive = employee.IsActive;

        // --- ЗБЕРІГАЄМО. ONBEFORESAVECHANGES ТЕПЕР ПОБАЧИТЬ РІЗНИЦЮ ---
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!EmployeeExists(employee.Id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
return View(employee);
}
// GET: Employees/Delete/5
// Тільки адмін може бачити сторінку видалення та видаляти
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    if (id == null) return NotFound();

    var employee = await _context.Employees
        .Include(e => e.Department) // Додаємо це
        .Include(e => e.Position) // І це
        .FirstOrDefaultAsync(m => m.Id == id);

    if (employee == null) return NotFound();

    return View(employee);
}

// --- POST: EMPLOYEES/DELETE/5 ---
[HttpPost, ActionName("Delete")]
[Authorize(Roles = "Admin")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var employee = await _context.Employees.FindAsync(id);

```

```

    if (employee != null)
    {
        _context.Employees.Remove(employee);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool EmployeeExists(int id)
{
    return _context.Employees.Any(e => e.Id == id);
}

public async Task<Employee?> FindApprover(int employeeId)
{
    // --- БЕРЕМО СПІВРОБІТНИКА ТА ЙОГО ВІДДІЛ ---
    var employee = await _context.Employees
        .Include(e => e.Department)
        .FirstOrDefaultAsync(e => e.Id == employeeId);

    if (employee == null || employee.DepartmentId == null) return null;

    var currentDept = await _context.Departments
        .Include(d => d.Manager)
        .FirstOrDefaultAsync(d => d.Id == employee.DepartmentId);

    // --- ЦИКЛ ПОШУКУ ВГОРУ ПО ІЄРАХІЇ ---
    while (currentDept != null)
    {
        // --- ЯКЩО В ПОТОЧНОМУ ВІДДІЛІ Є МЕНЕДЖЕР І ЦЕ НЕ САМ ЗАЯВНИК — ВІН ПОГОДЖУВАЧ ---
        if (currentDept.ManagerId != null && currentDept.ManagerId != employeeId)
        {
            return currentDept.Manager;
        }

        // --- ЯКЩО МЕНЕДЖЕРА НЕМАЄ, ЙДЕМО В БАТЬКІВСЬКИЙ ВІДДІЛ ---
        if (currentDept.ParentDepartmentId != null)
        {
            currentDept = await _context.Departments
                .Include(d => d.Manager)
                .FirstOrDefaultAsync(d => d.Id == currentDept.ParentDepartmentId);
        }
        else
        {
            break;
        }
    }

    return null;
}

```

```

    }

    [HttpPost]
    public async Task<IActionResult> UploadDocument(int employeeId, string documentType,
    IFormFile file)
    {
        if (file == null || file.Length == 0) return RedirectToAction("Details", new { id = employeeId });

        // --- СТВОРЮЄМО ШЛЯХ ДО ПАПКИ ---
        string uploadsFolder = Path.Combine(_webHostEnvironment.WebRootPath, "uploads",
        "documents");
        if (!Directory.Exists(uploadsFolder)) Directory.CreateDirectory(uploadsFolder);

        // --- ГЕНЕРУЄМО УНІКАЛЬНЕ ІМ'Я ФАЙЛУ, ЩОБ УНИКНУТИ ДУБЛІКАТІВ ---
        string uniqueFileName = $"{employeeId}_{Guid.NewGuid()}_{file.FileName}";
        string filePath = Path.Combine(uploadsFolder, uniqueFileName);

        // --- ЗБЕРІГАЄМО ФАЙЛ ФІЗИЧНО ---
        using (var fileStream = new FileStream(filePath, FileMode.Create))
        {
            await file.CopyToAsync(fileStream);
        }

        // --- ЗАПИСУЄМО ДАНІ В БД ---
        var doc = new EmployeeDocument
        {
            EmployeeId = employeeId,
            DocumentType = documentType,
            FileName = file.FileName,
            FilePath = "/uploads/documents/" + uniqueFileName,
            UploadDate = DateTime.Now
        };

        _context.Documents.Add(doc);
        await _context.SaveChangesAsync();

        return RedirectToAction("Details", new { id = employeeId });
    }
}
}

```

Б.3 Код інтеграційного сервісу взаємодії з оркестратором (CamundaService.cs)

```

using Camunda.Api.Client;
using Camunda.Api.Client.ProcessDefinition;
using Camunda.Api.Client.UserTask;
using System.Collections.Generic;
using AppHR.Models;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using AppHR.Data;
using System.Threading.Tasks;
using System;

namespace AppHR.Services
{
    public class CamundaService : ICamundaService
    {
        private readonly CamundaClient _camundaClient;
        private readonly AppDbContext _context;
        private readonly ILogger<CamundaService> _logger; // Поле для логування

        // --- ВИПРАВЛЕНИЙ КОНСТРУКТОР З ІН'ЄКЦІЄЮ ILOGGER ---
        public CamundaService(AppDbContext context, ILogger<CamundaService> logger)
        {
            _context = context;
            _logger = logger;
            _camundaClient = CamundaClient.Create("http://localhost:8080/engine-rest");
        }

        public async Task<string> StartVacationProcess(
            int requestId,
            string employeeName,
            string managerId,
            string employeeId,
            bool isApprovalRequired,
            int timeoutDays,
            string vacationType,
            DateTime startDate,
            DateTime endDate
        )
        {
            // --- СТВОРЮЄМО ПАРАМЕТРИ ЗАПУСКУ ТА BUSINESSKEY ---
            var startParams = new StartProcessInstance()
            {
                BusinessKey = requestId.ToString()
            };

            // --- ДОДАЄМО ЗМІННІ ДЛЯ ЛОГІКИ GATEWAYS ---
            startParams.Variables.Add("needsApproval", VariableValue.FromObject(isApprovalRequired));

            // --- НАЛАШТУВАННЯ ТАЙМАУТУ ДЛЯ ТАЙМЕРА НА СХЕМІ ---
            string isoTimeout = $"P{timeoutDays}D";

```

```

startParams.Variables.Add("managerTimeout", VariableValue.FromObject(isoTimeout));

// --- ПРИЗНАЧЕННЯ ВИКОНАВЦІВ ---
startParams.Variables.Add("managerAssignee", VariableValue.FromObject(managerId));
startParams.Variables.Add("employeeAssignee", VariableValue.FromObject(employeeId));
startParams.Variables.Add("hr_department", VariableValue.FromObject("hr_department"));

// --- ПЕРЕДАЧА ДЕТАЛЬНОЇ ІНФОРМАЦІЇ ДЛЯ ВІДОБРАЖЕННЯ В ТАБЛИЦІ ЗАВДАНЬ ---
startParams.Variables.Add("requestId", VariableValue.FromObject(requestId));
startParams.Variables.Add("employeeName", VariableValue.FromObject(employeeName));
startParams.Variables.Add("vacationType", VariableValue.FromObject(vacationType));

// --- ФОРМАТУЄМО ДАТИ В РЯДКИ, ЩОБ ЇХ БУЛО ЗРУЧНО ЧИТАТИ У VIEW ---
startParams.Variables.Add("startDate",
VariableValue.FromObject(startDate.ToString("dd.MM.yyyy")));
startParams.Variables.Add("endDate",
VariableValue.FromObject(endDate.ToString("dd.MM.yyyy")));

// --- ФОРМУЄМО ТЕХНІЧНИЙ ОПИС ЗАВДАННЯ ---
string description = $"Запит №{requestId} на {vacationType}: {employeeName}";
startParams.Variables.Add("taskDescription", VariableValue.FromObject(description));

// --- ЗАПУСК ПРОЦЕСУ В CAMUNDA ---
var processInstance = await _camundaClient.ProcessDefinitions
    .ByKey("VacationProcess")
    .StartProcessInstance(startParams);

return processInstance.Id;
}

public async Task<List<CamundaTaskDto>> GetTasksForUserAsync(string userId)
{
// --- ОТРИМУЄМО ЗАВДАННЯ ---
var personalTasks = await _camundaClient.UserTasks
    .Query(new TaskQuery { Assignee = userId }).List();

var employee = await _context.Employees
    .Include(e => e.Department)
    .FirstOrDefaultAsync(e => e.UserId == userId);

List<UserTaskInfo> groupTasks = new List<UserTaskInfo>();

if (employee?.Department?.Name != null && employee.Department.Name.Contains("HR"))
{
    groupTasks = await _camundaClient.UserTasks
        .Query(new TaskQuery { CandidateGroup = "hr_department" }).List();
}

var allRawTasks = personalTasks.Concat(groupTasks)
    .GroupBy(t => t.Id)

```

```

.Select(g => g.First())
.ToList();

var taskDtos = new List<CamundaTaskDto>();

// --- ОБРОБКА КОЖНОГО ЗАВДАННЯ ---
foreach (var t in allRawTasks)
{
    // --- ОТРИМУЄМО ВСІ ЗМІННІ ПРОЦЕСУ ДЛЯ ЦЬОГО ЗАВДАННЯ ---
    var camundaVariables = await _camundaClient.UserTasks[t.Id].Variables.GetAll();

    var dto = new CamundaTaskDto
    {
        Id = t.Id,
        Name = t.Name,
        ProcessInstanceId = t.ProcessInstanceId,
        CreationTime = t.Created,
        Assignee = t.Assignee,
        TaskDefinitionKey = t.TaskDefinitionKey,

        EmployeeName = camundaVariables.ContainsKey("employeeName")
            ? camundaVariables["employeeName"].Value.ToString() : "Не вказано",

        Description = camundaVariables.ContainsKey("taskDescription")
            ? camundaVariables["taskDescription"].Value.ToString() : "Запит на відпустку",

        VacationType = camundaVariables.ContainsKey("vacationType")
            ? camundaVariables["vacationType"].Value.ToString() : "Не вказано",

        StartDate = camundaVariables.ContainsKey("startDate")
            ? camundaVariables["startDate"].Value.ToString() : "-",

        EndDate = camundaVariables.ContainsKey("endDate")
            ? camundaVariables["endDate"].Value.ToString() : "-",

        ManagerApproved = camundaVariables.ContainsKey("managerApproved")
            && (bool)camundaVariables["managerApproved"].Value
    };

    foreach (var variable in camundaVariables)
    {
        dto.Variables.Add(variable.Key, new CamundaVariableDto
        {
            Value = variable.Value.Value,
            Type = variable.Value.Type.ToString()
        });
    }

    taskDtos.Add(dto);
}

```

```

    }

    return taskDtos;
}
public async Task CompleteTask(string taskId, bool approved, string comment = "")
{
    var taskResource = _camundaClient.UserTasks[taskId];
    var taskVariables = await taskResource.Variables.GetAll();

    // --- ОТРИМУЄМО REQUESTID ДЛЯ ПОШУКУ ВІДПОВІДНОГО ЗАПИСУ В SQL БАЗІ ---
    if (!taskVariables.ContainsKey("requestId")) return;
    int requestId = int.Parse(taskVariables["requestId"].Value.ToString());

    // --- ЗАВЕРШУЄМО ЗАВДАННЯ В CAMUNDA, ПЕРЕДАЮЧИ РІШЕННЯ (APPROVED) ---
    var request = new CompleteTask();
    request.Variables.Add("approved", VariableValue.FromObject(approved));

    // --- ЯКЩО Є КОМЕНТАР (НАПРИКЛАД, ПРИЧИНА ВІДХИЛЕННЯ), ДОДАЄМО ЙОГО В ЗМІННІ ПРОЦЕСУ---
    if (!string.IsNullOrEmpty(comment))
    {
        request.Variables.Add("comment", VariableValue.FromObject(comment));
    }

    await taskResource.Complete(request);

    // --- АВТОМАТИЧНО ОНОВЛЮЄМО СТАТУС У БАЗІ ДАНИХ ДЛЯ СИНХРОНІЗАЦІЇ З UI ---
    var vacationRequest = await _context.VacationRequests.FindAsync(requestId);
    if (vacationRequest != null)
    {
        if (!approved)
        {
            vacationRequest.Status = VacationStatus.Rejected;
        }
        else
        {
            // --- ВИЗНАЧАЄМО НОВИЙ СТАТУС ЗАЛЕЖНО ВІД ТОГО, ЯКЕ САМЕ ЗАВДАННЯ БУЛО ЗАВЕРШЕНО ---
            var task = await taskResource.Get();
            vacationRequest.Status = task.TaskDefinitionKey switch
            {
                // --- ПІСЛЯ ПОГОДЖЕННЯ МЕНЕДЖЕРОМ ЗАПИТ СТАЄ НА РОЗГЛЯДІ---
                "UserTask_ManagerApproval" => VacationStatus.Pending,

                // --- ПІСЛЯ ПОГОДЖЕННЯ HR ЗАПИТ ОТРИМУЄ СТАТУС ПОГОДЖЕНО ---
                "UserTask_HRReview" => VacationStatus.Approved,

                _ => vacationRequest.Status
            };
        }
    }

    // --- ЗБЕРІГАЄМО ЗМІНИ В SQL SERVER ---
    await _context.SaveChangesAsync();
}

```

```

    }
}
public async Task DeleteProcessInstanceAsync(string processInstanceId)
{
    if (string.IsNullOrEmpty(processInstanceId)) return;

    try
    {
        // --- ВИКОРИСТОВУЄМО ІСНУЮЧИЙ КЛІЄНТ ЗАМІСТЬ НЕВІДОМОГО HTTPCLIENT ---
        await _camundaClient.ProcessInstances[processInstanceId].Delete();
        _logger.LogInformation($"Процес {processInstanceId} видалено.");
    }
    catch (Exception ex)
    {
        _logger.LogError($"Помилка видалення процесу: {ex.Message}");
    }
}

public async Task<List<TaskViewModel>> GetActiveTasksAsync()
{
    var tasks = await _camundaClient.UserTasks.Query(new TaskQuery { Active = true }).List();
    return tasks.Select(t => new TaskViewModel
    {
        TaskId = t.Id,
        Name = t.Name,
        Created = t.Created,
        ProcessInstanceId = t.ProcessInstanceId
    }).ToList();
}

public async Task FinalizeRequest(string processInstanceId)
{
    // --- ЗНАХОДИМО ЗАПИТ ЗА ТЕХНІЧНИМ ID ПРОЦЕСУ CAMUNDA ---
    var request = await _context.VacationRequests
        .FirstOrDefaultAsync(r => r.CamundaProcessInstanceId == processInstanceId);

    if (request != null)
    {
        // --- ВСТАНОВЛЮЄМО ФІНАЛЬНИЙ ЮРИДИЧНИЙ СТАТУС ---
        request.Status = VacationStatus.Ordered;

        await _context.SaveChangesAsync();
        _logger.LogInformation($"Запит {request.Id} успішно переведено в статус Ordered.");
    }
}
}
}
}

```

Б.4 Класи моделей даних та полегшених об'єктів передачі даних (DTOs)

Б.4.1 Клас Employee.cs

```

using System;
using System.ComponentModel.DataAnnotations;
namespace AppHR.Models
{
    public class Employee
    {
        public int Id { get; set; }

        [Display(Name = "Прізвище")]
        public string LastName { get; set; }

        [Display(Name = "Ім'я")]
        public string FirstName { get; set; }

        [Display(Name = "По батькові")]
        public string MiddleName { get; set; }

        [Display(Name = "Співробітник")]
        public string FullName => $"{LastName} {FirstName} {MiddleName}".Trim();

        [Display(Name = "Електронна пошта")]
        public string Email { get; set; }

        [Display(Name = "Телефон")]
        public string Phone { get; set; }

        [Display(Name = "Працює з")]
        [DataType(DataType.Date)]

        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime HireDate { get; set; } = DateTime.Now;

        [Display(Name = "Відділ")]
        public int DepartmentId { get; set; }
        public Department? Department { get; set; }

        [Display(Name = "Посада")]
        public int PositionId { get; set; }
        public Position? Position { get; set; }

        [Display(Name = "Статус")]
        public bool IsActive { get; set; } = true;
        public DateTime DateOfBirth { get; set; }

        // --- ДНІ, ЩО ЗАЛИШИЛИСЯ З МИНУЛИХ РОКІВ ---
        public int RemainingDaysFromLastYear { get; set; } = 0;
    }
}

```

```
// --- ІНДИВІДУАЛЬНЕ НАЛАШТУВАННЯ (ПРІОРИТЕТ НАД ГЛОБАЛЬНИМ) ---
public bool? CanTakeExtraDays { get; set; }

public bool AllowNegativeBalance { get; set; } = false;

public string? UserId { get; set; }
public virtual ICollection<EmployeeDocument> Documents { get; set; } = new
List<EmployeeDocument>();
}

public class Position
{
    public int Id { get; set; }
    public string Title { get; set; }
}
public class CreateButtonModel
{
    public string Action { get; set; } = "Create";
    public string Text { get; set; } = "Додати";
    public string Icon { get; set; } = "fa-plus";
    public string Class { get; set; } = "btn-primary";
}
}
```

Б.4.2 Клас VacationRequest.cs

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
namespace AppHR.Models
{
    public enum VacationStatus
    {
        Created = 0,    ///--- ЩОЙНО СТВОРЕНА ---
        Pending = 1,    // --- НА РОЗГЛЯДІ ----
        Approved = 2,   // --- ПОГОДЖЕНА МЕНЕДЖЕРОМ ---
        Rejected = 3,   // --- ВІДХИЛЕНА (З ПРИЧИНОЮ) ---
        Ordered = 4,    // --- ВИДАНО ОФІЦІЙНИЙ НАКАЗ ---
        Cancelled = 5   // --- СКАСОВАНА ПІСЛЯ ВИДАННЯ НАКАЗУ ---
    }
    public class VacationRequest
    {
        public int Id { get; set; }

        [Display(Name = "Співробітник")]
        public int EmployeeId { get; set; }

        [ValidateNever]
    }
}
```

```

public Employee? Employee { get; set; }

public int VacationTypeId { get; set; }

[ValidateNever]
public VacationType? VacationType { get; set; }

[Required, Display(Name = "Дата початку")]
[DataType(DataType.Date)]
[DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}", ApplyFormatInEditMode = true)]
public DateTime StartDate { get; set; }

[Required, Display(Name = "Дата завершення")]
[DataType(DataType.Date)]
[DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}", ApplyFormatInEditMode = true)]
public DateTime EndDate { get; set; }

public DateTime CreatedAt { get; set; } = DateTime.Now;
public string? InstanceId { get; set; }

[Display(Name = "Статус")]
public VacationStatus Status { get; set; } = VacationStatus.Created;

public string? CamundaProcessInstanceId { get; set; }

public string? DeclineReason { get; set; }

public string? OrderNumber { get; set; }

}
}

```

Б.4.2 Клас передачі даних CamundaTaskDto.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace AppHR.Models
{
    public class CamundaTaskDto
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public string ProcessInstanceId { get; set; }
        public DateTime CreationTime { get; set; }
    }
}

```

```

public string Assignee { get; set; }

[Display(Name = "Співробітник")]
public string EmployeeName { get; set; }

public string Description { get; set; }
public string TaskDefinitionKey { get; set; }
public string VacationType { get; set; }
public string StartDate { get; set; }
public string EndDate { get; set; }
public string Comment { get; set; }
public bool ManagerApproved { get; set; }

public Dictionary<string, CamundaVariableDto> Variables { get; set; } = new Dictionary<string,
CamundaVariableDto>();
}

// --- ДОПОМІЖНИЙ КЛАС, ЩОБ С# РОЗУМІВ СТРУКТУРУ ЗМІННОЇ CAMUNDA ---
public class CamundaVariableDto
{
    public object Value { get; set; }
    public string Type { get; set; }
}
}

```

Б.5 Клас контролера DocumentTemplatesController.cs та реалізація модуля DbInitializer.cs

Б.5.1 Клас контролера DocumentTemplatesController.cs

```

using AppHR.Data;
using AppHR.Models;
using global::AppHR.Data;
using global::AppHR.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace AppHR.Controllers
{
    [Authorize(Roles = "Admin")]
    public class DocumentTemplatesController : Controller
    {
        private readonly AppDbContext _context;

        public DocumentTemplatesController(AppDbContext context)

```

```

    {
        _context = context;
    }

// --- СПИСОК ШАБЛОНОВ ---
public async Task<IActionResult> Index()
{
    return View(await _context.DocumentTemplates.ToListAsync());
}

// --- GET: DOCUMENTTEMPLATES/CREATE ---
public IActionResult Create()
{
    return View();
}

//--- POST: DOCUMENTTEMPLATES/CREATE ---
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Name,DocumentType,Content,IsActive")]
DocumentTemplate documentTemplate)
{
    if (ModelState.IsValid)
    {
        if (documentTemplate.IsActive)
        {
            var others = await _context.DocumentTemplates
                .Where(t => t.DocumentType == documentTemplate.DocumentType)
                .ToListAsync();
            foreach (var t in others) t.IsActive = false;
        }

        documentTemplate.LastModified = DateTime.Now;
        _context.Add(documentTemplate);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(documentTemplate);
}

// --- РЕДАГУВАННЯ (GET) ---
public async Task<IActionResult> Edit(int? id)
{
    if (id == null) return NotFound();

    var template = await _context.DocumentTemplates.FindAsync(id);
    if (template == null) return NotFound();

    return View(template);
}

```

```

// --- РЕДАГУВАННЯ (POST) --
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,DocumentType,Content,IsActive")] DocumentTemplate documentTemplate)
{
    if (id != documentTemplate.Id) return NotFound();

    if (ModelState.IsValid)
    {
        try
        {
            if (documentTemplate.IsActive)
            {
                var otherTemplates = await _context.DocumentTemplates
                    .Where(t => t.DocumentType == documentTemplate.DocumentType && t.Id != id)
                    .ToListAsync();

                foreach (var t in otherTemplates)
                {
                    t.IsActive = false;
                }
            }

            documentTemplate.LastModified = DateTime.Now;
            _context.Update(documentTemplate);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!TemplateExists(documentTemplate.Id)) return NotFound();
            else throw;
        }
        return RedirectToAction(nameof(Index));
    }
    return View(documentTemplate);
}

public async Task<IActionResult> Details(int? id)
{
    if (id == null) return NotFound();

    var template = await _context.DocumentTemplates
        .FirstOrDefaultAsync(m => m.Id == id);

    if (template == null) return NotFound();

    return View(template);
}

```

```

private bool TemplateExists(int id)
{
    return _context.DocumentTemplates.Any(e => e.Id == id);
}

}

```

Б.5.2 Реалізація модуля DbInitializer.cs

```

using AppHR.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Linq;

namespace AppHR.Data
{
    public static class DbInitializer
    {
        public static void Initialize(AppDbContext context)
        {
            context.Database.EnsureCreated();

            if (!context.VacationTypes.Any())
            {
                context.VacationTypes.AddRange(
                    new VacationType { Name = "Щорічна основна", DeductsFromBalance = true,
RequiresOrder = true },
                    new VacationType { Name = "За власний рахунок", DeductsFromBalance = false,
RequiresOrder = true },
                    new VacationType { Name = "Лікарняний", DeductsFromBalance = false, RequiresOrder =
false }
                );
                context.SaveChanges();
            }

            // --- ІНІЦІАЛІЗАЦІЯ ШАБЛОНІВ ДОКУМЕНТІВ ---
            if (!context.DocumentTemplates.Any())
            {
                context.DocumentTemplates.AddRange(
                    new DocumentTemplate
                    {
                        Name = "Стандартна заява на відпустку",
                        DocumentType = "Vacation",
                        IsActive = true,
                        LastModified = DateTime.Now,

```

```

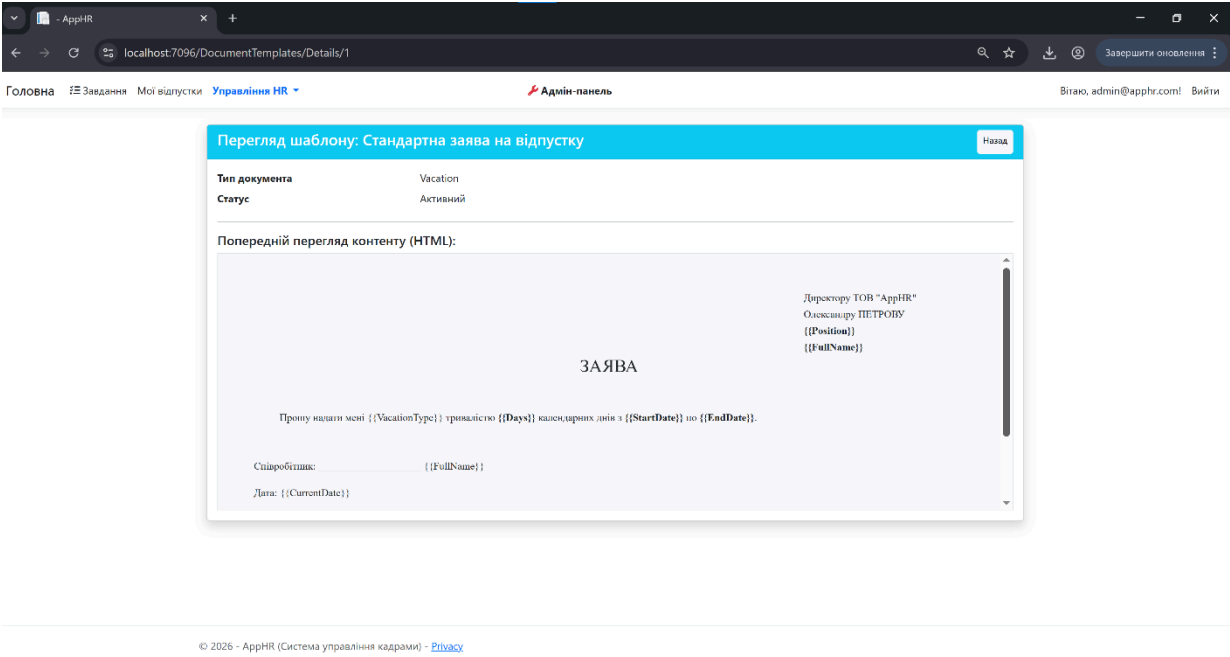
        Content = @"<div style='font-family: ""Times New Roman""; padding: 30px;'>
            <div style='text-align: right;'>Директору ТОВ
"" AppHR ""<br>{{Position}}<br>{{FullName}}</div>
            <h3 style='text-align: center; margin-top: 50px;'>ЗАЯВА</h3>
            <p>Прошу надати мені відпустку на {{Days}} календарних днів з {{StartDate}}
по {{EndDate}}.</p>
            <br><br><span>{{CurrentDate}}</span><span style='float: right;'>_____
(підпис)</span>
        </div>"
    },
    new DocumentTemplate
    {
        Name = "Наказ про надання відпустки",
        DocumentType = "VacationOrder",
        IsActive = true,
        LastModified = DateTime.Now,
        Content = @"<div style='text-align: center; font-family: sans-serif;'>
            <h2>НАКАЗ № {{OrderNumber}}</h2>
            <p>від {{CurrentDate}}</p>
            <br>
            <p>НАДАТИ: <b>{{FullName}}</b> ({{Position}})</p>
            <p>Відпустку терміном на {{Days}} днів з {{StartDate}} по {{EndDate}}.</p>
        </div>"
    }
    );
    context.SaveChanges();
}
}
}
}
}

```

ДОДАТОК В

Зразки вихідних документів

В.1 Шаблон заявки на відпустку



В.2 Приклад друкованої форми заявки на відпустку

Директору ТОВ "AppHR"
Олександру ПЕТРОВУ
**Інженер технічної
підтримки(Support Engineer)**
Хоменко Руслан Михайлович

ЗАЯВА

Пропу надати мені за власний рахунок тривалістю **1** календарних днів з **26.04.2026** по **26.04.2026**.

Співробітник: _____ Хоменко Руслан Михайлович

Дата: 08.05.2026

ТОВ "AppHR"
Код ЄДРПОУ: 12345678


Скануйте для перевірки

В.3 Приклад друкованої форми наказу на відпустку

ТОВ "APPHR"

НАКАЗ № 28-В

08.05.2026 р.

Про надання відпустки

НАКАЗУЮ:

Надати **Хоменко Руслан Михайлович** що займає посаду **Інженер технічної підтримки(Support Engineer)**, щорічна основна відпустку тривалістю **5** календарних днів з **11.05.2026** по **15.05.2026**.

Директор ТОВ "AppHR"
(підпис)

Олександр ПЕТРОВ

З наказом ознайомлений: _____ Хоменко Руслан Михайлович

ТОВ "AppHR"

Код ЄДРПОУ: 12345678



Скануйте для перевірки